University of Verona

Department of Computer Science

Graduate School of Natural Sciences and Engineering

Doctoral Program in Computer science

Cycle xxxviii

Advanced Neural Networks Verification for Safe and Explainable Intelligent Systems

Ph.D. Candidate:
Luca Marzari

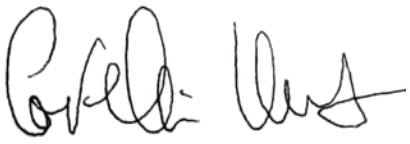
Coordinator: Prof. Umberto Castellani

Advisor: Prof. Alessandro Farinelli

Co-Advisor: Prof. Ferdinando Cicalese

S.S.D. INF/01

Advanced Neural Networks Verification for Safe and Explainable Intelligent Systems
Luca Marzari
Ph.D. Thesis
Verona, Italy

Coordinator: 

Umberto Castellani

Advisor: 

Alessandro Farinelli

Co-Advisor: 

Ferdinando Cicalese

Acknowledgements

I want to start by thanking my advisors, Alessandro Farinelli and Ferdinando Cicalese. Thank you, Nando and Alessandro, for the hours we spent together at the blackboard, talking and working through ideas that became a big part of this thesis. Those moments are memories I will always remember with great pleasure.

I would like to thank Changliu Liu for her kind and helpful guidance during my research visit at CMU. I am also very grateful to Francesco Leofante and everyone in the ISLa lab, especially Francesco, Daniele, Celeste and Veronica, for their support and for patiently enduring my constant complaints throughout this challenging journey. A special thanks goes to "RL boy" Enrico Marchesini, who guided me during my first year of the PhD, becoming not only a fantastic colleague but also, and most importantly, a great friend.

A special thanks (with equal contribution :D) goes to my wonderful parents, Anna and Giuliano. Thank you for your endless patience, your kind words, and your constant presence. To Stefano, Adriana, and Giulia, my second family, thank you for welcoming me into your lives and for all your support. This thesis has also been possible thanks to you.

To my lifelong friends, Marco, Nadir, Lucio, and Pietro, who helped me maintain mental health when it was clear that I was losing it, thank you.

Last but not least, Anna, the love of my life. Words could never fully express how much you mean to me, so I will simply say: thank you.
This thesis is for you.

Declaration of Authorship

I, Luca Marzari, declare that this thesis titled "Advanced Neural Networks Verification for Safe and Explainable Intelligent Systems" and the works presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University;
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- Where I have consulted the published work of others, this is always clearly attributed;
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- During the preparation of this work, I used *ChatGPT* and *Grammarly* AI tools in order to grammar and spelling check, paraphrase, and reword. After using these tools, I reviewed and edited the content as needed. I took full responsibility for the publication's content.
- I have acknowledged all main sources of help.

Signed: Luca Marzari

Date: 19/02/2026

There are two people in a wood, and they run into a bear. The first person gets down on his knees to pray; the second person starts lacing up his boots. The first person asks the second person, "My dear friend, what are you doing? You can't outrun a bear." To which the second person responds, "I don't have to. I only have to outrun you."

– The Imitation Game

Abstract

Deep neural networks (DNNs) have revolutionized artificial intelligence (AI), enabling breakthroughs across domains ranging from medical robotics to sustainable energy management. Yet, their fragility to adversarial perturbations and their opacity as "black boxes" pose significant risks when deployed in safety-critical contexts.

In this thesis, we address these challenges by advancing the field of neural network verification, with a particular focus on enhancing the safety and explainability of intelligent systems. A first set of contributions develops novel verification techniques that address the limitations of existing approaches. Through abstract interpretation and probabilistic reasoning, this work introduces scalable methods that not only provide sound guarantees but also extend verification to richer, semantically grounded safety properties. In particular, the introduction of the #DNN-Verification and AllDNN-Verification problems establishes new theoretical foundations that enable quantifying and localizing unsafe regions within the input space, thereby linking verification with interpretability.

Building upon these theoretical advances, this work demonstrates how verification can be effectively integrated into deep reinforcement learning (DRL) scenarios. From post-training verification for model selection to verification-informed training, we show how safety assurances can guide policy optimization in safety-critical applications such as robotic navigation and medical procedures. Crucially, these methods further allow the automatic collection of task-level safety properties, reducing reliance on brittle, hand-designed specifications.

Finally, this thesis extends the role of verification toward explainability by providing the first rigorous complexity analysis of generating robust Counterfactual Explanations (CEs). Leveraging probabilistic verification methods, we propose new algorithms that produce counterfactuals with provable probabilistic robustness guarantees, ensuring explanations that remain valid after fine-tuning the model.

Together, these contributions establish verification as a unifying tool at the intersection of safety and explainability of intelligent systems. By bridging theory with practical deployment in the real world, this thesis advances the vision of trustworthy, transparent, and reliable AI systems.

Contents

List of Symbols

Neural Network

$d \in \mathbb{R}^n$	Input space dimension.
$\boldsymbol{x} \in \mathbb{R}^d$	Generic input vector.
$f : \mathbb{R}^d \to \mathbb{R}$	Neural Network (NN) function.
N	Number of layers in the DNN.
d_i	Number of nodes in the layer $i = 1 \ldots, N$.
$z_j^{(i)}(\boldsymbol{x})$	j-th pre-activated node value in the i-th layer.
$\hat{z}_j^{(i)}(\boldsymbol{x}) = \sigma(z_j^{(i)}(\boldsymbol{x}))$	j-th activated node value in the i-th layer using the activation function σ.
$\boldsymbol{z}^{(i)}(\boldsymbol{x}) = (z_1^{(i)}(\boldsymbol{x}), \ldots, z_{d_i}^{(i)}(\boldsymbol{x}))$	pre-activated node vector in the i-th layer.
$\hat{\boldsymbol{z}}^{(i)}(\boldsymbol{x})$	Activated node vector in the i-th layer using σ.
$\boldsymbol{W}^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$	Inter level weight matrix.
$\boldsymbol{b}^{(i)} \in \mathbb{R}^{d_i}$	Inter level bias vector.
$f(\boldsymbol{x})$	Output value of the Neural Network.
$\boldsymbol{W}$	NN's Weights matrix.
$\boldsymbol{b}$	NN's bias matrix.
η	Learning rate that controls the magnitude of each update in the training of a NN.
$\mathcal{L}$	Generic loss function.
$\mathbb{O} := [1, n] \times \mathbb{R}$	Index and the node with the greatest value in the output vector.
$g : \mathbb{R}^n \to \mathbb{O}$	Function that returns the index and the node with the greatest value in the output vector of a DNN.

(Constrained) Reinforcement Learning

$\mathcal{S}$	State space.
$\mathcal{A}$	Action space.
s_t	State at timestep t.
a_t	Action at timestep t.
$P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$	State transition probability distribution.
$\rho : \mathcal{S} \to [0, 1]$	Initial uniform state distribution.
$R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$	Reward function.
$\gamma \in [0, 1)$	Discount factor.
$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$	RL policy.

$\tau := (s_0, a_0, s_1, a_1, \dots)$	Trajectory.
$\zeta(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$	Expected discounted reward.
$V_\pi : \mathcal{S} \to \mathbb{R}$	State Value function.
$Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$	Action Value function.
$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$	Advantage function.
D_{KL}	Kullback–Leibler divergence statistical distance.
$\alpha = D_{KL}^{\max}(\pi, \pi')$	Max KL divergence between two policies.
$C : \mathcal{S} \times \mathcal{A} \to \{0, 1\}$	Cost function.
$\zeta_C(\pi) := \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t C(s_t, a_t)\right]$	Expected value cost function.
c	Cost threshold.
λ	Lagrangian multiplier.
$\mathbf{P}$	Safety properties buffer.
$\mathcal{P}$	DRL safety property.
β	Similarity threshold to merge two safety properties.
ω	Area size around an unsafe state to generate the safety property.
d_{unsafe}	Smallest interval encoding a potentially unsafe situation.
Neural Network Verification	
$\mathcal{X}$	Safety property's pre-condition.
$\mathcal{Y}$	Safety property's post-condition.
$\langle \mathcal{X}, \mathcal{Y} \rangle$	Safety property for DNN-Verification.
$\mathcal{R}(\mathcal{X}, f)$	Over-approximated output reachable set.
$\mathcal{P}$	Input perturbation region.
$[l_j^{(i)}, u_j^{(i)}]$	Intermediate reachable set for the j-th node in the ith layer.
$\underline{\boldsymbol{A}}^{(i)}$	Linearization coefficients vector i-th layer for lower bound computation.
$\underline{\boldsymbol{D}}^{(i)}$	Linearization coefficients diagonal matrix i-th layer for lower bound computation.
$\overline{\boldsymbol{A}}^{(i)}$	Linearization coefficients vector i-th layer for upper bound computation.
$\overline{\boldsymbol{D}}^{(i)}$	Linearization coefficients diagonal matrix i-th layer for upper bound computation.
$\underline{\boldsymbol{b}}^{(i)}$	Linearization coefficients bias vector i-th layer for lower bound computation.
$\overline{\boldsymbol{b}}^{(i)}$	Linearization coefficients bias vector i-th layer for upper bound computation.
L	Lipschitz constant.
R	Coverage ratio in probabilistic approaches.
δ	Confidence error.
ψ	Confidence level.
Abstract Interpretation	
$\wp(\cdot)$	Power set.

$\wp(\mathbb{R})^d$	Set of all subsets of $\mathbb{R}^d$.
$f^\sharp : \wp(\mathbb{R})^d \to \wp(\mathbb{R})^n$	Abstract semantics of f associating with any $\mathcal{X} \in \wp(\mathbb{R})^d$ a tuple $\mathcal{R} \in \wp(\mathbb{R})^n$.
$\mathbb{C} \stackrel{\text{def}}{=} [1, n] \times \wp(\mathbb{R})$	Index and tuple of a reachable set in the output of a DNN.
$g^\sharp : \wp(\mathbb{R})^n \to \wp(\mathbb{C})$	Function that returns indices and values of the maximum intervals in a tuple of reachable sets $\mathcal{R}$.
$\dot{\subseteq}$	Pointwise extension of $\subseteq$ to tuples of sets.
$\overline{\mathfrak{I}} : \wp(\mathbb{R})^d \to \wp(\mathbb{R})^d$	Widening input perturbation function.
$uco(\wp(\mathbb{R})^d)$	Set of all upper closure operators (i.e., monotonic, idempotent, and extensive functions) over $\wp(\mathbb{R})^d$.
$C \in uco(\wp(\mathbb{C}))$	Function that abstracts the result of the abstracted DNN $f^\#$

Counterfactual Explanations

d	Input space dimension.
$\mathcal{X}$	Input space.
$\mathcal{M} : \mathcal{X} \to [0, 1]$	Neural network classifier.
$\theta \in \Theta \subseteq \mathbb{R}^k$	Parameters vector.
$\mathcal{M}_\Theta$	Parametric classifier.
$\mathcal{M}_\theta$	Instantiation of a parametric classifier.
S	Model shift/change.
$d_p(\cdot, \cdot)$	Distance metric.
δ	Models distance threshold.
$\Delta := \{S \mid d_p(\mathcal{M}_\theta, S(\mathcal{M}_\theta)) \leq \delta\}$	Set of model changes less then δ threshold.
$\mathcal{I}$	Interval neural network.

CHAPTER 1

Introduction

"It is through science that we prove, but through intuition that we discover."

– Henri Poincaré

Deep neural networks (DNNs) have profoundly transformed artificial intelligence (AI), enabling intelligent systems to perceive, decide, and act with unprecedented accuracy. From image recognition [203] to robotic manipulation [98, 171], and from medical robotics [248] to environmental sustainability [32, 181, 23, 24, 184], their adoption has extended into increasingly complex and safety-critical domains, including autonomous navigation [249, 172] and large-scale decision-making systems [169].

Although these fields are diverse, they all share a crucial need to align with the EU's vision for *trustworthy, human-centric and reliable AI-based decision-making* [261]. In this thesis, we show how, despite their remarkable empirical performance across a wide range of domains, neural networks remain fundamentally opaque and fragile as function approximators, and thus developing techniques that can provide safety guarantees for these models is of paramount importance. Specifically, two key challenges prevent their reliable deployment:

1. **Susceptibility to adversarial inputs:** small, often human-imperceptible perturbations in input data that can lead to drastically incorrect predictions, thus undermining both safety and robustness of the system.
2. **Lack of interpretability:** DNNs' internal decision-making processes are often inscrutable, earning them the label of "black boxes".

Crucially, these vulnerabilities are not just theoretical. To provide the reader with a concrete example of adversarial input in a robotic navigation context, consider a mobile robot trained via any state-of-the-art deep reinforcement learning (DRL) technique that has to navigate toward a goal using only LiDAR inputs, without any global map of the environment. Even if such a robot performs empirically well (measured in terms of how many times the agent reaches the goal without colliding), studies such as [8] demonstrate that specific input configurations can cause the agent to fall into undesired, and often unsafe, behaviors. One such scenario is illustrated in Fig. 1.1, where a seemingly minor change in perception drastically alters the outcome of the agent's policy.

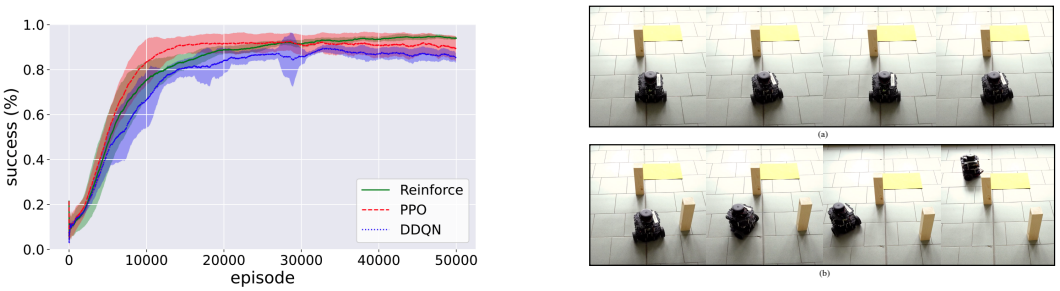


Fig. 1.1: Explanatory image of adversarial input in a DRL setup. The robot is trained and is generally able to navigate and reach the yellow target in the environment (as shown in the learning curves on the left). On the right (a), formal verification detects an unsafe input configuration, where the agent exhibits a suboptimal behavior as it is stuck in an infinite alternating loop. When an obstacle is added, changing the agent's observation, the robot is able to escape from the loop and reach the target (b). The video of this experiment is reported here.

To systematically search for such critical behaviors before deployment, the research field of formal verification (FV) of neural networks [150], has emerged as a valuable solution to provide formal assurances on the safety aspect of these functions. Specifically, formal verification aims to mathematically prove that, under all allowable inputs within a specified domain, a neural network will (or will not) behave according to a given specification. However, the development of effective verification tools for deep neural networks remains a challenging problem. Firstly, the same characteristics that make DNNs powerful—high dimensionality, non-linearity, and layered composition, also make the verification an NP-hard problem to solve [131, 283]. As a result, current techniques often struggle with scalability, rely on overly conservative approximations, or become computationally infeasible when applied to modern architectures or real-world applications. In addition, most existing verification methods focus on verifying low-level safety and robustness properties, such as those defined by bounded input ranges or l_p norms. Yet, real-world scenarios require verifying far more high-level and semantically rich properties. For example, in autonomous driving, it is not sufficient to define safety in terms of small perturbations in pixel space. Instead, safety constraints must capture high-level concepts—such as *"do not turn right if another vehicle is present on the right"*, which pose a substantial challenge for current verification tools that are primarily designed for low-level safety and robustness checks. Finally, the potential of verification techniques to enhance the explainability of AI systems,particularly those based on supervised and reinforcement learning, remains largely underexplored in the current literature.

To address these challenges, this thesis tackles three interconnected research question.

(RQ.1) *How can we enhance scalability and expressivity of formal verification techniques to larger and more realistic neural networks while maintaining safety guarantees?*

The scalability and expressiveness of formal verification methods is crucial to make them applicable to realistic deep neural networks. Overcoming the computational bottlenecks caused by the high dimensionality and non-linearity of DNNs is essential to provide provable safety guarantees for models deployed in real-world settings. This first objective thus focuses on developing efficient and scalable verification strategies that preserve soundness while extending applicability to large architectures.

(RQ.2) *How can verification be incorporated into the DRL training loop, and how can we move beyond simple, hand-designed safety properties to automatically define meaningful behavioral preferences for complex robotic tasks?*

Integrating verification techniques into the learning process of intelligent systems aims to move beyond post-hoc analysis, fostering a closer interplay between training and formal reasoning. The solutions developed in this direction help bridge the gap between low-level safety specifications and the design of more semantically meaningful, task-relevant, safety properties.

(RQ.3) *What role can verification methods play in enabling explainability, e.g., via counterfactual reasoning, in complex decision-making systems?*

Understanding the connection between verification and explainability is of paramount importance, as verification methods go beyond heuristic or empirical validation by providing provable guarantees about a model's behavior. Such guarantees are essential for enabling explanations that are not only interpretable but also formally grounded in the system's verified properties. Answering to this research question therefore bridges verification and explainable AI, contributing to the development of transparent and trustworthy intelligent systems.

Throughout this work, we present novel verification techniques that advance both the development of explainable AI (XAI) methods and the enhancement of safety in deep reinforcement learning applications. The proposed approaches integrate sound approximation, probabilistic guarantees, and task-specific semantics to highlight the impact of verification as a unifying tool, enabling more interpretable models while simultaneously improving the reliability and safety of learning-based systems. Our contributions, briefly highlighted in the next section, make verification and explainable AI not only more theoretically grounded but also practically applicable in high-stakes, real-world environments.

1.1 Main Contributions

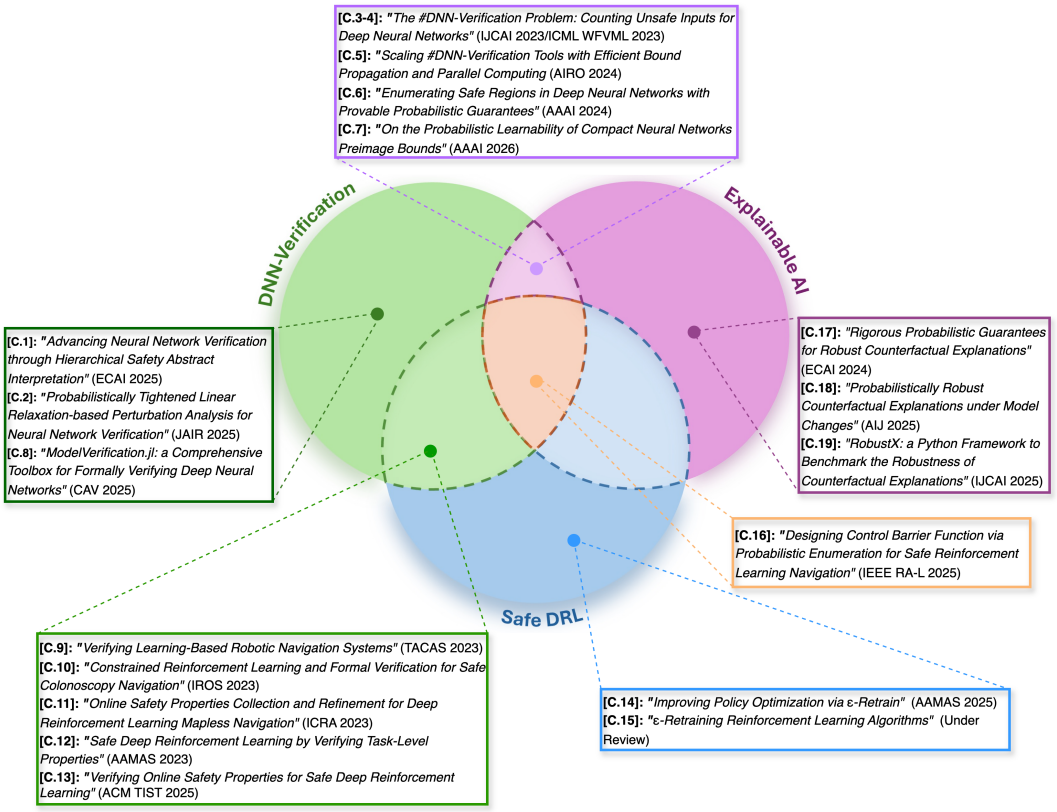


Fig. 1.2: A schematic overview of the contributions made in each of the areas covered in this thesis.

This work lies at the intersection of theoretical computer science and artificial intelligence, with a focus on the verification and explainability of deep neural networks, particularly in the context of safe deep reinforcement learning applications. This thesis contributes to both fields by tackling fundamental computational challenges related to the tractability, complexity, and approximability of verifying and interpreting modern AI systems. By developing novel abstraction techniques and scalable probabilistic methods, this research advances the theoretical foundations of verification, while also providing practical tools and demonstrations (highlighted in Fig.1.3) for improving the reliability, safety, and transparency of AI-driven decision-making.

(a) Mobile Navigation (b) Aquatic Monitoring

Fig. 1.3: Application scenarios used in this thesis.

In the following, we will divide the contributions of this work into three areas and their intersections, depicted in Fig.1.2, namely *DNN-Verification*, *Safe DRL applications*, and finally *Explainable AI*.

We briefly summarize the contributions in each area in the following:

- ***DNN-Verification***: to address the NP-hard nature of verifying deep neural networks, we introduce a novel abstract interpretation-based technique that computes sound over-approximations of DNN outputs under uncertain inputs. This enables tractable safety analysis even for large models. Notably, our approach supports reasoning over a hierarchical structure of safety and robustness properties, allowing for more expressive and flexible verification in complex scenarios where traditional safety constraints are difficult to specify or verify. To further enhance scalability in real-world systems, we develop approximation algorithms that relax overly conservative bounds while still providing provable probabilistic guarantees on safety. Drawing from randomized algorithms and probabilistic reasoning, these methods not only tackle core challenges in AI safety but also contribute new theoretical insights into the approximability of hard combinatorial problems—an important theme in theoretical computer science.
  Beyond these results, standard DNN verification typically produces a binary outcome, either the model is safe or unsafe, without quantifying how much of the input space is affected by violations. This limited expressivity motivates the introduction of a new perspective: the #DNN-Verification. Rather than merely identifying whether a counterexample exists, we aim to compute the total measure (e.g., volume or probability) of the input space that satisfies a given safety property. Building on this foundation, we further introduce the AllDNN-Verification problem, which aims to enumerate all distinct regions of the input space where a given safety property holds. However, as for other classical counting-enumerating problems such as #SAT,AllSAT [265], exactly enumerating the input regions where the safety property holds is computationally intractable, as we will show in this thesis, it is a #P-hard problem. Nonetheless, its expressive power enables a new intersection between verification and explainability. By explicitly identifying the portions of the input space that satisfy (or violate) a safety property, we can provide valuable information to guide model analysis, improve training procedures through targeted data augmentation, and inform safe recovery strategies by identifying and avoiding risky regions during deployment. In this context, large or fragmented unsafe regions are particularly challenging to analyze, especially in time-sensitive or safety-critical applications. Hence, producing compact and interpretable representations of the identified regions becomes essential to enhance model explainability and support effective fallback mechanisms.
  To address this problem, we propose novel techniques based on ensembles of randomized decision trees, which approximate the geometry of these regions while maintaining interpretability. These methods allow us to efficiently represent and analyze the structure of safe and unsafe input spaces, offering both theoretical and practical contributions to the field of verifiable and explainable AI.

- ***Safe DRL applications***: We explore how the formal and probabilistic verification methods introduced in the previous part of the thesis can be effectively applied to ensure the safety of deep reinforcement learning systems. Due to the NP-hard nature of formal verification, these techniques are typically applied post-training. Within this set of contributions, we investigate two complementary directions:
  (i) **Post-training verification for model selection**: we demonstrate how formal verification can be used to guide model selection in safety-critical applications, such as autonomous navigation in medical robotic colonoscopy. Here, verification helps identify the most reliable policy among several candidates, based on formal safety guarantees.
  (ii) **Verification-informed training**: We investigate the integration of our novel probabilistic verification methods into the DRL training loop, enabling safety-aware learning. Additionally, we study how safety-relevant information collected during training can be leveraged to automatically infer task-level safety properties, particularly in scenarios where such properties are difficult to specify manually.
  (iii) **Verification-inspired recovery behavior approach**: We integrate probabilistic enumeration methods with control theory approaches to design deployment-time recovery behaviors for reinforcement learning policies, ensuring safety in robotic navigation and aquatic monitoring tasks.
- ***Explainable AI***: the theoretical results developed in the context of neural network verification naturally lead to the third core dimension of this thesis: the investigation of novel techniques for the explainability of AI systems, with a particular focus on counterfactual explanations (CEs). Despite the increasing attention counterfactual explanations have received in recent years, the computational complexity underlying their generation remains largely overlooked. Most existing approaches are heuristic or optimization-based, often lacking formal analysis of their tractability and offering no guarantees on the robustness of the generated explanations, that is, whether small changes in the model or input would invalidate them. In contrast, this thesis starts by tackling the problem from a theoretical standpoint. We begin with a rigorous study of the computational complexity of generating robust counterfactuals. Specifically, we prove that computing robust counterfactuals for widely-used piecewise-linear models (e.g., neural networks with ReLU activations) is NP-hard. This foundational result highlights the intrinsic difficulty of generating reliable explanations and underscores the need for novel algorithmic strategies. Motivated by this insight, we build upon the probabilistic methods introduced in the context of DNN verification to design new approaches for generating counterfactual explanations with provable robustness guarantees. These methods combine and extend existing techniques, while also introducing new probabilistic reasoning tools to ensure that the generated counterfactuals are not only actionable but also stable under small perturbations or model shifts. This last part of the work contributes both to the theoretical understanding of the computational limits of explainability and to the development of practical tools for producing trustworthy explanations in real-world, safety-critical decision-making systems.

1.2 Thesis Outline and List of Included Articles

This thesis is organized into six chapters, each addressing a core aspect of the research and its contributions at the intersection of verification, deep reinforcement learning, and explainable AI.

- **Chapter 2 – Background and Related Work:** This chapter introduces the foundational concepts and terminology used throughout the thesis. We provide background on neural network verification, reinforcement learning, and explainable AI, with a focus on counterfactual explanations. For each topic, we discuss the most relevant related literature and highlight the main limitations and open problems that this work aims to address.
- **Chapter 3 – On Advanced Neural Network Verification Techniques:** This chapter presents our first core contribution, addressing research question **RQ.1**. We introduce novel techniques for the verification of deep neural networks based on abstract interpretation and probabilistic reasoning. Additionally, we propose the new formulations of #DNN-VERIFICATION and ALLDNN-VERIFICATION, which extend standard verification to more expressive and quantitative forms. This chapter also begins to explore the connection between verification and explainability, laying the foundation for addressing **RQ.3**.
- **Chapter 4 – Formal and Probabilistic Verification for Safe DRL Applications:** Focusing on the challenging **RQ.2**, this chapter explores how verification techniques can be integrated into the training loop of deep reinforcement learning systems. We propose methods for using verification to guide model selection in safety-critical domains and introduce a novel approach to automatically design task-level safety properties during training. These techniques are particularly targeted at robotic applications where traditional, hand-crafted safety specifications are insufficient or impractical.
- **Chapter 5 – Towards Explainable AI via Verification Techniques:** This chapter develops the contributions related to problem of **RQ.3**, focusing on the use of verification, especially probabilistic techniques, as a foundation for robust and actionable counterfactual explanations. We analyze the computational complexity of generating counterfactuals and propose scalable solutions with formal robustness guarantees, providing both theoretical insights and practical tools for enhancing the trustworthiness of AI systems.
- **Chapter 6 – Conclusion and Open Challenges:** The final chapter summarizes the main contributions of the thesis. We conclude by discussing the limitations of the current work and outlining promising directions for future research in the areas of scalable verification, safe learning, and explainable AI.

To present the contributions described above, this thesis adopts the format of a compilation thesis. It includes part of peer-reviewed articles, reproduced in full or only partially, that were developed during the course of the Ph.D. between 2023 and 2025. These articles are complemented by novel results and developments, currently under review, and thus not yet published. The following list provides the included contributions, in the order they appear in the thesis, along with the corresponding chapter, section, and the specific research problem addressed.[1]

Chapter 3: On Advanced Neural Network Verification Techniques

C.1 Marzari L., Mastroeni I. and Farinelli A. (2025). *"Advancing Neural Network Verification through Hierarchical Safety Abstract Interpretation"*, European Conference on Artificial Intelligence (ECAI). **(Section 3.1, DNN-Verification)** [182].

C.2 Marzari L., Cicalese F. and Farinelli A. (2025). *"Probabilistically Tightened Linear Relaxation-based Perturbation Analysis for Neural Network Verification"*, Journal of Artificial Intelligence Research (JAIR). **(Section 3.2, DNN-Verification)** [178].

C.3 Marzari L., Corsi D., Cicalese F. and Farinelli A. (2023). *"The #DNN-Verification Problem: Counting Unsafe Inputs for Deep Neural Networks"*, International Joint Conference on Artificial Intelligence (IJCAI). **(Section 3.3, DNN-Verification)** [173].

C.4 Marzari L., Corsi D., Cicalese F. and Farinelli A. (2023). *"Counting Unsafe Inputs for Deep Neural Networks"*, 2nd Workshop on Formal Verification of Machine Learning ICML. **(Section 3.3, DNN-Verification, same work as above)**.

C.5 Marzari L., Roncolato G., and Farinelli A. (2024). *"Scaling #DNN-Verification Tools with Efficient Bound Propagation and Parallel Computing"*, 10th Italian Workshop on Artificial Intelligence and Robotics (AIRO 2023) **(Section 3.4, DNN-Verification)** [175].

C.6 Marzari L., Corsi D., Marchesini E., Farinelli A. and Cicalese F. (2024). *"Enumerating Safe Regions in Deep Neural Networks with Provable Probabilistic Guarantees"*, Conference of the American Association for Artificial Intelligence (AAAI). **(Section 3.5, DNN-Verification)** [176].

C.7 Marzari L., Bicego M., Cicalese F. and Farinelli A. (2025). *"On the Probabilistic Learnability of Compact Neural Networks Preimage Bounds"*, Conference of the American Association for Artificial Intelligence (AAAI). **(Section 3.6, DNN-Verification)**.

C.8 Wei T., Marzari L.*, Hu H.*, Yun S. K.*, Niu P.*, Luo X. and Liu C. (2025). *"ModelVerification.jl: a Comprehensive Toolbox for Formally Verifying Deep Neural Networks"*, International Conference on Computer Aided Verification (CAV). **(Section 3.7, DNN-Verification)** [282].

[1] $*$ indicates equal contribution.

Chapter 4: Formal and Probabilistic Verification for Safe DRL Applications

C.9 Amir G., Corsi D., Yerushalmi R., Marzari L., Harel D., Farinelli A., and Katz G. (2023). *"Verifying Learning-Based Robotic Navigation Systems"*, International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). **(Some ideas contained in Section 4.1, Safe DRL applications)** [8].

C.10 Corsi D.*, Marzari L.*, Pore A.*, Farinelli A., Casals A., Fiorini P. and Dall'Alba D. (2023). *"Constrained Reinforcement Learning and Formal Verification for Safe Colonoscopy Navigation"*, IEEE International Conference on Intelligent Robots and Systems (IROS). **(Section 4.1, Safe DRL applications)** [52].

C.11 Marzari L., Marchesini E, and Farinelli A. (2023). *"Online Safety Properties Collection and Refinement for Deep Reinforcement Learning Mapless Navigation"*, IEEE International Conference on Robotics and Automation (ICRA) **(Section 4.2, Safe DRL applications)** [174].

C.12 Marchesini E.*, Marzari L.*, Farinelli A., and Amato C. (2023), *"Safe Deep Reinforcement Learning by Verifying Task-Level Properties"*, International Conference on Autonomous Agents and Multiagent Systems (AAMAS). **(Section 4.2, Safe DRL applications)** [168].

C.13 Marzari L., Cicalese F., Farinelli A., Amato C. and Marchesini E. (2025). *"Verifying Online Safety Properties for Safe Deep Reinforcement Learning"*, ACM Transactions on Intelligent Systems and Technology (TIST). **(Section 4.2, Safe DRL applications)** [179].

C.14 Marzari L., Donti L. P., Liu C. and Marchesini E. (2025), *"Improving Policy Optimization via ε-Retrain"*, International Conference on Autonomous Agents and Multiagent Systems (AAMAS). **(Section 4.3, Safe DRL applications)** [180].

C.15 Marzari L., Liu C., Donti L. P., and Marchesini E. (2025), *"ε-Retraining Reinforcement Learning Algorithms"*, Under review at Journal of Autonomous Agents and Multiagent Systems (JAAMAS). **(Section 4.3, Safe DRL applications)**.

C.16 Marzari L., Trotti F., Marchesini E. and Farinelli A. (2025). *"Designing Control Barrier Function via Probabilistic Enumeration for Safe Reinforcement Learning Navigation"*, IEEE Robotics and Automation Letters (RA-L). **(Section 4.4, Safe DRL applications)** [183].

Chapter 5 Towards Explainable AI via Verification Techniques

C.17 Marzari L., Leofante F., Cicalese F and Farinelli A. (2024). *"Rigorous Probabilistic Guarantees for Robust Counterfactual Explanations"*, European Conference on Artificial Intelligence (ECAI). **(Section 5.1, Explainable AI)** [177].

C.18 Marzari L., Leofante F., Cicalese F and Farinelli A. (2025). *"Probabilistically Robust Counterfactual Explanations under Model Changes"*, Artificial Intelligence Journal (AIJ). **(Section 5.1, Explainable AI)**.

C.19 Jiang J.*, Marzari L.*, Purohit A, and Leofante F. (2025). *"RobustX: a Python Framework to Benchmark the Robustness of Counterfactual Explanations"*, International Joint Conference on Artificial Intelligence (IJCAI). **(Section 5.2, Explainable AI)** [122].

Chapter 2

Background and Related Work

"If I have seen further, it is by standing on the shoulders of giants."

– Isaac Newton

In this section, we present the basic notation and background necessary to follow the results presented in this thesis. For the sake of clarity and readability, we assume the reader is already familiar with neural networks, reinforcement learning, and neural network verification; therefore, these topics are only briefly outlined. Nonetheless, we include concrete examples to illustrate the more complex concepts and analyze in detail related work from the literature relevant to the main themes of this work.

2.1 Neural Networks

A deep neural network classifier $f : \mathbb{R}^d \to \mathbb{R}$ (depicted in Fig. 2.1), where d refers to the input space dimension. For each layer $i = 1, \ldots, N$, we let d_i be the number of nodes in layer i. We use $z_j^{(i)}$ to denote the jth node in layer i (according to some fixed ordering of the nodes in the same level).

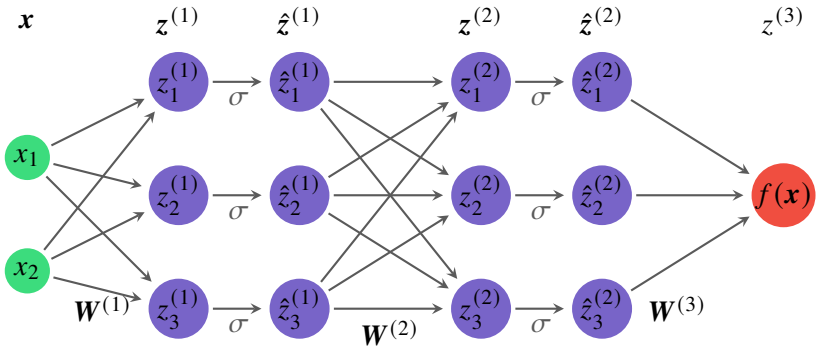


Fig. 2.1: Explanatory example of the notation used in this work for a neural network classifier f.

For a given input vector $\boldsymbol{x}$, we associate to node $z_j^{(i)}$ two values: the preactivation value, denoted by $z_j^{(i)}(\boldsymbol{x})$, and the postactivation value $\hat{z}_j^{(i)}(\boldsymbol{x})$ obtained by applying a (typically non-convex) activation function σ to the preactivation value, i.e., $\hat{z}_j^{(i)}(\boldsymbol{x}) = \sigma(z_j^{(i)}(\boldsymbol{x}))$. The preactivation value of node $z_j^{(i)}$ is obtained as a linear combination of the post-activation values of the nodes in the previous layer. In formulas, let $\boldsymbol{z}^{(i)}(\boldsymbol{x}) = (z_1^{(i)}(\boldsymbol{x}), \ldots, z_{d_i}^{(i)}(\boldsymbol{x}))$ and $\hat{\boldsymbol{z}}^{(i)}(\boldsymbol{x}) = (\hat{z}_1^{(i)}(\boldsymbol{x}), \ldots, \hat{z}_{d_i}^{(i)}(\boldsymbol{x}) = \sigma(\boldsymbol{z}^{(i)}(\boldsymbol{x})) = (\sigma(z_1^{(i)}(\boldsymbol{x})), \ldots, \sigma(z_{d_i}^{(i)}(\boldsymbol{x})))$. Then, $\boldsymbol{z}^{(i)}(\boldsymbol{x}) = \boldsymbol{W}^{(i)}\hat{\boldsymbol{z}}^{(i-1)}(\boldsymbol{x}) + \boldsymbol{b}^{(i)}$, for some given inter level weight matrix $\boldsymbol{W}^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$ and bias vector $\boldsymbol{b}^{(i)} \in \mathbb{R}^{d_i}$—as resulting from the network training. In the following we assume, without loss of generality, that there is a single node in the Nth layer, which we simply denote by $z^{(N)}$.[1] Hence we have $f(\boldsymbol{x}) = \hat{\boldsymbol{z}}^{(N)}(\boldsymbol{x}) = z^{(N)}(\boldsymbol{x})$.

The goal of learning through a DNN is to find the vector of weights $\boldsymbol{W}^{(i)}$ for each layer $i = 1, \ldots, N$ that minimizes a given loss function $\mathcal{L}$ on a given training set. A well-known loss function is the Minimum Squared Error (MSE) defined as:

$$\mathcal{L}(\boldsymbol{W}) = \mathbb{E}[(f(\boldsymbol{x}) - f^*(\boldsymbol{x}))^2]$$

where $\boldsymbol{x}$ is a given input vector for f, $f^*(\boldsymbol{x})$ is the expected output and $f(\boldsymbol{x})$ is the predicted value from the DNN. Such a loss function measures the overall performance of the DNN by quantifying how "far" the network's outputs $f(\boldsymbol{x})$ over the desired outputs $f^*(\boldsymbol{x})$. Once the loss function has been created, we search for the parameters to minimize our loss function. This optimization process is based on the so-called *gradient descent*, which is an iterative approach for altering parameter $\boldsymbol{W}$ in the opposite direction of the loss function's gradient [141]:

$$\Delta \boldsymbol{W} = -\eta \, \nabla_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{W}) = -\eta \, \frac{\partial \mathcal{L}(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

where η is the learning rate that controls the magnitude of each update, the gradient (or partial derivative) represents how the loss function $\mathcal{L}$ changes when each parameter is slightly increased. If the $\nabla \boldsymbol{W}$ changes with respect to a single parameter (e.g., a weight w) is positive, increasing the weight increases the loss function (i.e., the error), so the weight should be slightly decreased instead. If the gradient is negative, one should increase the weight. Hence, by following the negative (or positive) direction of such a gradient, the function will minimize (or maximize) such a desired objective.

[1] One can simply enforce this condition for networks that do not satisfy this assumption by adding one layer and encoding the requirements of a safety property to be verified in a single output node as a margin between logits, which is positive if only if the property is respected [150, 275].

2.2 (Deep) Reinforcement Learning

Reinforcement learning is a branch of machine learning where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards over time. Specifically, the a RL problem is typically modeled as Markov decision processes (MDPs), modeled as a tuple $(\mathcal{S}, \mathcal{A}, P, \rho, R, \gamma)$; $\mathcal{S}$ and $\mathcal{A}$ are the finite sets of states and actions, respectively, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability distribution, $\rho : \mathcal{S} \to [0, 1]$ is the initial uniform state distribution, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, and $\gamma \in [0, 1)$ is the discount factor. In policy optimization algorithms, agents learn a parameterized stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, modeling the probability to take an action $a_t \in \mathcal{A}$ in a state $s_t \in \mathcal{S}$ at a certain step t. The goal is to find the parameters that maximize the expected discounted reward $\zeta(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$, where $\tau := (s_0, a_0, s_1, a_1, \dots)$ is a trajectory with $s_0 \sim \rho(s_0)$, $a_t \sim \pi(a_t|s_t)$, $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$. We also define state and action value functions V_π and Q_π modeling the expected discount return starting from the state s_t (and action a_t for Q_π) and following the policy π thereafter as: $V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \dots}[\sum_{i=0}^{\infty} \gamma^i R(s_{t+i}, a_{t+i})]$ and $Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \dots}[\sum_{i=0}^{\infty} \gamma^i R(s_{t+i}, a_{t+i})]$. Given the current state and action, we can also measure how much better or worse the agent performs compared to its expected performance—the advantage function $A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$.

To derive a bound on the policy improvement, Schulman et al. [231] also define the expected advantage of a new policy π' over the old π, and relate the expected discounted return of π' to π: $\tilde{A}(s) = \mathbb{E}_{a\sim\pi'(\cdot|s)}[A_\pi(s, a)]$, and $\zeta(\pi') = \zeta(\pi) + \mathbb{E}_{\tau\sim\pi'}[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)]$. In practice, the dependency on trajectories induced by π' makes the above equation hard to optimize. To address this, the authors introduce a surrogate local approximation $L_\pi(\pi')$ to $\zeta(\pi')$, using the state distribution over the current policy π rather than π':

$$\begin{aligned} L_\pi(\pi') &= \zeta(\pi) + \sum_s \rho_\pi(s) \sum_a \pi'(a|s) A_\pi(s, a) \\ &= \zeta(\pi) + \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t \tilde{A}(s_t)\right]. \end{aligned} \tag{2.1}$$

With the above intuitions, they derive an upper bound on the absolute difference between the objectives:

$$\begin{aligned} |\zeta(\pi') - L_\pi(\pi')| \le \frac{4\alpha^2\gamma k}{(1-\gamma)^2} \quad \text{with } k = \max_{s,a} |A_\pi(s, a)|, \\ \alpha = D_{KL}^{\max}(\pi, \pi') = \max_s D_{KL}(\pi(\cdot|s) \;||\; \pi'(\cdot|s)). \end{aligned} \tag{2.2}$$

Finally, by employing the relationship between the total variation (TV) divergence and the Kullback–Leibler (KL) divergence $D_{TV}(p||q)^2 \le D_{KL}(p||q)$ [213], Schulman et al. [231] prove the following lower bound on the policy improvement:

$$\zeta(\pi') \ge L_\pi(\pi') - C D_{KL}^{\max}(\pi, \pi'), \text{ with } C = \frac{4k\gamma}{1-\gamma^2}. \tag{2.3}$$

2.2.1 Exploration in RL

In addition to the *vine* TRPO method discussed in the previous section, a range of works investigate the idea of changing the initial state distribution [211, 64, 191]. However, these works focus on improving exploration towards achieving higher returns rather than enforcing specific desired behaviors. For example, [191] uses states from past experiences to guide the agent toward states with higher payoffs. Similarly, [64] stores and revisits promising states to explore the environment more efficiently. In contrast, in this thesis, we will introduce a method to: (i) focus on refining agent behavior by repeatedly training on states where it failed to adhere to specific preferences, which makes it more applicable in tasks where behavior consistency and safety are required; and (ii) provide a lower bound on the policy improvement for mixed restart state distributions. In fact, our method is more closely related to the constrained MDP-related literature that is discussed in the following section.

2.2.2 Constrained Markov Decision Process

Constrained RL encourages a behavioral preference, or a safety specification, such as the ones we consider in our work [223, 243, 227]. To this end, the classical MDP extends to a constrained MDP (CMDP) considering an additional set of $C := \{C_i\}_{i \in n}$ indicator cost functions and $\mathbf{c} \in \mathbb{R}^n$ hard-coded thresholds for the constraints [6]. The goal of constrained RL algorithms is to maximize the expected reward while limiting the accumulation of costs under the thresholds. To this end, policy optimization algorithms typically employ the Lagrangian to transform the problem into an unconstrained one that is easy to implement over existing algorithms [201].

Consider the case of a single constraint characterized by a cost function $C : \mathcal{S} \times \mathcal{A} \to \{0, 1\}$—define the expected cost function $\zeta_C(\pi) := \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t C(s_t, a_t) \right]$, and a cost threshold c. The Lagrangian applies a differentiable penalty $\mathcal{L}_C(\lambda) = -\lambda \left(\zeta_C(\pi) - c \right)$ to the policy optimization objective, where λ is the so-called *Lagrangian multiplier*. These algorithms thus take an additional gradient descent step in λ: $\nabla_\lambda \mathcal{L}_C(\lambda) = c - \zeta_C(\pi)$. The multiplier is forced to be ≥ 0 as it acts as a penalty when the constraint is not satisfied (*i.e.*, λ increases) while decreasing to 0 and removing any penalty when the constraint holds. However, choosing arbitrarily small values for the threshold potentially causes a detrimental trade-off between the main task and cost objectives, ultimately leading to policies that fail to solve the problem for which they are trained. Moreover, the cost metric employed in the safety evaluation is purely empirical and does not provide any provable guarantees on the actual adherence to behavioral preferences. To address these issues, we leverage neural network formal verification.

2.2.3 Related work

Garcıa and Fernández [77] presented an exhaustive taxonomy of the main families of approaches for safe DRL, analyzing the pros and cons of each category. At a high level, safety in DRL has been pursued through formal verification techniques, control-theoretic approaches, model-based methods, constrained optimization, and penalty-based formulations. In this thesis, we focus on the model-free setup, where a variety of methods have been developed to promote safe exploration and policy learning. Nonetheless, in the following, we briefly mention the other research directions, discussing their benefits and limitations.

Formal and Probabilistic Methods-based Safe DRL. A line of research at the intersection of DRL and formal methods guarantees safety by synthesizing a runtime enforcement mechanism —commonly referred to as a *shield* —to ensure that the learned agent adheres to logic specifications [66, 135]. These methods provide strong safety guarantees but struggle to scale due to the exponential growth in the size of the synthesized automaton. As a result, they are often confined to low-dimensional domains or discrete environments. To address the scalability issue, several probabilistic approaches have arisen for safe DRL. Specifically, Jansen et al. [112] introduced *probabilistic shields* that guarantee safety with high probability while allowing efficient exploration in stochastic environments. Carr et al. [43], extended these ideas to partially observable MDPs (POMDPs).

Statistical verification (SV) techniques [296] have also been employed to validate that the agent's behavior satisfies signal temporal logic (STL) properties, which are closely related to the specifications used in formal verification. However, standard SV methods do not directly provide actionable penalties for constraint violations, unlike our proposed framework based on violation signals. Furthermore, recent approaches such as OFTEN-DeepRL [156] and LEGIBLE [250] integrate symbolic reasoning, rule mining, and norm-guided training to ensure that policies remain both effective and compliant with ethical or domain-specific constraints.

Control Barrier Function-based Safe DRL. Control-theoretic approaches such as control barrier functions (CBFs) offer a promising direction for ensuring safety, also in continuous control systems where shielding methods typically struggle. CBFs enforce forward invariance of a safe set by modifying control actions to stay within safe regions [260]. Recent works have also extended CBFs to learning-based settings by combining them with formal methods and RL [151, 107, 241]. However, these approaches require access to accurate system dynamics or high-quality estimators, which limits their applicability in high-dimensional or partially observed settings.

Model-based Safe DRL. Model-based DRL methods have been investigated in constrained and unconstrained settings [128, 303, 187, 189], but having (or approximating) a model is not always possible. Moreover, Gaussian process (GP)-based methods model the uncertainty of the environment and utilize probabilistic safety guarantees to guide exploration [4, 309]. However, GPs scale poorly with data and typically assume smooth, low-dimensional dynamics, which again limits their general use in DRL scenarios. Furthermore, these methods often require querying the

system extensively before executing policies, which can be impractical in safety-critical settings.

Constrained and Model-Free Policy Optimization. In contrast to model-based approaches, model-free constrained optimization methods have become a popular paradigm for safe DRL in single and multi-agent contexts [2, 293, 306, 14, 15]. Constrained policy optimization (CPO) [2] provides a principled framework to enforce constraints during policy updates, but it suffers from infeasibility issues due to approximation errors and relies on expensive second-order computations. Similarly, PCPO [293] uses second-order methods and has shown mixed improvements over CPO [306]. A different line of work considers Lyapunov-based approaches [48]. However, the cardinality of Lyapunov constraints equals the number of states. Hence, Lagrangian methods, described above, are commonly employed over these demanding solutions due to their simplicity and good performance. In general, constrained DRL presents significant drawbacks, as shared by all the approaches described above. For example, setting an incorrect threshold value may lead to algorithms being too permissive or, conversely, too restrictive [77]. Moreover, the guarantees of some of these approaches rely on strong assumptions that can not be satisfied in standard DRL settings, such as having access to an optimal policy. Constrained DRL is thus not devoid of short-term fatal behaviors, as it can fail at satisfying the safety constraints. Moreover, constraints naturally limit exploration, which may result in getting stuck in local optima or failing to learn desired behaviors properly [104, 162]. Despite the mentioned issues, constrained DRL methods are an important benchmark for safe DRL, and in this work, we decided to compare our approach with Lagrangian methods [243] as they reduce the complexity of prior approaches while showing higher performance and comparable constraints satisfaction.

Reward Shaping and Penalty-Based Methods. Another way to address safety in DRL is by retraining and incorporating penalties in the learning process, which has been proven effective in several works [77, 236, 1, 149]. For example, IPO [152] uses a penalty function based on constraints, giving a zero penalty when constraints are satisfied and a negative infinity upon violation. However, tuning IPO's specific parameters and the constraint threshold is not straightforward and may result in lower performance over Lagrangian methods. For this reason, RCPO [251], which proposes a straightforward Lagrangian-based penalty, often leads to better performance. Similarly, P3O [301] uses the scaled gap between an additional cost value function and desired safety thresholds as a penalty to eliminate cost constraints and optimize for safety. As we will discuss later, our objective is to incorporate quantitative verification metrics within the DRL training process. To assess the effectiveness of our verification-guided approach, we will compare it against RCPO and P3O as baseline methods.

2.3 Neural Network Verification

As previously described, DNNs are processing systems that include a collection of connected units called neurons, which are organized into one or more layers of parameterized non-linear transformations. Given an input vector $\boldsymbol{x}$, the value of each next hidden node in the network is determined by computing a linear combination of node values from the previous layer and applying a non-linear function node-wise (i.e., the activation function). Hence, by propagating the initial input values through the subsequent layers of a DNN, we obtain either a label prediction (e.g., for image classification tasks) or a value representing the index of an action (e.g., for a decision-making task).

Fig. 2.2 shows a concrete example of how a DNN computes the output. Given an input vector $\boldsymbol{x} = [1, 0]^T$, the weighted sum layer computes the value $z^{(2)} = [+5, -1]^T$. This latter is the input for the so-called activation function *Rectified Linear Unit* (ReLU), which computes the following transformation, $ReLU(x) = \max(0, x)$. Hence, the result of this activated layer is the vector $z^{(2)} = [+5, 0]^T$. Finally, the network's single output is again computed as a weighted sum, giving us the value -5.

The DNN-Verification problem can be defined in two main ways, either using the satisfiability formulation or the reachability. In general, the problem takes as input a tuple $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$, where f is a trained DNN, while $\langle \mathcal{X}, \mathcal{Y} \rangle$ encodes respectively the safety property as an input-output relationship for the DNN. More specifically, $\mathcal{X}$ is a precondition on the input that defines the possible input configuration we are interested in (typically expressed using geometric polytopes such as a Cartesian product of hyperrectangles), while $\mathcal{Y}$ encompasses the postcondition that represents the output results we aim to guarantee formally. Hence, the problem consists of verifying that for each input that lies in $\mathcal{X}$ when fed to the DNN f, the resulting output also satisfies $\mathcal{Y}$ [131, 150].

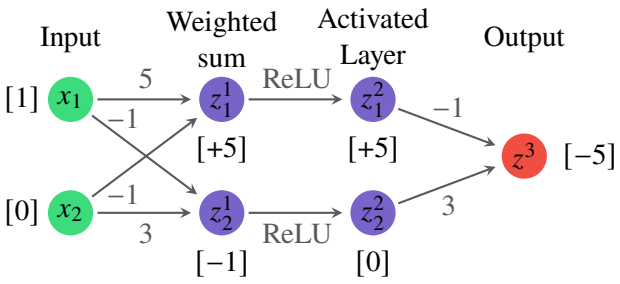


Fig. 2.2: A simple example of a DNN f that will be used as a running example throughout the section.

2.3.1 Satisfiability/Optimization formulation

In the satisfiability formulation, we can formally define the DNN-Verification as follows:

Definition 2.1 (DNN-Verification Problem):
Input: *A tuple* $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$.
Output: *`SAT` if* $\exists\, \boldsymbol{x} \mid \mathcal{X}(\boldsymbol{x}) \wedge \mathcal{Q}(f(\boldsymbol{x}))$ *and* `UNSAT` *otherwise, indicating that no such x exists.*

As an example of how this problem can be employed for checking the existence of unsafe input configurations for a DNN, suppose we aim to verify that the DNN $\mathcal{N}$ of Fig. 2.2, for any input in the interval $[0, 1]$, outputs a value greater than or equal 0. Hence, in the satisfiability formulation we define $\mathcal{X}$ as the predicate on the input vector $\boldsymbol{x} = (x_1, x_2)$ which is true iff $\boldsymbol{x} \in [0, 1] \times [0, 1]$, and $\mathcal{Y}$ as the predicate on the output $z^{(3)}$ which is true iff $z^{(3)} = f(\boldsymbol{x}) < 0$, that is we set $\mathcal{Y}$ to be the negation of our desired property. Then, solving the DNN-Verification problem on the instance $\langle f, \mathcal{X}, \mathcal{Y} \rangle$, we get `SAT` iff there is a counterexample that violates our property.

Since for the input vector $\boldsymbol{x} = [1, 0]^T$ (also reported in Fig. 2.2), the output of f is < 0, in the example, the result of the DNN-Verification problem (with the postcondition being the negated of the desired property) is `SAT`, meaning that there exists at least a single input configuration $\boldsymbol{x}$ that satisfies $\mathcal{X}$ and for which $f(\boldsymbol{x}) < 0$. As a result, we can say that the network is not safe for the desired property.

Although the satisfiability formulation of the DNN-Verification problem appears conceptually simple, solving it is computationally hard in practice. This is primarily due to the piecewise-linear and non-convex structure induced by activation functions like ReLU. Each ReLU unit introduces a branching behavior—depending on whether its input is positive or negative, it either passes the input as-is or outputs zero. For a network with n ReLU units, this results in up to 2^n different linear regions, each corresponding to a specific activation pattern. Therefore, verifying a property requires exploring an exponentially large space of such activation configurations. This leads to the problem being NP-hard even for shallow networks with few layers. In fact, it has been shown that even verifying properties for networks with a single hidden layer and piecewise-linear activations is already intractable in the worst case [131].

To address this complexity, a variety of optimization-based and symbolic reasoning methods have been developed. Tools like Reluplex [131], Marabou [287] attempt to verify a given property by searching for counterexamples that would falsify it; if no such counterexample is found, the property is deemed to hold. These methods encode the network and the specification into mathematical constraints, often leveraging Linear Programming (LP) [65] or Mixed-Integer Linear Programming (MILP) [253] formulations to model the piecewise-linear behavior of activation functions such as ReLU. In particular, Reluplex extends the simplex algorithm with ReLU-specific rules, enabling efficient reasoning about piecewise-linear constraints. SMT (Satisfiability Modulo Theories) solvers [18] also play a critical role: they allow these tools to reason over both Boolean structure and numeric theories like real arithmetic, enabling symbolic reasoning and case splitting across activation patterns [38, 37].

By combining symbolic and numerical reasoning, these approaches can effectively verify safety and robustness properties for small- to medium-sized networks.

However, these verification tools are generally designed to be *sound* and *complete* within their respective theoretical frameworks: they are guaranteed to find a counterexample if one exists (soundness), and to correctly conclude the absence of counterexamples when none are present (completeness). However, this theoretical rigor comes at a significant computational cost. Due to the NP-hard nature of the verification problem, these tools often struggle with scalability, meaning that their performance degrades rapidly as the size and depth of the network increase. Furthermore, the combinatorial explosion of possible activation patterns in networks with many ReLU units leads to intractable runtimes or memory exhaustion even for moderately sized models. As a result, these methods are often limited to relatively small networks and simplified safety properties.

2.3.2 Reachability formulation

To overcome the limitations of SMT-based solvers for the DNN-Verification problem, reachability analysis has emerged as a promising alternative. Rather than searching for a single counterexample, reachability methods aim to overapproximate the set of all possible outputs that the neural network can produce for a given input region.

Definition 2.2 (DNN-Verification Problem):
Input: *A tuple* $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$.
Output: $\texttt{Safe} \iff \mathcal{R}(\mathcal{X}, f) \subseteq \mathcal{Y}$.

Specifically, in the reachability formulation, the verification problem takes as input a tuple $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$, where, unlike in the satisfiability formulation, $\mathcal{X}$ and $\mathcal{Y}$ directly describe the geometric regions of the input and output spaces that capture the properties of interest. Hence, a reachability-based FV for neural networks tool propagates the intervals $\mathcal{X}$ through f performing a layer-by-layer reachability analysis to compute the output reachable set $\mathcal{R}(\mathcal{X}, f)$.[2] The tool then checks if $\mathcal{R}(\mathcal{X}, f) \subseteq \mathcal{Y}$, meaning that the agent satisfies the preference for all the states in $\mathcal{X}$.

Figure 2.3 shows a simplified overview of the verification process, which checks if (at least) one violation of the behavioral preference exists in $\mathcal{X}$. Due to overapproximation errors introduced by the propagations, FV tools iteratively split $\mathcal{X}$ into sub-domains $\mathcal{X}_i$ (the first two blocks in the figure) [274]. When the output reachable set $\mathcal{R}(\mathcal{X}_i, f)$ is not included in $\mathcal{Y}$ (the second to last block), the iterative procedure ends—the behavioral preference is violated if at least one portion of the domain $\mathcal{X}$ falls within this scenario.

While this approach may sacrifice completeness (e.g., due to overapproximation), it enables scalable and sound verification by operating on sets instead of individual input points, and by avoiding exhaustive case-splitting over activation patterns. Reachability-based methods thus offer a favorable trade-off between accuracy and efficiency, particularly for verifying safety properties in larger or more complex networks.

[2] In the next sections, we provide the reader with a comprehensive example of how this computation is performed.

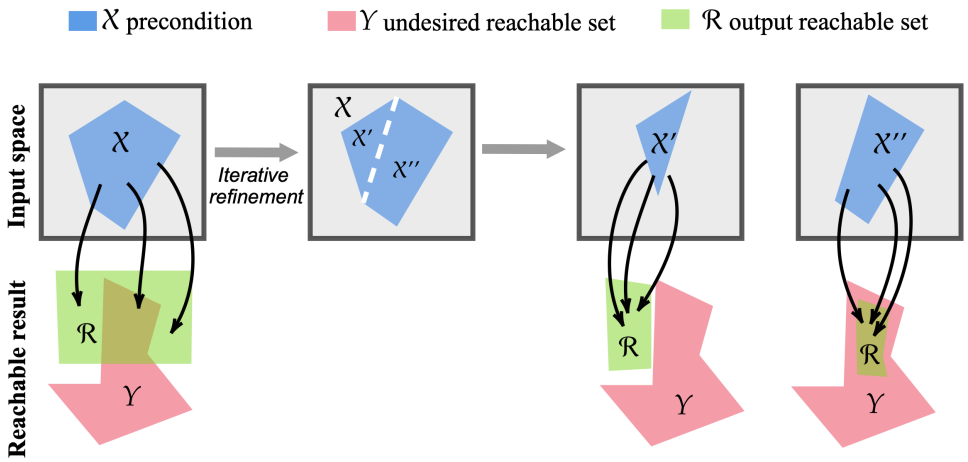


Fig. 2.3: Overview of reachability analysis for neural networks verification.

2.3.3 Robustness Verification

In contrasts with the more general satisfiability-based formulation (Def.2.1), which searches for a single counterexample violating the desired property, and with the reachability-based formulation (Def. 2.2), which overapproximates the full output set $\mathcal{R}(\mathcal{X}, f)$ and checks its containment in the (un)safe set $\mathcal{Y}$, ROBUSTNESS VERIFICATION specifically targets local adversarial robustness, making it a central task in the analysis of safety and trustworthiness for AI systems, especially in adversarial settings.

Specifically, an instance of the ROBUSTNESS VERIFICATION focuses on ensuring that small perturbations to a given input do not cause a misclassification or unsafe behavior. Intuitively, a neural network is said to be robust at a point $\boldsymbol{x}$ if, for all inputs within a bounded perturbation region $\mathcal{P}$ (e.g., an ℓ_p-ball around $\boldsymbol{x}$), the output of the network remains consistent with the original prediction.

Formally, we define the robustness verification problem of deep neural networks as follows.

Definition 2.3 (ROBUSTNESS VERIFICATION problem):
Input: *A tuple* $\mathcal{T} = \langle f, \mathcal{P} \rangle$.
Output: $\texttt{Robust} \iff \min_{\boldsymbol{x} \in \mathcal{P}} f(\boldsymbol{x}) := z^{(N)}(\boldsymbol{x}) > 0.$

For a neural network classifier with a single output[3] given an input point of interest $\boldsymbol{x}_0$, for which $f(\boldsymbol{x}_0) > 0$, and an input perturbation region $\mathcal{P} = \mathcal{P}_{\boldsymbol{x}_0,\epsilon} = \{\boldsymbol{x} \mid ||\boldsymbol{x} - \boldsymbol{x}_0||_\infty \leq \epsilon\}$, i.e., we set $p = \infty$, obtaining an N-dimensional hypercube, we aim to find, if there exists, an input $\boldsymbol{x} \in \mathcal{P}$ such that $f(\boldsymbol{x}) \leq 0$, thus resulting in a

[3] As previously stated, we assume, without loss of generality, that there is a single node in the Nth layer of the network. One can simply enforce this condition for networks that do not satisfy this assumption by adding one layer and encoding, for instance, the robustness property that we aim to verify in a single output node as a margin between logits, which produces a positive output only if the correct label is predicted [150, 275].

violation of the property.[4] If $f(\boldsymbol{x}) > 0 \ \forall \boldsymbol{x} \in \mathcal{P}$, we say f is robust (or verified) to all the possible input perturbations in $\mathcal{P}$. A possible way to prove the property is to solve the optimization problem in terms of $\min_{\boldsymbol{x}\in\mathcal{P}} f(\boldsymbol{x})$ and by checking if the result is positive.

In order to cope with the NP-hardness of the problem, (in)complete verifiers usually relax the DNNs' non-convexity to obtain over-approximate sound lower and upper bounding functions on f, respectively, denoted by $\underline{f}$ and $\overline{f}$, i.e., $\underline{f}(\boldsymbol{x}) \leq f(\boldsymbol{x}) \leq \overline{f}(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{P}$. Therefore, if $\underline{f}^* = \min_{\boldsymbol{x}\in\mathcal{P}} \underline{f}(\boldsymbol{x}) > 0$, then also $f^* = \min_{\boldsymbol{x}\in\mathcal{P}} f(\boldsymbol{x}) > 0$, i.e., the real minimum value of f will be positive, and similarly if $\overline{f}^* = \min_{\boldsymbol{x}\in\mathcal{P}} \overline{f}(\boldsymbol{x}) \leq 0$ than also $f^* = \min_{\boldsymbol{x}\in\mathcal{P}} f(\boldsymbol{x}) \leq 0$.

In both these situations, we can return a provable result. However, if $\underline{f}^* < 0 < \overline{f}^*$, we cannot be sure about the sign of f^*, and we typically have to proceed with a branch and bound (BaB) [38] process. More specifically, many FV tools firstly recursively divide the original verification problem into smaller subdomains, either by dividing the perturbation region [274] or by splitting ReLU neurons into positive/negative linear domains [37]. Secondly, they bound each subdomain with specialized (incomplete) verifiers, typically linear programming solvers [65], which can fully encode neuron split constraints. The verification process ends once we verify all the subdomains of this searching tree, or we find a single counterexample $\boldsymbol{x}$ such that $\overline{f}(\boldsymbol{x}) \leq 0$. Even though LP-verifiers are mainly used in complete FV tools, recent Linear Relaxation Perturbation Analysis (LiRPA)-based approaches [291, 275] show how to solve an optimization problem that is equivalent to the costly LP-based methods with neuron split constraints while maintaining the efficiency of bound propagation techniques, significantly outperforming LP-verification time thanks to GPU acceleration.

2.3.4 Linear Relaxation-based Perturbation Analysis

As previously highlighted, the non-convex transformation imposed by $\hat{z}^{(i)}$, i.e., by the non-linear activation functions, yields Def. 2.3 to be a non-convex NP-hard optimization problem to solve [131, 283]. To address this problem, LiRPA approaches [297, 237, 290, 291, 275] propose to cope with the non-linearity of the function computed by a neural network by computing linear approximations of each non-linear unit. The high-level idea is to compute bounds on each neuron's function in the DNN, for instance, all the ReLU nodes, that can be expressed by linear functions, for which the above *robustness verification problem* can be efficiently solved. In detail, using interval bound propagation (IBP) [154], we first compute a *reachable set* for each neuron $z_j^{(i)}$.

Definition 2.4 (Reachable set): *Given an input region $\mathcal{P} \subseteq \mathbb{R}^d$, the reachable set of a neuron $z_j^{(i)}$ is defined as the interval $[l_j^{(i)}, u_j^{(i)}]$, where $l_j^{(i)} \leq z_j^{(i)}(\mathbf{x}) \leq u_j^{(i)}$ for all $\mathbf{x} \in \mathcal{P}$.*

[4] The verification of specifications beyond the ℓ_∞ norm—such as non-convex specifications or related constraints is typically supported in state-of-the-art tools.

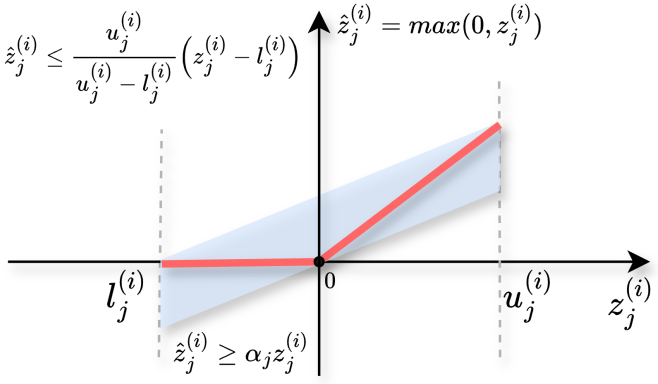


Fig. 2.4: Linear relaxation for ReLU($z_j^{(i)}$)

That is, $l_j^{(i)}$ and $u_j^{(i)}$ represent lower and upper bounds, respectively, on the pre-activation values of neuron $z_j^{(i)}$ over the entire perturbation region. The ReLU post-activation value of the node is given by $\hat{z}_j^{(i)} = \max(0, z_j^{(i)})$. Therefore, the node is considered *"unstable"* if its pre-activated bounds are such that $u_j^{(i)} > 0 > l_j^{(i)}$ and a linear approximate bound can be computed as depicted in Fig. 2.4. Otherwise, the node is either considered *"active"* if $l_j^{(i)} \geq 0$ or *"inactive"* if $u_j^{(i)} \leq 0$. Once linear bounds are established across all neurons, two propagation methods are typically employed: forward and backward. In forward propagation, the linear bounds for each neuron are expressed in terms of the input and propagated layer by layer until the output is reached. In backward propagation, we start from the output and propagate the bounds backward to earlier layers until we can express a linear relation between input and output. In detail, to improve the tightness of the bounds, [290] proposes a refined backward computation strategy that leverages information from a prior forward propagation as shown in Alg. 1.

After computing all reachable sets using a method such as interval bound propagation [154], the backward analysis constructs layer-wise linear relaxations starting from the output layer back to the input. For clarity purposes, in the following, we will only report the notation for the linear lower bound computation, but similar considerations also apply to the upper bound.

For a DNN composed of N layers, the base case is defined as $\underline{\boldsymbol{A}}^{(N)} = I$. Moreover, in the case of a single output node, $\underline{\boldsymbol{A}}^{(N-1)} = \boldsymbol{w}^\top$, where $\boldsymbol{w}$ is the weight vector of the final layer. For the remaining layers $i \in \{N-2, \ldots, 1\}$, the linear relaxation is propagated using the recurrence $\underline{\boldsymbol{A}}^{(i)} = \underline{\boldsymbol{A}}^{(i+1)} \underline{\boldsymbol{D}}^{(i)} \boldsymbol{W}^{(i)}$, where $\boldsymbol{W}^{(i)}$ is the inter level weight matrix and $\underline{\boldsymbol{D}}^{(i)}$ is a diagonal matrix encoding the linear relaxation of the activation function at the i-th layer. For each layer i, we also recursively compute a bias vector $\underline{\boldsymbol{b}}^{(i)}$ based on the reachable sets of the i-th layer nodes and the matrix $\underline{\boldsymbol{A}}^{(i+1)}$. Each diagonal entry $\underline{\boldsymbol{D}}_{j,j}^{(i)}$ depends on the preactivation bounds of neuron j (computed during the forward pass) and the sign of $\underline{\boldsymbol{A}}_j^{(i+1)}$, i.e., the coefficient

Algorithm 1: `LiRPA`[297] backward output bounds computation

Input: A DNN f with N layers, and input $\boldsymbol{x}_0$ and an ε perturbation to compute the perturbation region $\mathcal{P}$, *interm_bounds* (optional).
Output: provable lower bound of f when considering $\mathcal{P}$.

if *interm_bounds* == $\emptyset$ **then**
  interm_bounds $\leftarrow$ `IBP`$(f, \mathcal{P})$
$\underline{\boldsymbol{A}}^{(N)} = I, \underline{\boldsymbol{A}}^{(N-1)} = \boldsymbol{w}^\top$ ▷ $\boldsymbol{w}^\top$ is the weight vector of the final layer.
for $i \in \{N-2, \ldots, 1\}$ **do**
  $\underline{\boldsymbol{D}}^{(i)} \leftarrow$ `ComputeDiagonalMatrix`(interm_bounds$[i]$, $\underline{\boldsymbol{A}}^{(i+1)}$) ▷ as in Eq. 2.5
  $\underline{\boldsymbol{b}}^{(i)} \leftarrow$ `ComputeBiasVector`(interm_bounds$[i]$, $\underline{\boldsymbol{A}}^{(i+1)}$) ▷ as in Eq. 2.5
  $\underline{\boldsymbol{A}}^{(i)} \leftarrow \underline{\boldsymbol{A}}^{(i+1)}\,\underline{\boldsymbol{D}}^{(i)}\,\boldsymbol{W}^{(i)}$ ▷ $\boldsymbol{W}^{(i)}$ is the weight matrix of layer i.
$\underline{d} \leftarrow \underline{\boldsymbol{A}}^{(N-1)}\underline{\boldsymbol{b}}^{(N-2)} + \ldots + \underline{\boldsymbol{A}}^{(2)}\underline{\boldsymbol{b}}^{(1)}$
lower_bound $\leftarrow -||\underline{\boldsymbol{A}}^{(1)}||_1 \cdot \varepsilon + \underline{\boldsymbol{A}}^{(1)}\boldsymbol{x}_0 + \underline{d}$ ▷ using Hölder's inequality for $\min_{\boldsymbol{x}\in\mathcal{P}} \underline{\boldsymbol{A}}^{(1)}(\boldsymbol{x}) + \underline{d}$
return *lower_bound*

associated with neuron j in the linear relaxation of the next layer.[5] At the end of the process, we obtain a final linear lower bound function given by

$$\boldsymbol{a}^T_{\text{LiRPA}}(\boldsymbol{x}) + \boldsymbol{c}_{\text{LiRPA}},$$

where $\boldsymbol{a}^T_{\text{LiRPA}}(\boldsymbol{x}) = \underline{\boldsymbol{A}}^{(1)}\boldsymbol{x}$ and $\boldsymbol{c}_{\text{LiRPA}} = \underline{\boldsymbol{A}}^{(N)}\underline{\boldsymbol{b}}^{(N-1)} + \cdots + \underline{\boldsymbol{A}}^{(2)}\underline{\boldsymbol{b}}^{(1)}$.

A lower bound on the minimum of f in $\mathcal{P}$ (the ℓ_∞ norm ball around $\boldsymbol{x}_0$) is then easily obtained using Hölder's inequality [297] as

$$\min_{\boldsymbol{x}\in\mathcal{P}} \boldsymbol{a}^T_{\text{LiRPA}}(\boldsymbol{x}) + \boldsymbol{c}_{\text{LiRPA}} = -||\underline{\boldsymbol{A}}^{(1)}||_1 \cdot \varepsilon + \underline{\boldsymbol{A}}^{(1)}\boldsymbol{x} + \boldsymbol{c}_{\text{LiRPA}}. \tag{2.4}$$

To provide the reader a concrete and practical illustration of this approach, in the following, we present a simple example of linear bound computation for the toy DNN shown in Figure 2.5, using CROWN, a state-of-the-art LiRPA-based method [297].

Example of linear computation with LiRPA

Consider a neural network with two inputs, two hidden layers with ReLU activation, and one single output. Following the notation introduced above, we define:

$$\boldsymbol{W}^{(1)} = \begin{bmatrix} 2 & 1 \\ -3 & 4 \end{bmatrix}, \quad \boldsymbol{W}^{(2)} = \begin{bmatrix} 4 & -2 \\ 2 & 1 \end{bmatrix}, \quad \boldsymbol{w}^{(3)^T} = [-2, 1];$$

and, for simplicity, we set the bias terms in the layers to zero. We consider an original input $\boldsymbol{x}_0{}^T = [0, 1]$ and an ℓ_∞ $\varepsilon = 2$ perturbation around it, thus obtaining a perturbation region $\mathcal{P} = [[-2, 2], [-1, 3]]$.

By propagating these intervals through the DNN, we obtain the interval $[-56, 32]$ as the output reachable set. Given the reasonable size of the neural network, before

[5] The explicit formulas used for computing $\underline{\boldsymbol{D}}^{(i)}$ and $\boldsymbol{b}^{(i)}$ can be found in the discussion of the example provided below.

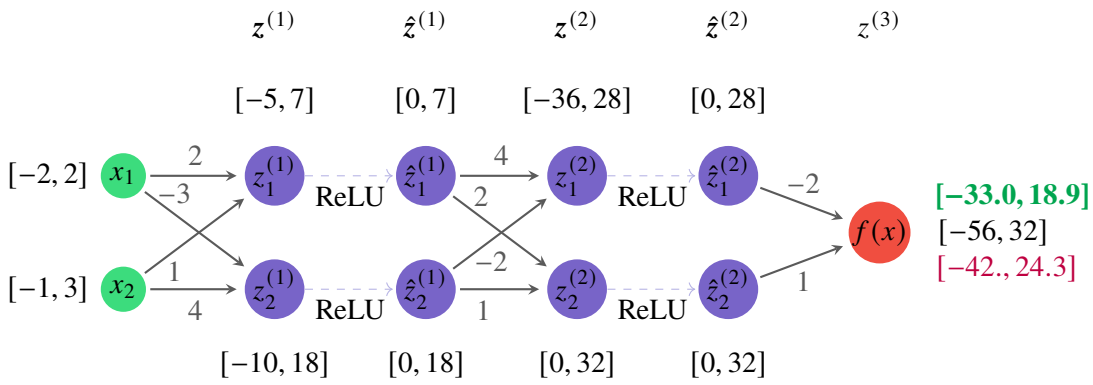


Fig. 2.5: Toy DNN used in this example. Intervals reported in green are the exact output reachable set computed via MIP, in black are the ones of IBP, and finally, in purple, the results for CROWN considering the input $[[-2, 2], [-1, 3]]$.

computing the linear lower and upper bounds using `LiRPA`, we employed mixed integer programming (MIP) [252] solution to compute the true min and max of the function, respectively, which correspond to $[-33, 18.89]$, highlighted in green in Fig. 2.5.

To compute the lower and upper bounds using CROWN [297], we employ LiRPA's backward computation strategy. To this end, it is helpful to represent the neural network as reported in Fig. 2.6.

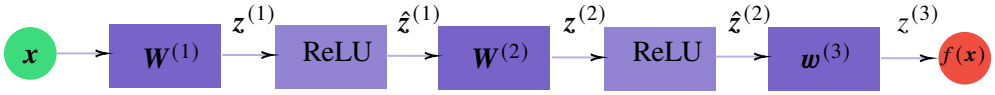


Fig. 2.6: Alternative representation of toy DNN of Figure 2.5.

We note that $\hat{z}^{(2)}$ and $\hat{z}^{(1)}$ contain non-linear activation functions (ReLU), and we have to linearize them to keep the linear relationship between the output and these hidden layers. To this end, we create $2 \times$ #ReLU layers (for the lower and upper bound, respectively) diagonal matrices $\underline{\boldsymbol{D}}^{(2)}, \overline{\boldsymbol{D}}^{(2)}, \underline{\boldsymbol{D}}^{(1)}, \overline{\boldsymbol{D}}^{(1)}$ and bias vectors $\underline{\boldsymbol{b}}^{(2)}, \overline{\boldsymbol{b}}^{(2)}, \underline{\boldsymbol{b}}^{(1)}, \overline{\boldsymbol{b}}^{(1)}$ reflecting the impact of each ReLU nodes on the final output. We report for simplicity here the original definition provided in [297] (a similar definition is applied to compute the i-th layer $\overline{\boldsymbol{D}}^{(i)}$ and $\overline{\boldsymbol{b}}^{(i)}$):

$$\underline{\boldsymbol{D}}^{(i)} = \begin{cases} 1 & \boldsymbol{l}_j \geq 0, \\ 0 & \boldsymbol{u}_j \leq 0, \\ \alpha_j & \boldsymbol{u}_j > 0 > \boldsymbol{l}_j \text{ and } \boldsymbol{A}_j^{(i+1)} \geq 0, \\ \frac{\boldsymbol{u}_j}{\boldsymbol{u}_j - \boldsymbol{l}_j} & \boldsymbol{u}_j > 0 > \boldsymbol{l}_j \text{ and } \boldsymbol{A}_j^{(i+1)} < 0 \end{cases} \quad \underline{\boldsymbol{b}}^{(i)} = \begin{cases} 0 & \boldsymbol{l}_j > 0 \text{ or } \boldsymbol{u}_j \leq 0, \\ 0 & \boldsymbol{u}_j > 0 > \boldsymbol{l}_j \text{ and } \boldsymbol{A}_j^{(i+1)} \geq 0, \\ -\frac{\boldsymbol{u}_j \boldsymbol{l}_j}{\boldsymbol{u}_j - \boldsymbol{l}_j} & \boldsymbol{u}_j > 0 > \boldsymbol{l}_j \text{ and } \boldsymbol{A}_j^{(i+1)} < 0. \end{cases} \tag{2.5}$$

In the following, for simplicity, we always set $\alpha_j = 0$. After defining the i-th diagonal matrix, we can compute the i-th layer relaxation with respect to the

output as $\underline{\boldsymbol{A}}^{(i)} = \underline{\boldsymbol{A}}^{(i+1)}\underline{\boldsymbol{D}}^{(i)}\boldsymbol{W}^{(i)}$ and similarly for the $\overline{\boldsymbol{A}}^{(i)}$. In the beginning, we set $\underline{\boldsymbol{A}}^{(4)} = \overline{\boldsymbol{A}}^{(4)} = I$ and $\underline{\boldsymbol{A}}^{(3)} = \overline{\boldsymbol{A}}^{(3)} = \boldsymbol{w}^{(3)^T}$ and write starting from right to left (backward computation)[6]

$$
\begin{aligned}
f(x) &= z^{(3)}(\boldsymbol{x}) \\
&= \boldsymbol{w}^{(3)^T}\hat{z}^{(2)}(\boldsymbol{x}) \\
&\geq \underline{\boldsymbol{A}}^{(3)}\underline{\boldsymbol{D}}^{(2)}z^{(2)}(\boldsymbol{x}) && \text{computing a linearization for } \hat{z}^{(2)} \\
&\geq \underbrace{\underline{\boldsymbol{A}}^{(3)}\underline{\boldsymbol{D}}^{(2)}\boldsymbol{W}^{(2)}}_{\underline{\boldsymbol{A}}^{(2)}}\hat{z}^{(1)}(\boldsymbol{x}) && \text{rewriting } z^{(2)} = \boldsymbol{W}^{(2)}\hat{z}^{(1)} \\
&\geq \underline{\boldsymbol{A}}^{(2)}\underline{\boldsymbol{D}}^{(1)}z^{(1)}(\boldsymbol{x}) && \text{computing a linear bound for } \hat{z}^{(1)} \\
&\geq \underbrace{\underline{\boldsymbol{A}}^{(2)}\underline{\boldsymbol{D}}^{(1)}\boldsymbol{W}^{(1)}}_{\underline{\boldsymbol{A}}^{(1)}}(\boldsymbol{x}) && \text{rewriting } z^{(1)} = \boldsymbol{W}^{(1)}\hat{z}^{(0)} = \boldsymbol{W}^{(1)}(\boldsymbol{x}) \\
&\geq \underline{\boldsymbol{A}}^{(1)}(\boldsymbol{x}) + \underline{d}.
\end{aligned}
$$

Hence, in order to linearize $\hat{z}^{(2)}(\boldsymbol{x})$ we compute $\underline{\boldsymbol{D}}^{(2)}, \overline{\boldsymbol{D}}^{(2)}$ and $\underline{\boldsymbol{b}}^{(2)}, \overline{\boldsymbol{b}}^{(2)}$ which presely correspond to

$$
\underline{\boldsymbol{D}}^{(2)} = \begin{bmatrix} \frac{u}{u-l} & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.4375 & 0 \\ 0 & 1 \end{bmatrix} \qquad \underline{\boldsymbol{b}}^{(2)} = \begin{bmatrix} \frac{-ul}{u-l} \\ 0 \end{bmatrix} = \begin{bmatrix} 15.75 \\ 0 \end{bmatrix}
$$

$$
\overline{\boldsymbol{D}}^{(2)} = \begin{bmatrix} \alpha & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \qquad \overline{\boldsymbol{b}}^{(2)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}
$$

where $\underline{\boldsymbol{D}}^{(2)}_{j,j}$ element is computed looking at each intermediate pre-activated bounds of $\boldsymbol{z}^{(2)}_j$ and the sign of j-th element of the vector $\underline{\boldsymbol{A}}^{(3)}$. Thus we have $\underline{\boldsymbol{A}}^{(2)} = \underline{\boldsymbol{A}}^{(3)}\underline{\boldsymbol{D}}^{(2)}\boldsymbol{W}^{(2)} = [-1.5, 2.75]$ and $\overline{\boldsymbol{A}}^{(2)} = \overline{\boldsymbol{A}}^{(3)}\overline{\boldsymbol{D}}^{(2)}\boldsymbol{W}^{(2)} = [2, 1]$. We proceed computing the diagonal matrix $\underline{\boldsymbol{D}}^{(1)}$, $\overline{\boldsymbol{D}}^{(1)}$ and bias vectors $\underline{\boldsymbol{b}}^{(1)}$, $\overline{\boldsymbol{b}}^{(1)}$ for $\hat{\boldsymbol{z}}^{(1)}$. In detail, we obtain,

$$
\underline{\boldsymbol{D}}^{(1)} = \begin{bmatrix} \frac{u}{u-l} & 0 \\ 0 & \alpha \end{bmatrix} = \begin{bmatrix} 0.583 & 0 \\ 0 & 0 \end{bmatrix} \qquad \underline{\boldsymbol{b}}^{(1)} = \begin{bmatrix} \frac{-ul}{u-l} \\ 0 \end{bmatrix} = \begin{bmatrix} 2.92 \\ 0 \end{bmatrix}
$$

$$
\overline{\boldsymbol{D}}^{(1)} = \begin{bmatrix} \frac{u}{u-l} & 0 \\ 0 & \frac{u}{u-l} \end{bmatrix} = \begin{bmatrix} 0.583 & 0 \\ 0 & 0.643 \end{bmatrix} \qquad \overline{\boldsymbol{b}}^{(1)} = \begin{bmatrix} \frac{-ul}{u-l} \\ \frac{-ul}{u-l} \end{bmatrix} = \begin{bmatrix} 2.92 \\ 6.43 \end{bmatrix}
$$

with $\underline{\boldsymbol{A}}^{(1)} = \underline{\boldsymbol{A}}^{(2)}\underline{\boldsymbol{D}}^{(1)}\boldsymbol{W}^{(1)} = [-1.75, -0.875]$ and $\overline{\boldsymbol{A}}^{(1)} = \overline{\boldsymbol{A}}^{(2)}\overline{\boldsymbol{D}}^{(1)}\boldsymbol{W}^{(1)} = [0.40, 3.74]$.

Finally, we compute the sum in the bias vectors $\underline{d} = \underline{\boldsymbol{A}}^{(3)}\underline{\boldsymbol{b}}^{(2)} + \underline{\boldsymbol{A}}^{(2)}\underline{\boldsymbol{b}}^{(1)} = -35.88$ and $\overline{d} = \overline{\boldsymbol{A}}^{(3)}\overline{\boldsymbol{b}}^{(2)} + \overline{\boldsymbol{A}}^{(2)}\overline{\boldsymbol{b}}^{(1)} = 12.27$.

The final linear relation is thus $\underline{f}(\boldsymbol{x}) \geq \underline{\boldsymbol{A}}^{(1)}(\boldsymbol{x}) + \underline{d}$ and $\overline{f}(\boldsymbol{x}) \leq \overline{\boldsymbol{A}}^{(1)}(\boldsymbol{x}) + \overline{d}$. Using Hölder's inequality [297], we obtain

[6] We report the lower bound version but for the upper we have similar consideration with the reversed inequality.

$$\underline{f}_{\text{CROWN}} = \min_{\boldsymbol{x}\in\mathcal{P}} \underline{\boldsymbol{A}}^{(1)}(\boldsymbol{x}) + \underline{d} = -||\underline{\boldsymbol{A}}^{(1)}||_1 \cdot \varepsilon + \underline{\boldsymbol{A}}^{(1)}\boldsymbol{x}_0 + \underline{d}$$
$$= -5.25 - 0.875 - 35.88 = -42.$$

$$\overline{f}_{\text{CROWN}} = \max_{\boldsymbol{x}\in\mathcal{P}} \overline{\boldsymbol{A}}^{(1)}(\boldsymbol{x}) + \overline{d} = ||\underline{\boldsymbol{A}}^{(1)}||_1 \cdot \varepsilon + \overline{\boldsymbol{A}}^{(1)}\boldsymbol{x}_0 + \overline{d}$$
$$= 8.28 + 3.74 + 12.27 = 24.3.$$

As we can notice, we obtain a tight over-approximation of the true lower and upper bounds, significantly improving the bounds derived with naive IBP ($[-56, 32]$).

2.3.5 Related Work

Formal Verification.

In recent years, significant research has been dedicated to formal verification and especially to robustness verification [150]. For this type of verification is important to cite sound and complete verifiers such as mixed integer programming (MIP) [252] and satisfiability module theory(SMT)-based solvers [131, 287]. Existing DNN formal verification tools like AI2[79], DeepPoly[237], leverage abstract interpretation to over-approximate the network's behavior by propagating abstract domains (e.g., intervals, zonotopes) through its layers. This enables efficient verification of properties such as robustness by bounding the output set while maintaining soundness. In addition to this class of verification approaches, Kofnov et al. [134] developed techniques to derive exact distributional bounds for networks with stochastic input, ensuring guaranteed error margins for predictive behavior. Furthermore, Athavale et al. [13] proposed an automated verification framework for confidence-based two-safety properties, allowing global robustness and fairness analysis in DNN. Other related work by Carr et al. [42] applies novel formal verification techniques to recurrent neural network (RNN) [188]-based policy learning. Specifically, the authors target policy verification under temporal logic constraints in partially observable MDP setups by extracting finite-state controllers from RNNs to enable model checking. In contrast, this thesis will focus on classification and reinforcement learning tasks in classical MDPs (i.e., fully observable), targeting safety and robustness verification of general feedforward deep neural networks under input perturbations.

The most closely related works to our proposal comprise LiRPA-based verification approaches that focus on increasing the quality of linear bounds of the most popular activation functions, such as ReLU, and more general activation functions. More specifically, [290] proposes a framework for deriving and computing near-optimal sound bounds with linear relaxation-based perturbation analysis for neural networks. This framework is the base of all the most famous state-of-the-art formal verification tools such as CROWN [297], α-CROWN [291], β-CROWN [275], GCP-CROWN [298], the top performer on last years VNN-COMP [198, 35]. Notably, this approach has also been recently employed to both provably [136] and approximate

[304] bounding neural network preimages. These approaches are typically combined with Branch-and-Bound (BaB) [38] methods, like MN-BaB[71], dividing the original verification problem into smaller subdomains either, for instance, dividing the input perturbation region [274] or splitting ReLU neurons into positive/negative linear domains [38, 37], thus combining precision with completeness at the cost of higher computational effort. All these approaches will be used as a worst-case verification result in our experiments in this thesis.

Sampling-based Approaches for Provable Verification Certificates.

Different approaches have tried to incorporate a sampling-based approach to enhance either the linear relaxation of arbitrary non-linear functions [205, 25, 292] or the verification process [16], but still maintaining provable verification certificates. For instance, [25, 205] proposed a method synthesizing linear bounds for arbitrary complex activation functions, such as GeLU [99] and Swish [221], by combining a sampling technique with an LP solver to synthesize candidate lower and upper bound coefficients and then certifying the final result via SMT solvers [76]. In [292], the authors focus on estimating the actual domain of an activation function by combining Monte Carlo simulation and gradient descent methods to compute an underestimated domain, which is then paired with over-approximations to define provable linear bounds. If, on the one hand, some of the contributions presented in this thesis also employ a similar sampling-based procedure to compute the estimated domain of reachable sets, on the other hand, they differ fundamentally in both nature and focus. Specifically, as we will see, our approach provides an explicit formula that, for any desired level of confidence, gives the number of samples sufficient to estimate the minimum and maximum pre-activation value of any node (over the set of inputs in $\mathcal{P}$) with the given confidence. In addition, we are also able to estimate the maximum error between our estimates and the true extremal pre-activation values. Crucially, our approach adopts a probabilistic perspective to derive safety insights, whereas [292] relies on combining under- and over-approximations to produce provable bounds—an approach that may inherit the limitations of formal methods discussed earlier.

Probabilistic Verification.

In this thesis, several novel approaches will be proposed to enhance probabilistic verification of deep neural networks. In this vein, recently several works have explored probabilistic verification of machine learning models. For example, [63, 197, 308] focus on a probabilistic verification perspective for general machine learning models (e.g., deep generative/diffusion models), leveraging either uncertainty sources generated by specific encoders or using symbolic reasoning and probabilistic inference. [278] proposes a Monte Carlo-based method utilizing multi-level splitting to estimate the probability of rare events for robustness verification. Other approaches rely on Chernoff-Cramér bounds, e.g., [53] estimates the local variation of neural network mappings at training points to regularize the loss function. In [160], the authors probabilistically certify a neural network by overapproximating input regions where robustness is violated. In [206] a probabilistic certification approach is proposed that can be used in general attack settings to estimate the probability of a model failing

if the attack is sampled from a certain distribution. Another line of work, like [285] uses *extreme value theory* (EVT) [92] to statistically estimate the local Lipschitz constant and assess the probabilistic robustness of the model without relying on the overapproximation of LiRPA-based solvers. Finally, recent works [20, 238, 30] focus on different types of verification, such as probabilistic specifications for robustness verification.

More closely related to our solutions are the approaches used in PROVEN [284] and the Randomized Smoothing of [49], which also focus on probabilistic robustness guarantees for neural network classifiers. PROVEN builds upon LiRPA-based techniques, combining them with concentration inequalities to derive probabilistic bounds on the network's output under input perturbations. Similarly, Randomized Smoothing constructs a smoothed classifier by adding random noise, typically Gaussian, to the input and then provides robustness guarantees by analyzing the output distribution of the smoothed model. The solution proposed in Sec 3 aligns with these approaches in that it also leverages linear relaxation techniques and probabilistic reasoning, but introduces a novel perspective by focusing on how much sampling-based underestimations affect the linear bounds used during network linearization. This enables a finer-grained analysis of robustness with strong probabilistic guarantees, which is particularly useful in scenarios where the available methods for computing exact worst-case bounds either fail (by not terminating under reasonable time and space constraints) or terminate with only over-conservative bounds.

2.4 Explainable AI through Counterfactual Explanations

To ensure clarity and consistency, we adopt a different notation in Chapter 5 than the one used for neural network verification. This choice aligns with standard conventions in the explainability literature and matches the notation used in the published article included in this thesis. For a summary and comparison of the different notations, please refer to the *List of Symbols* section at the beginning of the thesis.

Neural networks and classification tasks

Let $\mathcal{X} \subseteq \mathbb{R}^d$ denote the input space of a *classifier* $\mathcal{M}_\theta : \mathcal{X} \to [0, 1]$ mapping an input $x \in \mathcal{X}$ to an output probability between 0 and 1. We consider classifiers implemented by feed-forward DNNs parameterized by a *(parameter) vector* $\theta \in \Theta \subseteq \mathbb{R}^k$. Given two parameter vectors $\theta, \theta' \in \Theta$, we refer to the corresponding classifiers $\mathcal{M}_\theta$ and $\mathcal{M}_{\theta'}$ as *instantiations* of the same parametric classifier $\mathcal{M}_\Theta$. We assume concrete valuations of θ are learned from a set of labeled inputs as customary in supervised learning settings [86]. Once θ has been learned, the classifier can be used for inference. Without any loss of generality, we focus on binary classification tasks, i.e., the classification decision produced by $\mathcal{M}_\theta$ for an unlabeled input x is 1 if $\mathcal{M}_\theta(x) \geq 0.5$, and 0 otherwise.

Counterfactual explanations

Existing methods in the literature define CEs as follows.

Definition 2.5 (Counterfactual Explanation (CE)): *Consider an input $x \in \mathcal{X}$ and a classifier $\mathcal{M}_\theta$ s.t. $\mathcal{M}_\theta(x) < 0.5$. Given a distance metric $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$, a* (valid) counterfactual explanation *is any x' such that:*

$$x' = \underset{\hat{x} \in \mathcal{X} \,:\, \mathcal{M}_\theta(\hat{x}) \geq 0.5}{\operatorname{argmin}} \; d(x, \hat{x})$$

Intuitively, given an input x for which the classifier produces a negative outcome, a counterfactual explanation is a new input x' which is similar to x, e.g., in terms of some specified distance between features values, and for which the classifier predicts a different outcome. Common choices for d include the ℓ_1 and ℓ_∞ norms [272], which will also be used in this work.

Robustness to model changes

Among several notions of robustness, recent work has placed emphasis on generating CEs that remain valid under (slight) changes in the classifier they were generated for. While existing approaches rely on a diverse range of techniques to solve this problem, they all share a common understanding of what constitutes a model shift, which we present next.

Definition 2.6 (Jiang et al. [116]): *Let $\mathcal{M}_\theta$ and $\mathcal{M}_{\theta'}$ be two instantiations of a parametric classifier $\mathcal{M}_\Theta$. For $0 \leq p \leq \infty$, the* p-distance between $\mathcal{M}_\theta$ and $\mathcal{M}_{\theta'}$ *is defined as $d_p(\mathcal{M}_\theta, \mathcal{M}_{\theta'}) = \|\theta - \theta'\|_p$.*

Definition 2.7 (Jiang et al. [116]): *A* model change *(w.r.t. a fixed p-distance) is a function S mapping a classifier $\mathcal{M}_\theta$ into another classifier $\mathcal{M}_{\theta'} = S(\mathcal{M}_\theta)$ such that:*

- *$\mathcal{M}_\theta$ and $\mathcal{M}_{\theta'}$ are instantiations of the same $\mathcal{M}_\Theta$;*
- *$d_p(\mathcal{M}_\theta, \mathcal{M}_{\theta'}) > 0$.*

Informally, a model change captures changes in the parameters of a DNN, but does not affect its architecture. Based on this definition, we can formalize the robustness property for a CE as follows.

Definition 2.8: *Consider an input $x \in \mathcal{X}$ and a classifier $\mathcal{M}_\theta$ s.t. $\mathcal{M}_\theta(x) < 0.5$. Let x' be a counterfactual explanation computed for x s.t. $\mathcal{M}_\theta(x') \geq 0.5$. Given a set of model changes Δ, we say that the counterfactual x' is* Δ-robust *if $S(\mathcal{M}_\theta)(x') \geq 0.5$ for all $S \in \Delta$.*

The definition of a model shift can be specialized to better characterize how θ is allowed to change under S. In the following, we report two most commonly studied notions of model changes: *Naturally-Occurring Model Changes* and *Plausible Model Changes*.

Definition 2.9 (Hamman et al. [94] (NOMC)): *Consider a classifier $\mathcal{M}_\theta$. A set of model changes Δ is said to be* naturally occurring *if for a (randomly) chosen model change S from Δ and $\mathcal{M}_{\theta'} = S(\mathcal{M}_\theta)$ being the new classifier obtained after applying S to $\mathcal{M}_\theta$ the following hold:*

- $\mathbb{E}[\mathcal{M}_{\theta'}(x)] = \mathcal{M}_\theta(x)$*; where the expectation is over the randomness of* $\mathcal{M}_{\theta'}$ *given a fixed value of* x*;*
- $Var[\mathcal{M}_{\theta'}(x)] = \nu_x$*, where* ν_x *represents the maximum variance of the prediction of* $\mathcal{M}_{\theta'}(x)$*, and whenever* x *lies on the data manifold* $\mathcal{X}$*,* ν_x *is upper bounded by a small constant* ν*;*
- *If* $\mathcal{M}_\theta$ *is Lipschitz continuous for some constant* γ_1*, then* $\mathcal{M}_{\theta'}(x)$ *is also Lipschitz continuous for some constant* γ_2*.*

Broadly speaking, a naturally-occurring model shift allows the application of arbitrary changes to θ as long as the resulting model remains part of a class of models that are expected to have the same behaviour. This is in contrast with the notion of plausible model shift [262, 116], which requires changes to be bounded.

Definition 2.10 (Jiang et al. [116] (PMC)): *Consider a classifier* $\mathcal{M}_\theta$ *and a new classifier* $\mathcal{M}_{\theta'} = S(\mathcal{M}_\theta)$ *obtained after applying a model shift* S *to* $\mathcal{M}_\theta$*. Given some* $\delta \in \mathbb{R}_{>0}$ *and* $0 \leq p \leq \infty$*,* S *is said to be* plausible *(w.r.t. the choice of parameters* δ *and* p*) if* $d_p(\mathcal{M}_\theta, S(\mathcal{M}_\theta)) \leq \delta$*.*

Therefore, for any choice of parameters p, δ, and any instantiation $\mathcal{M}_\theta$ of a parametric classifier $\mathcal{M}_\Theta$ we define the set of PMC Δ_p obtained by considering all changes S that satisfy Definition 2.10, i.e. $\Delta = \{S \mid d_p(\mathcal{M}_\theta, S(\mathcal{M}_\theta)) \leq \delta\}$. Indeed, the definition of PMC assumes that both models share the same architecture and parameter dimensionality, so that the norm-based distance $||\theta - \theta'||_p$ is well-defined. Nonetheless, we discuss in the following two possible ideas to overcome this limitation. The first one applies to models whose architectures are different, but still within the same "class" (e.g., two multi layer perceptrons with hidden layers of different sizes). In this case, one could devise a padding procedure that adds *"phantom nodes"* in each smaller layers, i.e., nodes with zero weights, to match the architecture of the other network. Another interesting direction would be to define "functional" or "behavioral" distances in a shared representation space. For instance, one may define a distance $d(f_\theta, f_{\theta'}) = \mathbb{E}_{x\sim\mathcal{D}}[||f_\theta(x) - f_{\theta'}(x)||]$, where $\mathcal{D}$ is a representative data distribution. Similarly, models can be compared through distances between feature embeddings or through Lipschitz-continuous surrogates that capture functional similarity independently of the underlying architecture. Such generalizations would preserve the boundedness principle of PMC while extending its applicability to models of different classes.

In the following, we will refer to any instantiation $\mathcal{M}_{\theta'}$ of $\mathcal{M}_\Theta$ which is obtainable by applying a model change in Δ to $\mathcal{M}_\theta$ as a realization of Δ. Moreover, whenever it is not explicitly specified, we will tacitly assume that the underlying distance $d_p(\cdot, \cdot)$ is the ∞-norm.

Jiang et al. [116] proposed to reason about robustness under PMC using an Interval Neural Network (INN) [218] as an intermediate representation.

Definition 2.11: *An interval neural network* $\mathcal{I}$ *is a neural network where on each edge* e *is associated with an interval* $I_e = [a_e, b_e]$*. A realization of the INN* $\mathcal{I}$ *is a neural network having the same topology of* $\mathcal{I}$ *and such that the weight* w_e *on edge* e *satisfies* $w_e \in I_e$*,, i.e., it is taken from the interval associated to the same arc in* $\mathcal{I}$*.*

Jiang et al. [116] exploits the fact that the interval weights of an INN allow to represent an over-approximation of all the possible models obtainable under a set of PMC Δ, thus providing a compact representation of the problem. Similarly, in our work, we use the INN representation to model the PMC concept. However, instead of analyzing the robustness of counterfactual explanations through the entire INN, we focus directly on reasoning regarding the potential realizations within the set Δ. This results in several computational improvements, as we will discuss in Chapter 5.

2.4.1 Related Work

Various methods for generating CEs for DNNs have been proposed. The seminal work of [272] framed the task of generating CEs as a gradient-based optimization problem and proposed a loss that promotes CE *validity* (i.e., the CE successfully changes the classification outcome of the network) and *proximity* (i.e., the CE is as close as possible to the original input for some distance metric). In addition to these metrics, other important properties have been highlighted as crucial for the practical applicability of CEs. Prominent examples include *plausibility* (i.e., the CE must lie on the data manifold) [217, 207] and *actionability* (i.e., the changes suggested by the CE must be achievable by the user in practice) [263]. Differently from these works, here we focus on the robustness property of CEs.

Several forms of CE robustness have been studied in the literature [118]. Robustness to input changes is the focus of, e.g. [12, 239, 59, 302, 143], where solutions are devised to ensure that explanation algorithms return similar CEs for similar inputs. In another line of work, [91, 270, 142, 209] considered the problem of generating adversarially robust CEs that preserve validity under imperfect (or noisy) execution. Robustness to model multiplicity is instead considered in, e.g. [207, 145, 121], where CEs that preserve validity across sets of models are sought. However, the study of these forms of robustness is outside the scope of this thesis, as our focus is on model changes. Robustness to model changes has been studied in, e.g. [62, 27, 200, 116, 94, 119]. Of these, the approaches of [262] and [119] are the most closely related to our work. The former presents an approach to generate robust CEs under PMS using techniques from continuous optimization, which is able to guarantee robustness in the average-case scenario. The latter instead solves the same problem using abstraction techniques and discrete optimization tools, obtaining robustness guarantees that hold under worst-case conditions. Given their relevance, both approaches will be considered for an extensive experimental comparison in Chapter 5.

CHAPTER 3

On Advanced Neural Network Verification Techniques

"All truths are easy to understand once they are discovered; the point is to discover them."

– Galileo Galilei

Chapter Contributions[1]

In this chapter we address **RQ.1** and partially **RQ.3**, focusing on how to improve the scalability and expressiveness of verification methods for ensuring the safety and explainability of complex intelligent systems. Hence, we introduce a novel abstract interpretation-based method that efficiently computes sound over-approximations of DNN outputs under input uncertainty. Our approach supports reasoning over hierarchical safety and robustness properties, enabling more expressive and scalable verification in complex scenarios. To improve tractability in real-world systems, we develop approximation algorithms with formal probabilistic safety guarantees, grounded in randomized and probabilistic reasoning. Beyond standard binary verification outcomes, we present the #DNN-VERIFICATION problem, which quantifies the portion of the input space satisfying a safety property. We further define ALLDNN-VERIFICATION, which aims to enumerate all regions where a specific safety property holds. These formulations bridge verification and explainability by revealing how and where models fail or succeed. To manage the #P-hardness of the problem, we introduce efficient, interpretable approximations based on ensembles of randomized decision trees, enabling compact representations of safe and unsafe input regions and contributing to both theoretical understanding and practical tools for verifiable and explainable AI.

[1] The content of this chapter is based on the following articles:
[C.1] Marzari et. al. (2025), *"Advancing Neural Network Verification through Hierarchical Safety Abstract Interpretation"*, European Conference on Artificial Intelligence (ECAI)
[C.2] Marzari L., et al. (2025). *"Probabilistically Tightened Linear Relaxation-based Perturbation Analysis for Neural Network Verification"*, Journal of Artificial Intelligence Research (JAIR).
[C.3] Marzari L., et al. (2023). *"The #DNN-Verification Problem: Counting Unsafe Inputs for Deep Neural Networks"*, International Joint Conference on Artificial Intelligence (IJCAI).
[C.4] Marzari L., et al. (2023). *"Counting Unsafe Inputs for Deep Neural Networks"*, 2nd Workshop on Formal Verification of Machine Learning ICML.
[C.5] Marzari L., et al. (2023). *"Scaling #DNN-Verification Tools with Efficient Bound Propagation and Parallel Computing"*, 10th Italian Workshop on Artificial Intelligence and Robotics (AIRO 2024).
[C.6] Marzari L., et al. (2024). *"Enumerating Safe Regions in Deep Neural Networks with Provable Probabilistic Guarantees"*, Conference of the American Association for Artificial Intelligence (AAAI).
[C.7] Marzari L., et al. (2026). *"On the Probabilistic Learnability of Compact Neural Networks Preimage Bounds"*, Conference of the American Association for Artificial Intelligence (AAAI).
[C.8] Wei T., Marzari L.*, Hu H.*, Yun S. K.*, Niu P.*, Luo X. and Liu C. (2025). *"ModelVerification.jl: a Comprehensive Toolbox for Formally Verifying Deep Neural Networks"*, International Conference on Computer Aided Verification (CAV).

3.1 Neural Network Verification through Hierarchical Output Abstraction

Deep neural networks have significantly advanced the field of artificial intelligence, powering applications from image recognition [203] and robotics medical applications [52] to autonomous navigation [249, 172]. The success of these approximation functions across various domains has also prompted their use in safety-critical areas where expensive hardware and human safety can be involved. However, the vulnerability of these systems to the so-called adversarial inputs [246]—imperceptible modifications to the original input data that can lead to wrong and potentially catastrophic decisions— has raised concerns about their reliability and safety.

To address such an issue, formal verification of neural networks [131, 237] has recently emerged as a promising solution. This process involves defining safety or robustness specifications that can be formally verified to ensure the network behaves as intended, even in the presence of these small input perturbations [150]. In detail, a property to be verified is defined as a precondition on the input space and a postcondition on the output. In this context, the use of abstract interpretation [54] and, in particular, abstract domain provides a key resource to solve efficiently the problem [79, 237]. In fact, the precondition of a safety specification is typically encoded using an abstraction of the input domain, exploiting different geometries, such as convex polytopes, to capture all the possible configurations of an unsafe scenario we aim to verify. The postcondition represents the desired (or undesired) outcome we expect from the neural network for that specific precondition. For instance, considering a scenario where a neural network guides an autonomous system, the precondition of a safety property can encode a specific unsafe situation where the autonomous agent is near an obstacle, and the postcondition is the action that should be avoided in order not to collide. Nonetheless, in many real-world scenarios, reducing the answer of formal verification to a binary classification of *“safe"* or *“unsafe"* nature leads to the risks of losing the full expressive potential of these FV tools. In particular, when encoding a safety property, a natural question arises: *“Are some actions only marginally unsafe and thus potentially acceptable, while others pose severe risks?"* Despite significant advancements of DNN verification tools over the years [150, 131, 297, 291, 275, 282], the binary encoding of properties “safe" or “unsafe" (“robust" or “not robust") often fails to capture the full range of potential system behaviors, resulting in the verification of properties that are either overly restrictive or excessively permissive. We argue that these limitations stem from the current focus on using abstract interpretation (in particular over-approximations) to derive concrete output results rather than employing abstract reasoning to directly analyze the structure of the output set.

To address this limitation, we introduce Abstract DNN-Verification, a novel problem formalization that enables reasoning directly within a hierarchical structure of tolerable unsafe outputs. This approach allows verification of a predefined hierarchy among outputs, providing a deeper understanding of the safety levels of a DNN and potentially helping the verification in ambiguous cases—such as when the over-approximation induced by the propagation of the abstract input domain produces overlapping output results. Specifically, in these situations, incomplete

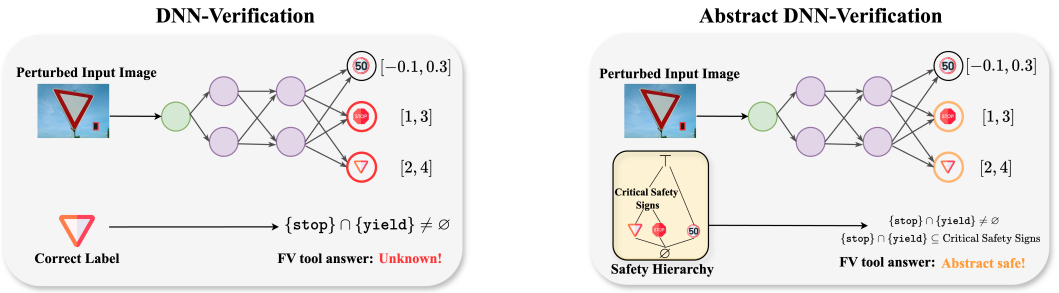


Fig. 3.1: Overview of the proposed ABSTRACT DNN-VERIFICATION in this thesis. On the left, given the intersection of the reachable sets, the concrete classes are not able to provide an answer. Conversely, if we use the first level of abstraction (right part of the image) even if the reachable sets are overlapped, i.e., the intersection is nonempty, it is fully contained in the set of *"safety-critical signs"* output abstraction, allowing to provide the *abstract safe* answer.

verifiers, which prioritize scalability over completeness, in the worst case, could return an *"unknown"* result, leaving the safety question unresolved. Conversely, sound and complete verifiers always provide an answer but require extensive input refinement [274] or splitting of the network's unstable regions [38, 275] to resolve the ambiguities, leading to significant computational overhead, especially for large-scale models. Although our formulation preserves the NP-hard complexity of the original problem [131], it introduces a novel capability: in cases with overlapping or ambiguous outputs, it allows analysts to classify the DNN as either *safe* or *abstract safe*. The latter refers to situations where the model's output does not match the exact target class but still falls within a predefined safe region of the output hierarchy specified for the verification process. Furthermore, it facilitates a deeper understanding of the regions or features that greatly impact the safety hierarchy, thus offering more actionable insights into the model's behavior.

Let us focus on the example in Fig. 3.1, and consider a scenario where a mobile robot relies on a neural network to interpret road signs from image inputs to make navigation decisions. For instance, suppose we test the system's robustness to local patch perturbations as depicted in the perturbed input image of Fig. 3.1. In this context, the standard FV process passes the perturbed image through the DNN and evaluates the verification outcome on over-approximated reachable output sets by selecting the greater reachable set.[2] However, the over-approximation error often leads to overlapping reachable sets, as shown in Fig. 3.1(left), where $\{\texttt{stop}\} \cap \{\texttt{yield}\} = [1,3] \cap [2,4] \neq \varnothing$, preventing a precise result without the necessity of further computational demanding operations. In contrast, with ABSTRACT DNN-VERIFICATION, for a given input perturbation, we can compute a set of potential DNN answers and then check if, for a given safety hierarchy, the set respects one or multiple levels of the latter, thus providing the possibility to the analyst to better understand the actual DNN's safety level. In the example, the defined hierarchy (yellow box in Fig. 3.1 (right)) groups both the $\{\texttt{stop}\}$ and $\{\texttt{yield}\}$ signs into the

[2] Typically, the node with greater value is the one selected by the DNN. We will provide further details in the next sections.

broader category of *"Critical Safety Signs"*. This hierarchy abstraction acknowledges that while a misclassification between these two signs would technically be an error, it represents a form of error that is still acceptable. Specifically, if the DNN misclassifies a "yield" as a "stop" sign, the hierarchy abstracts the decision on *"Critical Safety Signs"* which is strictly below the $\top$ (representing the fact that the DNN does not classify the sign as a "normal" sign, e.g., 50 mph), and thus the system's safety is abstractly preserved. We emphasize that the definition of the safety property, and the resulting hierarchy, is inherently shaped by the analyst's perspective and contextual judgment. This subjectivity, however, is widely acknowledged and accepted in the neural network verification literature, where it is typically grounded in domain knowledge and tailored to the specific operational context. Hence, our hierarchical approach can be designed at different levels of granularity to enrich the information available when violations occur, which is particularly valuable given the practical challenges of achieving complete model robustness in complex, real-world environments. On the example of misclassifying a "stop" sign as a "yield" sign, while this is indeed an error, our framework still offers important safety insight: the prediction remains within a class of potentially dangerous signs, rather than being misclassified as a "safe" sign. This information can be used to trigger post-verification fallback strategies, such as slowing down and stopping (even if it is a "yield" sign). In contrast, binary verification would mark this case as "not robust," providing no further guidance to the analyst. Throughout the thesis, we will show that even in safety-critical situations, our method delivers richer and more actionable feedback than binary verification, enabling system designers to understand not only that a failure occurred, but also why it occurred and how to respond appropriately.

Specifically, in the following sections, we guide the reader to understand how our new formulation allows us to answer multiple critical questions during the FV process, such as: *"Which level of unsafety does my model present?"*, *"Does the model output an action that is only potentially safe?"*, and *"How much do perturbations of a particular input feature impact on the overall safety?"*
Finally, to further confirm the impact of our new formulation, we apply Abstract DNN-Verification on realistic deep reinforcement learning human-robot cooperation tasks, namely Habitat [247, 229] as well as in two standard benchmarks of the international verification of neural networks competition (VNN-COMP) [35], such as CIFAR10 [137]. Results in the first experiment show that even relatively simple hierarchies enable in our novel problem formulation allow the ranking of different adversarial attacks based on the violation of one or more levels of the hierarchy, a capability absent in standard approaches that fail to differentiate between varying attack impacts. In the second one, for a given safety hierarchy, our approach provides valuable insights into the robustness level of deep neural networks, uncovering scenarios where instances traditionally classified as "not robust" may exhibit a specific form of robustness when viewed through the lens of tolerable (based on the safety hierarchy) misclassifications. This nuanced understanding of safety and robustness offers a more realistic and practical evaluation of the model's performance in safety-critical applications.

3.1.1 Preliminaries

This subsection recalls and extends for the sake of clarity the notation presented in Chapter 2, providing the reader with all the basic knowledge and notation to easily follow the section. We also discuss related work inherent to abstract interpretation.

Deep Neural Network.

Let $f : \mathbb{R}^d \to \mathbb{R}^n$ be a neural network that maps inputs in $\mathbb{R}^d$ to outputs in $\mathbb{R}^n$, with $m, n \in \mathbb{N}$, respectively. For a generic input $\mathbf{x} = \langle x_1, \ldots, x_d \rangle \in \mathbb{R}^d$ the corresponding output is $\mathbf{y} = f(\mathbf{x}) = \langle y_1, \ldots, y_n \rangle \in \mathbb{R}^n$. Typically, we are primarily interested in knowing which output value a DNN will select, corresponding to identifying the output node with the maximum value. Let $g : \mathbb{R}^n \to \mathbb{O}$ (with $\mathbb{O} \stackrel{\text{def}}{=} [1, n] \times \mathbb{R}$), be the function that, given the output of a DNN, returns the classification results in terms of both the index and the node with the greatest value in the output vector. Formally $g(\mathbf{y}) = \langle \text{argmax}(y_1, \ldots, y_n), \max(y_1, \ldots, y_n) \rangle$.

Abstract DNN's semantics.

Due to the neural network's inherent non-linearity and non-convexity, formal verification typically involves abstracting the input space and evaluating the verification outcome on over-approximated reachable output sets. Let $f^\sharp : \wp(\mathbb{R})^d \to \wp(\mathbb{R})^n$ be the abstract semantics of f associating with any $\mathcal{X} \in \wp(\mathbb{R})^d$ a tuple $\mathcal{R} \in \wp(\mathbb{R})^n$. In detail, $\mathcal{X}$ represents a tuple of possible input sets in $\wp(\mathbb{R})^d$, derived from input perturbations defined by a generic ℓ_p-norm. Please note the difference between $\wp(\mathbb{R})^d$ and $\wp(\mathbb{R}^d)$. The former considers a tuple of m independent intervals in $\mathbb{R}$, e.g., $\langle [0, 1], [0, 1] \rangle$. In contrast, the latter considers a set of subsets in $\mathbb{R}^d$, e.g., $[0, 1] \times [0, 1] \subseteq \mathbb{R}^2$, where each element in this subset is a tuple. In this work, we consider the ℓ_∞-norm for input perturbations in $\mathcal{X}$, but the same principles also apply to other norms. Analogously, $\mathcal{R}$ represents the tuple of output reachable sets in $\wp(\mathbb{R})^n$, i.e., the possible subset of outputs achievable when passing $\mathcal{X}$ through $f^\sharp$. Without loss of generality, in this section, we consider hyperrectangles to encode $\mathcal{X}$, i.e., the standard geometric definition as a generalization of a rectangle to multiple dimensions used in abstract interpretation and verification venues. In detail, $\mathcal{X} \in \left\{ \langle X_1, \ldots, X_d \rangle \,\middle|\, X_i = [l_i, u_i] \subseteq \mathbb{R}, l_i \leq u_i \; \forall i \in [1, d] \right\}$. Hence, the result of the propagation of $\mathcal{X}$ through $f^\sharp$ is $\mathcal{R} \in \left\{ \langle Y_1, \ldots, Y_n \rangle \,\middle|\, Y_i = [l'_i, u'_i] \subseteq \mathbb{R}, l'_i \leq u'_i \; \forall i \in [1, n] \right\}$. We define $\mathbb{C} \stackrel{\text{def}}{=} [1, n] \times \wp(\mathbb{R})$. Similar to the concrete semantics, we define a function $g^\sharp : \wp(\mathbb{R})^n \to \wp(\mathbb{C})$, which returns the set of both the indices and values of the maximum intervals in a tuple of reachable sets $\mathcal{R}$. The fact that $g^\sharp$ can return a *set* of intervals is due to the non-linearity of the DNN, causing an over-approximation of the reachable output sets and, therefore, leading to potential overlapping. Specifically, if $\mathcal{R} = \langle [l'_1, u'_1], [l'_2, u'_2], \ldots, [l'_n, u'_n] \rangle$, then $g^\sharp(\mathcal{R})$ returns the set of indices and values of the intervals for which $\max([l'_i, u'_i])$ is maximal. For instance, if $\mathcal{R} = \langle [0, 0.8], [0.12, 0.5], [1, 4] \rangle$, then $g^\sharp(\mathcal{R}) = \{\langle 3, [1, 4] \rangle\}$, but if $\mathcal{R} = \langle [2, 5], [0.12, 0.5], [1, 4] \rangle$, then $g^\sharp(\mathcal{R}) = \{\langle 1, [2, 5] \rangle, \langle 3, [1, 4] \rangle\}$.

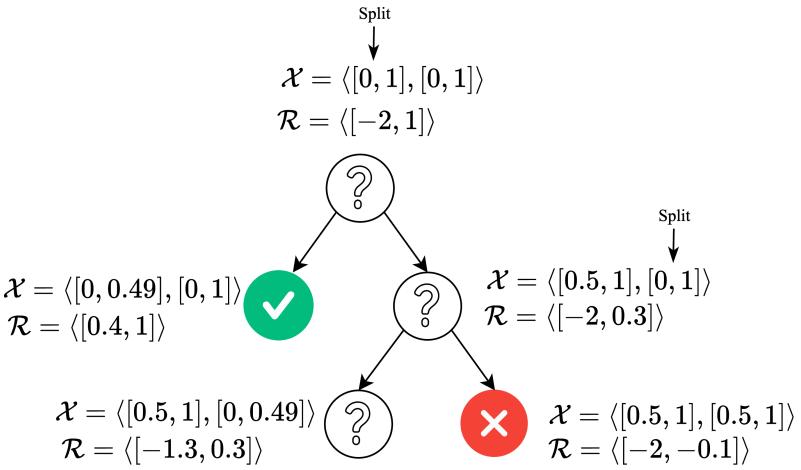


Fig. 3.2: Example of Branch-and-Bound verification process with iterative input refinement approach. In the example we consider the safe set $\mathcal{S} = [0, \infty)$ and, for the sake of simplicity, a DNN with a single output node, and thus reachable set $\mathcal{R}$.

DNN-Verification problem.

Definition 3.1 (DNN-Verification Problem):

Input: *A tuple* $\mathcal{T} = \langle f^\sharp, g^\sharp, \mathcal{X}, \mathcal{S} \rangle$.
Output: *Safe* $\iff g^\sharp \circ f^\sharp(\mathcal{X}) \subseteq \mathcal{S}$

A reachability-based FV tool for neural networks [150] using the abstract DNN semantics takes as input a tuple $\mathcal{T} = \langle f^\sharp, g^\sharp, \mathcal{X}, \mathcal{S} \rangle$, with $\mathcal{X} \in \wp(\mathbb{R})^d$ and $\mathcal{S} \subseteq \mathbb{C}$ characterizing the input-output relationship of the DNN we aim to verify. In this case, $\mathcal{S}$ represents the safe set where the output reachable set should fall for all possible $\mathbf{x} \in \mathcal{X}$.

By propagating $\mathcal{X}$ through $f^\sharp$ and applying $g^\sharp$, the tool computes the greatest output reachable set, i.e., the reachable set that has a lower bound greater than all the other upper bounds. It then checks whether it is contained in the desired reachable set, which can be represented either by a safe set $\mathcal{S}$, as in our settings, or an unsafe one. As previously mentioned, the results of $g^\sharp$ can produce overlapping reachable sets, thus preventing the possibility of directly stating whether the DNN is safe. To mitigate this issue, FV tools typically split either the elements in the tuple $\mathcal{X}$ into sub-domains [274] (each node in Fig. 3.2) or unstable neurons in the neural network [38, 275]. When at least an output reachable set of a specific subdomain is not included in $\mathcal{S}$ (the red leaf), the iterative procedure ends—the property is violated since at least one portion of the original domain $\mathcal{X}$ produces a reachable set not included in the desired $\mathcal{S}$. Otherwise, if the tool is sound and complete, it should verify all the leaves before stating that the DNN is provable safe. However, in the worst case, the NP-complete nature of DNN-Verification [131], leads to an exponential increase in the number of iterative refinements required to obtain a precise answer, and common incomplete verifiers stop at a given depth returning an *unknown* answer. In this scenario, our approach introduces a potential solution where the safety hierarchy allows in each leaf node the FV tool to provide multiple

levels of safety assessments instead of the binary *safe-unsafe* classification. This can reduce the number of iterative refinements needed for complete verifiers while it could prevent an *unknown* final outcome for incomplete verifiers.

Related Work

A similar intuition for reasoning with a hierarchical structure of tolerable unsafe outputs to provide richer analyses has been recently proposed in [82], where the concept of *Coherence* as *"weakened robustness properties"* is introduced.[3]

Definition 3.2 (*Coherence* [82].): *Let* $\mathfrak{I} : \mathbb{R}^m \to \wp(\mathbb{R}^m)$ *be an input perturbation (extensive) function on* $\mathbb{R}^m$ *(i.e.,* $\forall \mathbf{x} \in \mathbb{R}^m.\ \mathbf{x} \in \mathfrak{I}(\mathbf{x})$*), written* $\mathfrak{I} \in Pert(\mathbb{R}^m)$ *and* $g \circ f : \mathbb{R}^m \to \mathbb{O}$*. Let* $C \in uco(\wp(\mathbb{O}))$[4] *be a domain modeling the categories of classes that can be misclassified.*[5] *A DNN classifier* $g \circ f$ *satisfies Coherence w.r.t. the input* $\Omega \subseteq \mathbb{R}^m$*,* C *and* $\mathfrak{I}$ *if* $\forall \mathbf{x} \in \Omega.\ C \circ (g \circ f) \circ \mathfrak{I}(\mathbf{x}) \subsetneq \mathbb{O}$*, where* $g \circ f$ *is additively lifted to sets of inputs, those in* $\mathfrak{I}(\mathbf{x})$*.*

Nonetheless, in [82], the authors only focus on verifying whether the *concrete* execution of a DNN over a region of inputs (i.e., lifted to sets of concrete inputs in $\mathbb{R}^m$) yields an acceptable answer (i.e., one that is not *"unknown"*) up to a fixed desired output observation, modeled as an abstraction of potential concrete outputs in $\mathbb{O}$. In contrast, our approach seeks to handle any *over-approximating* execution of the DNN. Specifically, we aim to perturb the abstract input by considering a broader class of abstract inputs rather than (sub)sets of such inputs as defined in Def. 3.2. Consequently, we start by reformulating the definition using a new, more general concept of input perturbation. We then formally show why our new formulation is necessary and how the reasoning with abstract semantics addresses the limitations of the current definition.

3.1.2 The Abstract DNN-Verification Problem

This section illustrates how our novel formulation fills the gaps of [82], providing a clear motivation for the contribution of this work. Hence, we start by defining a different notion of input perturbation, widening each element of the input tuple. This input perturbation is especially useful for reasoning on the DNN's abstract semantics defined in Chapter 2.

[3] An analogous notion, i.e., adequacy, has been proposed for tuning precision in static program analysis [185, 81].

[4] A function $f : \mathtt{D} \to \mathtt{D}$ is an upper closure operator on $\mathtt{D}$ ($f \in uco(\mathtt{D})$), if it is idempotent ($\forall x \in \mathtt{D}.\ f \circ f(x) = f(x)$), extensive ($\forall x \in \mathtt{D}.\ f(x) \geq x$), and monotone.

[5] For instance in Fig. 3.1 where we have the category "Critical safety sign" modeling the fact that we accept a misclassification between the signs in this category/set

Definition 3.3 (*Widening Input Perturbation function*): *A* widening *input perturbation function* $\overline{\mathfrak{I}} \in \overline{Pert}(\wp(\mathbb{R})^d)$ *is a function* $\overline{\mathfrak{I}} : \wp(\mathbb{R})^d \to \wp(\mathbb{R})^d$ *such that* $\forall \mathcal{X} \in \wp(\mathbb{R})^d.\ \mathcal{X} \dot{\subseteq} \overline{\mathfrak{I}}(\mathcal{X})$*, where* $\dot{\subseteq}$ *is the pointwise extension of* $\subseteq$ *to tuples of sets.*

Intuitively, the function $\overline{\mathfrak{I}}$ introduces an expansion perturbation, which enlarges the range of intervals to account for more tolerant inputs.

Proposition 3.1: *Given* $\mathcal{X} \in \wp(\mathbb{R})^d$ *and* $\overline{\mathfrak{I}} \in \overline{Pert}(\wp(\mathbb{R})^d)$*, then there exists a perturbation* $\mathfrak{I} \in Pert(\mathbb{R}^d)$ *such that, for any* $\mathbf{x} \in \mathcal{X}$*, if* $\mathbf{z} \in \mathfrak{I}(\mathbf{x})$ *then* $\mathbf{z} \dot{\in} \overline{\mathfrak{I}}(\mathcal{X})$*. Formally,* $\forall \mathbf{x} \in \mathbb{R}^d,\ \mathfrak{I}(\mathbf{x}) \stackrel{\text{def}}{=} \left\{ \mathbf{z} \,\middle|\, \mathbf{z} \dot{\in} \overline{\mathfrak{I}}(\mathcal{X})) \right\}$*, where here* $\dot{\in}$ *is the point-wise* $\in$ *between tuples, i.e.,* $\langle x_1, \ldots, x_m \rangle \dot{\in} \langle X_1, \ldots, X_m \rangle$ *iff* $\forall i \in [1, m].\ x_i \in X_i$*.*

Now, let us define the novel notion of *Abstract Coherence*, extending the one defined in [82] to abstract executions of a DNN w.r.t. a given $g^\sharp$.

Definition 3.4 (*Abstract Coherence*): *Let* f *be a DNN, and* $f^\sharp$ *its abstraction. Consider* $C \in uco(\wp(\mathbb{C}))$*, and* $\overline{\mathfrak{I}} \in \overline{Pert}(\wp(\mathbb{R})^d)$*.* f *satisfies Abstract Coherence w.r.t.* $\langle f^\sharp, \overline{\mathfrak{I}}, C \rangle$ *(on* $\mathcal{X} \in \wp(\mathbb{R})^d$*) if* $C \circ (g^\sharp \circ f^\sharp) \circ \overline{\mathfrak{I}}(\mathcal{X}) \subsetneq \mathbb{C}$*.*

Broadly speaking, C is the function that abstracts the result of the abstracted DNN, potentially applied to input perturbations, determining whether the results include an *acceptable* degree of error (being strictly contained in $\mathbb{C}$), or lose too much (reaching $\mathbb{C}$). The following formal result proves our formulation's generality (and novelty) with respect to the one of [82].

Theorem 3.1: *Abstract Coherence (on* $\mathcal{X} \in \wp(\mathbb{R})^d$*) implies Coherence on any* $\mathbf{x} \in \mathcal{X}$*, but not vice-versa.*

Proof. Let $f^\sharp$ be the abstraction of a DNN f and $g^\sharp$ the decision function on the abstracted results. Consider $\overline{\mathfrak{I}} \in \overline{Pert}(\wp(\mathbb{R})^d)$ and $C \in uco(\wp(\mathbb{C}))$. Suppose to have *Abstract Coherence* of the DNN $g \circ f$ (w.r.t. $\langle f^\sharp, \overline{\mathfrak{I}}, C \rangle$). From Prop. 3.1 we have that

for any $\mathbf{x} \dot{\in} \mathcal{X}$, $\mathbf{z} \in \mathfrak{I}(\mathbf{x})$ implies $\mathbf{z} \dot{\in} \overline{\mathfrak{I}}(\mathcal{X})$. Moreover, $\forall \mathcal{Z} \in \wp(\mathbb{R})^d$ we have $g \circ f(\mathcal{Z}) \stackrel{\text{def}}{=} \left\{ g \circ f(\mathbf{z}) \,\middle|\, \mathbf{z} \dot{\in} \mathcal{Z} \right\} \subseteq g^\sharp \circ f^\sharp(\mathcal{Z})$. Suppose now $C \circ (g^\sharp \circ f^\sharp) \circ \overline{\mathfrak{I}}(\mathcal{X}) \subsetneq \mathbb{C}$, then by monotonicity of C, $\forall \mathbf{x} \dot{\in} \mathcal{X}$ we have $C \circ (g \circ f) \circ \mathfrak{I}(\mathbf{x}) \subseteq C \circ (g \circ f)(\overline{\mathfrak{I}}(\mathcal{X})) \subseteq C \circ (g^\sharp \circ f^\sharp)(\overline{\mathfrak{I}}(\mathcal{X})) \subsetneq \mathbb{C}$. Hence, *Abstract Coherence* w.r.t. $\langle f^\sharp, \overline{\mathfrak{I}}, C \rangle$ on $\mathcal{X}$ implies *Coherence* w.r.t. $\langle \mathfrak{I}, C \rangle$ on any $\mathbf{x} \dot{\in} \mathcal{X}$.
Let us provide the intuition that the opposite implication, in general, does not hold. Consider $\mathcal{X} \subseteq \mathbb{R}^d$, and suppose $\forall \mathbf{x} \in \mathcal{X}.\ C \circ (g \circ f) \circ \mathfrak{I}(\mathbf{x}) \subsetneq \mathbb{C}$ (*Coherence*). Let us define $\overline{\mathfrak{I}}(\mathcal{X}) \stackrel{\text{def}}{=} \langle X_1, \ldots, X_d \rangle \in \wp(\mathbb{R})^d$ such that $X_i = \left\{ x_i \,\middle|\, \langle x_1, \ldots, x_d \rangle \in \mathfrak{I}(\mathbf{x}),\ \mathbf{x} \dot{\in} \mathcal{X} \right\}$. Then $C \circ (g \circ f) \circ \mathfrak{I}(\mathbf{x}) \subseteq C \circ (g \circ f) \circ \overline{\mathfrak{I}}(\mathcal{X}) \subseteq C \circ (g^\sharp \circ f^\sharp) \circ \overline{\mathfrak{I}}(\mathcal{X})$. Therefore we can always find a DNN approximation $f^\sharp$, a class abstraction C and an input perturbation $\mathfrak{I}$ such that $C \circ (g \circ f) \circ \mathfrak{I}(\mathbf{x}) \subsetneq \mathbb{C}$ does not imply $C \circ (g^\sharp \circ f^\sharp) \circ \overline{\mathfrak{I}}(\mathcal{X}) \subsetneq \mathbb{C}$.

To complete the proof, we provide an example where *Coherence* does not imply *Abstract Coherence*. To this end, let us consider the following example.

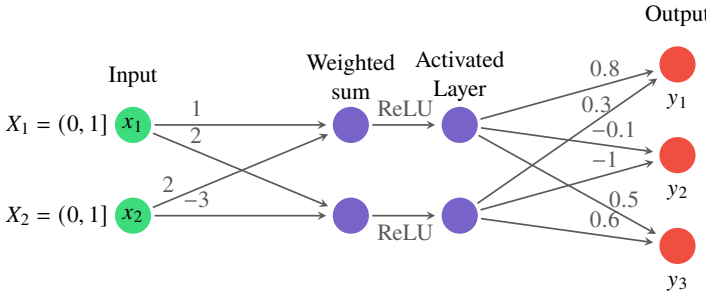


Fig. 3.3: Toy DNN for this example. $\mathbb{C} \stackrel{\text{def}}{=} \{c_i \stackrel{\text{def}}{=} \langle i, Y_i \rangle \mid i \in [1, 3]\}$.

Consider the DNN depicted in Fig. 3.3 where we have $x_1, x_2 \in [0, 1]$, namely $\mathcal{X} \stackrel{\text{def}}{=} \langle [0, 1], [0, 1] \rangle$, and $\mathcal{S} = \{c_1\} \subsetneq \mathbb{C}$. We then define $C = \{\mathbb{C}, \{c_1, c_2\}, \mathcal{S}, \varnothing\}$, meaning that we tolerate a classification that consider both c_1 and c_2 but c_3 is not accepted as considered unsafe. We start by verifying if *Coherence* is respected.

Consider an input $\mathbf{x} = \langle 0.5, 0.5 \rangle \in \mathcal{X}$. Let us define $\mathfrak{I}_\varepsilon(\mathbf{x})$ as ℓ_∞-ball perturbation of radius ϵ around $\mathbf{x}$; formally $\mathfrak{I}_\varepsilon(\mathbf{x}) = \{\mathbf{x}' \in \mathbb{R}^d \mid \|\mathbf{x}' - \mathbf{x}\|_\infty \leq \varepsilon\}$.

For this example, let us consider an ε radius for the perturbation equals 0.5 around $\mathbf{x}$ for the *Coherence* test. We start by noticing the first limitation of *Coherence* related to the necessity of executing the test on the DNN's concrete semantics. Specifically, even if we consider, for instance, a subset of the perturbation $\mathfrak{I}_{0.5}(\mathbf{x})$ that considers only the inputs of the same values, namely $\mathfrak{I}_{0.5}(\mathbf{x}) = \left\{ \langle x, x \rangle \middle| x \in (0, 1] \right\} \subseteq \mathbb{R}^2$, we have a infinity number of couples to test, which prevents the possibility to state whether *Coherence* holds or not. To address this issue, one possible solution is to assume some discretization of the input space (e.g., to the maximum resolution allowed by the machine precision), such that $\mathfrak{I}_{0.5}(\mathbf{x})$ becomes a finite set, that can be easily tested even in the concrete semantics.

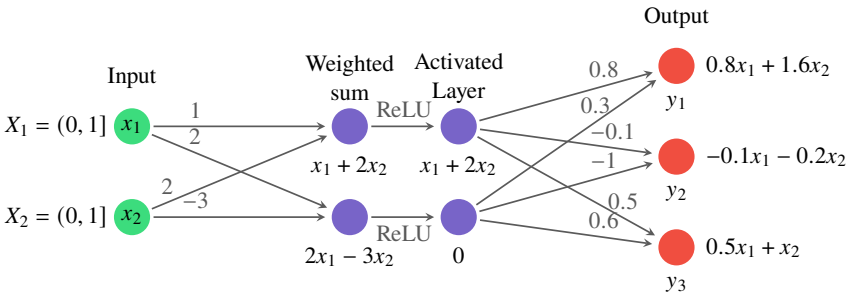


Fig. 3.4: Symbolic propagation of $\langle x_1, x_2 \rangle$ through the DNN considered in this example.

Nonetheless, in this specific example for the *Coherence* test, thanks to the simple nature of the DNN, we can employ the symbolic propagation [274] to compute linear equations of each layer w.r.t the previous one. In detail can be easily verified by direct inspection of the possible output values in the DNN of Fig. 3.4 that even if $\mathfrak{I}_{0.5}(\mathbf{x}) = \left\{ \langle x, x \rangle \middle| x \in (0, 1] \right\} \subseteq \mathbb{R}^2$ has infinite cardinality, for all $x_1 = x_2 \in (0, 1]$

the final result chosen by the DNN will always be y_1, as it presents the maximum value w.r.t the other ones.

Hence, from the fact that $C \circ (g \circ f) \circ \Im_{0.5}(\mathbf{x}) = \{c_1\} = \mathcal{S} \subsetneq \mathbb{C}$, we conclude *Coherence* is satisfied.

If we now consider the widening input perturbation function $\overline{\Im}_{0.5}(\mathbf{x}) \stackrel{\text{def}}{=} \mathcal{X} = \langle [0,1], [0,1] \rangle$ (considering, for a fair comparison, the same range of values as $\Im_{0.5}$, but now all the possible permutations in $[0,1] \times [0,1]$ and not only same couple of values $x_1 = x_2 \in (0,1]$), we observe that *Abstract Coherence* is not guaranteed. Specifically, our novel formulation of perturbation function $\overline{\Im}$ and abstract DNN semantics allows us to propagate the entire abstract input domain $\mathcal{X}$ through $f^\sharp$ (using, for instance, IBP as in the DNN of Fig. 3.5), obtaining the reachable set $\mathcal{R} = \langle [0, 3.0], [-2.3, 0], [0, 2.7] \rangle$. Applying $g^\sharp$ to $\mathcal{R}$ yields the set $\{c_1, c_3\}$. Consequently, the result of $C(g^\sharp(\mathcal{R})) = \mathbb{C}$, violates the *Abstract Coherence* requirement.

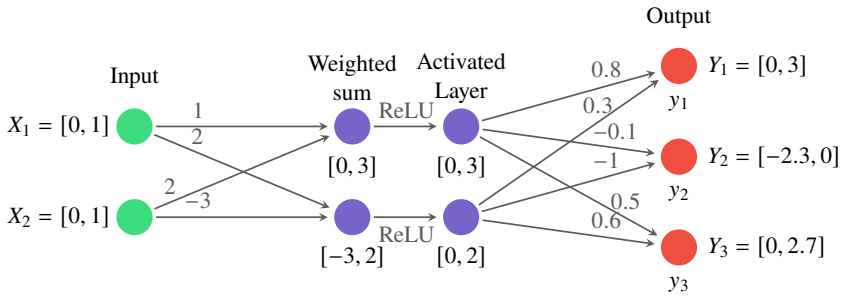


Fig. 3.5: Interval bound propagation of $\mathcal{X} = \langle [0,1], [0,1] \rangle$ through the DNN considered in this example.

To confirm this result, we employed a formal verification tool [176] to analyze whether, for any $\mathbf{x} \in \mathcal{X}$, the DNN avoids selecting the unsafe output y_3, which conflicts with the abstraction C. The tool identified a counterexample: $\mathbf{x} = \langle 0.2808, 0.0508 \rangle$, which produces the output $\mathbf{y} = \langle 0.4287, -0.4475, 0.4368 \rangle$. Evaluating $g(\mathbf{y})$ results in $\{c_3\}$ (unsafe), thereby confirming that *Abstract Coherence*, and safety in general, are not guaranteed for any $\mathbf{x} \in \mathcal{X}$.

This simple example highlights a key limitation of *Coherence*: the $\Im$ function employed in the formulation considers only a perturbation around a concrete fixed point $\mathbf{x}$. This restricted focus can mask potential unsafe behaviors that might emerge when the entire abstract domain $\mathcal{X}$ is propagated through the network. In contrast, our formulation starts from the entire abstract domain $\mathcal{X}$ and allows considering a more general abstract input by leveraging the widening input perturbation function $\overline{\Im}$ and abstract reasoning about the DNN's semantics. This demonstrates how *Abstract Coherence* offers guarantees over the entire abstract input domain rather than specific regions, thereby addressing the limitations of *Coherence*. □

At this point, we can define the ABSTRACT DNN-VERIFICATION (ADV) problem as an instantiation of Def. 3.4, verifying whether the result of the DNN-VERIFICATION applied to the perturbation of all set of abstracted inputs $\overline{\Im}(\mathcal{X})$ still respect the safe (or at least not unsafe) set determined by the abstraction C.

Definition 3.5 (ABSTRACT DNN-VERIFICATION):

Input: *A tuple* $\mathcal{T} = \langle f^\sharp, g^\sharp, \overline{\Im}(\mathcal{X}), C \rangle$

Output: *Abstract Safe* $\Leftrightarrow C \circ (g^\sharp \circ f^\sharp) \circ \overline{\Im}(\mathcal{X}) \subsetneq \mathbb{C}$

Where we recall that $\mathbb{C}$ here represents an undesired, unsafe potential result in the class prediction. With this formalization, the DNN-VERIFICATION problem becomes an instantiation of the abstract one, with $\overline{\Im} = id$ the identity function, and $C(g^\sharp \circ f^\sharp(\mathcal{X})) = \mathcal{S} \subsetneq \mathbb{C}$[6] if $g^\sharp \circ f^\sharp(\mathcal{X}) \subseteq \mathcal{S}$, $C(g^\sharp \circ f^\sharp(\mathcal{X})) = \mathbb{C}$, unsafe, otherwise. Hence, we have the following result.

Proposition 3.2: *If a DNN is safe w.r.t. the safe set* $\mathcal{S} \subsetneq \mathbb{C}$*, then it is also abstract safe w.r.t. any abstraction* $C \in uco(\wp(\mathbb{C}))$ *such that* $\mathcal{S} \in C$.

Hence, the design of C defines the tolerable level of misclassification. To clarify this, we provide an illustrative example.

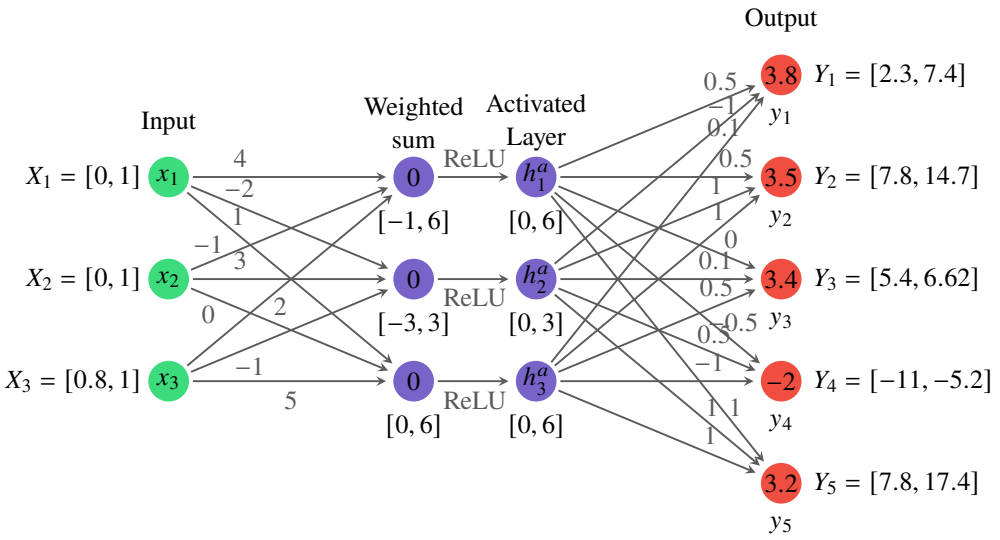


Fig. 3.6: Toy DNN employed in the ABSTRACT DNN-VERIFICATION example. $\mathbb{C} \stackrel{\text{def}}{=} \{c_i \stackrel{\text{def}}{=} \langle i, Y_i \rangle \mid i \in [1, 5]\}$.

Example 1: Consider the DNN in Fig. 3.6, where the abstracted input is $\mathcal{X} \stackrel{\text{def}}{=} \langle [0, 1], [0, 1], [0.8, 1] \rangle$ and suppose $\mathcal{S} = \{c_2, c_3, c_5\} \subsetneq \mathbb{C}$. This means that both c_1 and c_4 are unsafe, but we can suppose that there is a different degree of unsafeness between them, namely, c_4 is completely unsafe, while c_1 is only potentially unsafe (for instance, because it is quite unlikely to achieve). Starting from the concrete safe output set, we can define an output abstraction setting a tolerable misclassification, i.e., $C = \{\mathbb{C}, \{c_1, c_2, c_3, c_5\}, \mathcal{S}\}$ (i.e., from $\mathcal{S}$ we allow a more tolerable *unprecise* classification in $\{c_1, c_2, c_3, c_5\}$). We then consider the DNN as *abstract safe* if and only if the output abstraction of the network for $\mathcal{X}$ satisfies *Abstract Coherence* w.r.t.

[6] If $\mathcal{S} = \mathbb{C}$ the problem is meaningless, becoming always safe.

$\langle f^\sharp, g^\sharp, id, C\rangle$, namely $C \circ (g^\sharp \circ f^\sharp)(\mathcal{X})$ is different from $\mathbb{C}$ (here we suppose not to perturb the input, i.e., $\overline{\mathfrak{I}} = id$).

Although lifting the output of $g^\sharp$ can yield more informative answers with potentially fewer verification queries, the Abstract DNN-Verification problem, in the worst case, remains difficult to solve. Even with a large abstraction of tolerable answers, the number of iterative refinements required to reach a solution could still grow exponentially with the size of the instance. As a result, we have the following.

Proposition 3.3: *The Abstract DNN-Verification problem is NP-Complete.*

Proof. The inclusion in NP is straightforward to prove. A certificate for this problem is a specific set of abstracted reachable outputs resulting from the propagation of the perturbed inputs $\overline{\mathfrak{I}}(\mathcal{X})$ through $f^\sharp$ and $g^\sharp$. This set of outputs can be checked to see if $C(g^\#(\mathcal{R})) \subsetneq \mathbb{C}$. Given the approximated neural network function $f^\sharp$, we can compute the abstract reachable set of outputs $f^\sharp(\overline{\mathfrak{I}}(\mathcal{X}))$ using interval analysis or zonotope propagation, which are polynomial in the size of the network. Once the abstract output set from $f^\sharp(\overline{\mathfrak{I}}(\mathcal{X}))$ is obtained, we apply the following procedure to compute $g^\sharp$ (reported in Alg. 2), which is polynomial in the number of reachable sets.

Algorithm 2: Computing $g^\sharp$

Input: $\mathcal{R}$.
Output: $\wp(\mathbb{C})$.

$R = \wp(\mathbb{C})$
for $i \in [1, n]$ **do**
  for $j \in [1, n], j \neq i$ **do**
    if $\max(Y_i) \leq \min(Y_j)$ **then**
      $R = R \setminus \langle i, Y_i\rangle$
      exitfor
return R

Finally, we check if the abstract reachable set of outputs $g^\sharp \circ f^\sharp(\overline{\mathfrak{I}}(\mathcal{X}))$ is contained in the abstract safe set C. This inclusion check is usually a simple geometric containment problem (e.g., verifying if one interval or polytope is contained in another), which can indeed be done in polynomial time. Hence, the problem is in NP.

The hardness of the problem directly follows from the result of [131], showing that there exists a reduction from 3-SAT to the DNN-Verification problem.

□

Nevertheless, abstracting the safety test and allowing the formal verification tool to provide multiple responses, rather than simply classifying the system as *safe* or *unsafe*, can potentially reduce the cost of the iterative refinement process, improving scalability in practice. We will demonstrate this aspect in practice through our empirical evaluation in Sec. 3.1.3.

Applying ADV: a Simple Example

Let us consider again the DNN in Fig. 3.6, the abstracted input $\mathcal{X}$ and the safety set $\mathcal{S} = \{c_2, c_3, c_5\} \subsetneq \mathbb{C}$ as in Example 1. The propagation of $\mathcal{X}$ through $f^\sharp$ produces the abstracted DNN output $\mathcal{R} = \langle [2.3, 7.4], [7.8, 14.7], [5.4, 6.62], [-11, -5.2], [7.8, 17.4] \rangle$. In this situation, $g^\sharp(\mathcal{R}) = \{\langle 2, Y_2 \rangle, \langle 5, Y_5 \rangle\} = \{c_2, c_5\} \subseteq \mathcal{S}$, where both c_2 and c_5 are provable safe. This corresponds to *Abstract Coherence* with $C_s \stackrel{\text{def}}{=} \{\mathbb{C}, \mathcal{S}\}$. This result tells us that we do not know which of the outputs will be chosen between c_2 and c_5, but we know that it will be one of these two, and both are safe. And, formally, we have $C_s(g^\sharp(\mathcal{R})) = C_s(\{c_2, c_5\}) = \{c_2, c_3, c_5\} \subsetneq \mathbb{C}$, meaning that the DNN is provable safe, and thus for Proposition 3.2 also *abstract* safe.
By applying the input perturbation $\overline{\mathfrak{I}}$, such as $\varepsilon = 0.8$, on one input feature at a time, we can analyze how variations in individual inputs influence the output's abstract results, allowing us to gain additional safety insights into the DNN's behavior. In detail, let us consider $\overline{\mathfrak{I}}(\mathcal{X}) \stackrel{\text{def}}{=} \langle [0, 1], [0, 1], [0, 1] \rangle$, i.e., we perturb the last input feature's lower bound by ε. Hence, we have that $f^\sharp(\overline{\mathfrak{I}}(\mathcal{X}))$ produces $\mathcal{R} = \langle [0.8, 7.4], [3.5, 15.5], [3.4, 6.7], [-11, -0.5], [3.2, 18.2] \rangle$. By applying $g^\sharp$, which collects all the overlapping outputs, we obtain as set of potentially maximal values $\{c_1, c_2, c_3, c_5\}$, telling us that now the output c_1 (only potentially safe) is such that Y_1 overlaps all the intervals of safe outputs $\{c_2, c_3, c_5\}$, i.e., $g^\sharp(\mathcal{R}) = \{c_1, c_2, c_3, c_5\} \not\subseteq \mathcal{S}$ (and indeed $C_s(g^\sharp(\mathcal{R})) = \mathbb{C}$). This answer tells us that, by perturbing the X_3 feature, in any case, it is mathematically proven that the c_4 output cannot be selected, but maybe c_1 can, making the DNN no more provably safe. Nevertheless, if we consider instead abstract safety w.r.t. $C = \{\mathbb{C}, \{c_1, c_2, c_3, c_5\}, \mathcal{S}\}$ as in Example 1, then we can prove abstract safety without the necessity of further investigation, since $C(g^\sharp(\mathcal{R})) = \{c_1, c_2, c_3, c_5\} \subsetneq \mathbb{C}$.

3.1.3 Empirical Evaluation

In this section, we guide the reader in understanding the importance and impact of this novel encoding for the verification problem of deep neural networks. All data are collected on a cluster running Rocky Linux 9.34 equipped with Nvidia RTX A6000 (48 GiB) and a CPU AMD Epyc 7313 (16 cores).

We first consider the safety verification of a realistic deep reinforcement learning navigation task, namely *Habitat-Lab* [229, 247]. In this experiment, we employ the tool of [176], which allows us to select more realistic adversarial attacks (e.g., light attacks and sensor ruptures) rather than standard ℓ_∞-ball perturbation supported in state-of-the-art FV tools like α, β-*CROWN* [297, 291, 275]. In detail, we show how our novel formulation, with even relatively simple hierarchies, enables the ranking of adversarial inputs based on their impact, providing valuable insights into model robustness that traditional binary verification cannot capture. To further assess the impact of our proposal, we consider the robustness verification of well-known classification tasks on state-of-the-art benchmarks such as CIFAR10 [137], employed in VNN-COMP [35]. In this context, we employ α, β-*CROWN* [297, 291, 275], and we study the impact of different perturbation levels on standard discrete classes and the different output abstractions defined for each domain tested. Notably, the abstract safety specifications proposed in this work can be easily encoded using the standard

VNN-LIB format and thus can be verified by any other existing FV tools. Our empirical evaluation demonstrates that our novel formulation is already applicable in real-world scenarios, as evidenced by its successful use not only on standard VNN-COMP models but also on more complex architectures, such as a long short-term memory (LSTM) network with a ResNet18 visual backbone deployed in the realistic Habitat 3.0 environment.

Habitat-Lab Experiment.

Habitat [247, 229] is a high-performance simulator designed for training and evaluating deep reinforcement learning models, particularly in 3D environments for embodied AI tasks. In the *Social Navigation* task [247, 229] employed in our evaluation, we have a robotic agent that has to follow a humanoid without any collision. This evaluation considers the publicly released trained agent from the Habitat-Lab GitHub repository (`https://github.com/facebookresearch/habitat-lab`). In detail, we identify three unsafe situations (depicted in Fig.3.7) from a dataset of trajectories observed during evaluation, and we defined a set of provably safe actions and potentially safe actions (i.e., tolerable but less preferred than the safe ones), setting different thresholds for the linear and angular velocities that the agent should respect. Hence, the safe output abstraction C_1 here represents more stringent velocity thresholds, while the *abstract safe* output abstractions C_2 consider more relaxed ones. For instance, if the agent in the first unsafe scenario only slowly moves backward and turns around, it can still be tolerable (i.e., abstract safe) but is less preferable than moving backward with a higher linear velocity and no angular velocity, i.e., without turning around. If the agent does not respect any of the output abstractions, it is considered unsafe.

For the sake of clarity, in Fig. 3.8, we only report the abstraction for the first unsafe situation, but a similar strategy is applied to all the other situations considered using different thresholds as described in the following. Specifically, the original output space for the robotic agent includes linear and angular velocities in $\mathbb{R}$. To create a hierarchical abstraction of the output space, we first discretize it at the desired maximum resolution, then divide the actions into a set of discrete velocity classes, denoted as $\mathbf{V}$. Each class V_i in $\mathbf{V}$ is defined as $V_i = \{(v, w) \mid v_{i_{\min}} \leq v \leq v_{i_{\max}}, w_{i_{\min}} \leq$

Fig. 3.7: Three unsafe scenarios from the Habitat Lab experiments: On the left, the humanoid approaches the robot, which should move backward to avoid collision. In the center, the humanoid moves away, and the robot should follow while avoiding obstacles. On the right, the humanoid approaches from behind, and the robot should turn to avoid an unexpected collision while searching for the humanoid.

Situation	Safe output (v, ω)	Abstract safe output (v, ω)
1	$(-\infty, -0.4]$, $[0.0, 0.1]$	$(-\infty, -0.4]$, $(-\infty, +\infty)$
2	$[0.4, +\infty)$, $[0.0, 0.1]$	$[0.0, +\infty)$, $(-\infty, +\infty)$
3	$[0.0, +\infty)$, $[0.1, +\infty)$	$[-0.4, +\infty)$, $(-\infty, +\infty)$

Table 3.1: Definition of the safe and abstract safe output bounds for the Habitat experiments. Here, $-\infty, +\infty$ indicates any possible negative and positive robot velocities, respectively.

$w \leq w_{i_{\max}}\}$, where v and ω represent the linear and angular velocities, respectively, bounded by predefined lower and upper thresholds. Using these discrete classes, we can construct the first level of hierarchical abstraction, C_1, as illustrated in Fig. 3.8. We can further aggregate elements from C_1, resulting in an abstract classification within C_2.

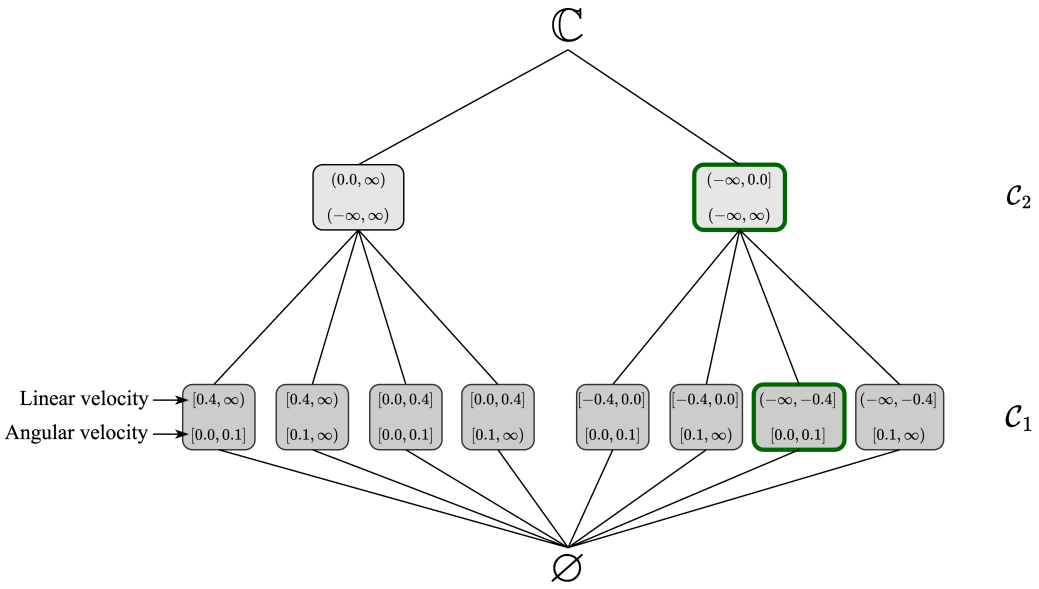


Fig. 3.8: Abstraction hierarchy used in the Habitat Lab experiment for *Situation 1*. For C_1 and C_2 we highlighted in dark green the *safe* and *abstract safe* desired outputs.

Our empirical evaluation is based on three different types of adversarial attacks (reported in Fig. 3.9).

In detail, we consider:

- **Light Attacks**: In this attack, inspired by [106], a random region of the original image is selected, and a natural light effect is simulated by proportionally increasing the pixel values within the patch, effectively overlaying it up to a complete white patch. In practice, this attack is performed by selecting a patch P in the image and perturbing its pixels according to the formula:

$$p'_i = \min\{p_i + \epsilon \cdot (1.0 - p_i), 1.0\}, \quad \forall p_i \in P.$$

where p_i represents the original pixel value, and p'_i is the perturbed value.

- **ϵ-ball Perturbations**: adopted in all of our experiments, which define ℓ_∞ balls with radius ϵ to simulate Gaussian noise applied over all input features.
- **Sensor Rupture**: This attack is modeled by selecting a patch P in the image and setting each pixel within the patch to black (i.e., a value of 0) with a probability ϵ. This corresponds to restricting the affected pixels to the closed interval $[0, 0]$ during verification. Mathematically, we define this attack as:

$$p'_i = \begin{cases} 0 & \text{with probability } \epsilon, \\ p_i & \text{with probability } (1 - \epsilon), \end{cases} \quad \forall p_i \in P,$$

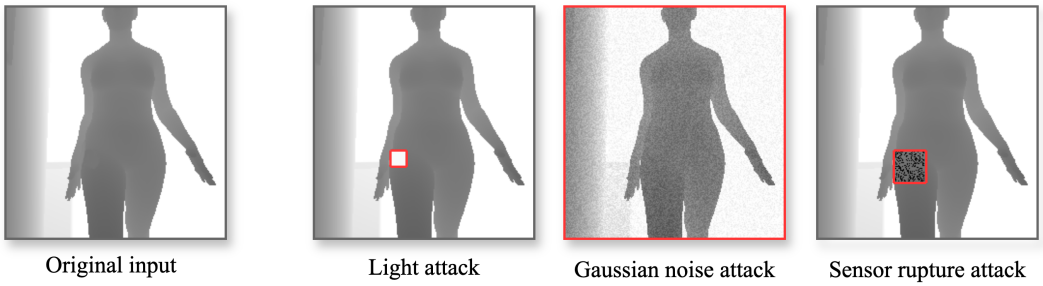


Fig. 3.9: A visual representation of each type of attack used in our empirical evaluation applied on the same arm depth camera input from the habitat-lab experiment. On the left is the original input before the adversarial attack. On the right, the different attacks are highlighted in red: the "light patch" attack with patch size 16×16 and ϵ set to 1.0, the ℓ_∞ ϵ-ball attack with ϵ set to 0.08 applied to all the input features, and finally, the "sensor rupture" attack with patch size 32×32 and ϵ set to 0.3.

Results in Tab. 3.2 demonstrate the varying impact of different attacks across situations. In *Situation 1*, “light patch" and ℓ_∞ attacks exhibit similar effects, maintaining abstract safe behavior under relaxed thresholds for linear and angular velocities, whereas “sensor rupture" attacks result in unsafe behavior by exceeding the linear velocity threshold, risking a collision. For *Situation 2*, “light patch" attacks lead to safe behavior under stringent thresholds, while ℓ_∞ and “sensor rupture" attacks produce only abstract safe outcomes by meeting relaxed thresholds. In *Situation 3*, all attacks result in safe behavior, respecting the imposed thresholds. We highlight that our novel problem formulation allows for a more nuanced understanding of the impact of different attacks. For example, in *Situation 1*, standard verification would classify all attacks with the same impact (all producing unsafe results) due to the lack of concrete, safe outcomes. In contrast, our approach reveals that “light patch" and ℓ_∞ perturbations have a lesser impact than “sensor rupture". This experiment demonstrates how the ADV problem can be leveraged in realistic safety-critical tasks to develop a pipeline for ranking adversarial attacks based on their impact on the output abstractions associated with the unsafe situation verified.

Situation	Perturb.	Patch size	ϵ	Safe	Abstract safe	Unsafe
1	Light patch	16×16	1.0		✓	
2	Light patch	16×16	1.0	✓		
3	Light patch	16×16	1.0	✓		
1	ℓ_∞	-	0.08		✓	
2	ℓ_∞	-	0.08		✓	
3	ℓ_∞	-	0.08	✓		
1	Sensor rupture	32×32	0.3			✓
2	Sensor rupture	32×32	0.3		✓	
3	Sensor rupture	32×32	0.3	✓		

Table 3.2: Empirical results on Habitat Lab benchmark.

CIFAR10 Experiment.

In this experiment, we test the effectiveness of the ADV to understand the impact of different ℓ_∞ ϵ-ball perturbations on different output abstractions. Specifically, we define three distinct degrees of output abstractions to evaluate robustness to perturbations, which we report in Fig. 3.10.

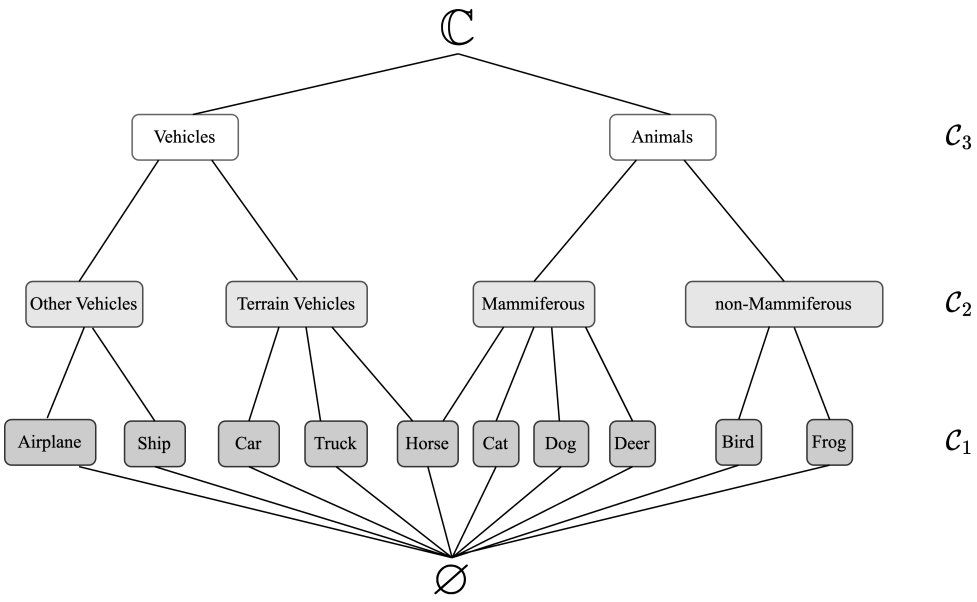


Fig. 3.10: Abstraction hierarchy used in the CIFAR10 experiment.

The first output abstraction corresponds to the precise classification modeled by $\mathcal{C}_1 \stackrel{\text{def}}{=} \{\mathbb{C}, \{\text{Airplane}\}, \{\text{Car}\}, \{\text{Bird}\}, \{\text{Cat}\}, \{\text{Deer}\}, \{\text{Dog}\}, \{\text{Frog}\}, \{\text{Horse}\}, \{\text{Ship}\}, \{\text{Truck}\}, \varnothing\}$, where we do not allow any misclassification. Then, we start by aggregating some elements to create a set of categories, modeling some degree of acceptable error in the classification. With this purpose, we define

ϵ Perturb.	Abstraction	Safe	Abstract Safe	Unsafe	Timeout
1/255	C_1	46	-	4	0
1/255	C_2	46	2	2	0
1/255	C_3	46	4	0	0
2/255	C_1	27	-	11	12
2/255	C_2	27	9	6	8
2/255	C_3	27	18	1	4

Table 3.3: Empirical results on CIFAR10 benchmark.

$C_2 \stackrel{\text{def}}{=} \{\mathbb{C}, \text{Other Vehicles}, \text{Terrain Vehicles}, \text{Mammiferous}, \text{non-Mammiferous}, \{\text{Horse}\}, \varnothing\}$ where the order relation depicted in Fig. 3.10 determines the set composition of the given categories. Note that C_2 ignores all the precise classes in C_1 except *Horse* being in both *Terrain Vehicles* and *Mammiferous*. This abstraction allows misclassifications within the same category but not between different categories. For example, misclassifying a *cat* as a *dog* is acceptable since both belong to the *Mammiferous* category, but misclassifying a *cat* as a *car*, which belongs to the *Terrain Vehicles* category, is not. Finally, we have the last level of aggregation, where we distinguish between *animals* to *vehicles*. Formally, we are considering $C_3 \stackrel{\text{def}}{=} \{\mathbb{C}, \text{Vehicle}, \text{Animal}, \{\text{Horse}\}, \varnothing\}$, where from Fig. 3.10 we can derive the definition of the class abstract categories. For each instance, we set a timeout verification threshold of 100 seconds.

We present the empirical results of the first experiment in Tab. 3.3. When verifying C_1 (i.e., with concrete classes), increasing the ϵ perturbation results in fewer safe instances and generally more timeouts. However, by introducing output abstraction with C_2 or C_3, we observe a reduction in the number of timeouts and a better reflection of the DNN's robustness. For example, with $\epsilon = 1/255$, while standard formal verification identifies four unsafe misclassifications, the DNN reliably distinguishes between *animals* and *vehicles* and nearly always differentiates between the finer subcategories of C_1, highlighting its actual robustness across these classifications. These results also support our hypothesis regarding the practical scalability of abstract verification, where providing multiple output levels can reduce the number of *unknown* outcomes, thereby lowering the likelihood of incurring timeouts. This is highlighted, for instance, in Tab. 3.3, where under an increasing perturbation of $\epsilon = 2/255$, we observe fewer timeout instances as the level of abstraction increases.

Summary. In this first section, we introduced the Abstract DNN-Verification, extending the standard formal verification of neural networks to include a hierarchical structure of safety and robustness properties. By allowing multiple levels of output abstraction, our approach addresses limitations in traditional verification methods, which rely on binary classifications of safe or unsafe outputs. This enhanced framework enables a more expressive analysis of deep neural networks, especially in complex scenarios where traditional safety properties are hard to write or to verify. We also establish the relationship and advancements with the concept of *weakened* robustness introduced in [82]. Importantly, we show how this formulation

can be adapted to realistic benchmarks and complex tasks to rank adversarial attacks based on the impact across different output abstraction levels and get a deeper insight into the safety degree of neural networks.

Nonetheless, the proposed approach is still based on provable over-approximations, which can inherently suffer from scalability limitations, as evidenced by the NP-hardness of the problem. To address this limitation, we argue that adopting a probabilistic perspective can provide scalable alternatives that still offer provable guarantees. In the following section, we explore a probabilistic perspective on the robustness verification of deep neural networks, aiming to balance tractability and reliability in high-dimensional settings.

3.2 Probabilistically Tightened Linear Relaxation-based Perturbation Analysis for Neural Network Verification

In this section, we focus on robustness verification, which aims to guarantee that a model's output remains consistently robust under small predefined input perturbations around a given point $\boldsymbol{x}_0$, typically defined as an ℓ_p ball of ε radius around the input $\boldsymbol{x}_0$, i.e., the set $\mathcal{P} = \mathcal{P}_{\boldsymbol{x}_0,\epsilon} = \{\boldsymbol{x} |\ ||\boldsymbol{x} - \boldsymbol{x}_0||_p \leq \epsilon\}$. A standard approach to proving the robustness property is to first encode the desired output of the classifier as the sign of the value f computed by a single (output) node, i.e., so that the correct classification in $\boldsymbol{x}_0$ corresponds to $f(\boldsymbol{x}_0) > 0$. Then, the robustness verification becomes equivalent to checking that $\min_{\boldsymbol{x}\in\mathcal{P}} f(\boldsymbol{x})$, is positive [275].

However, due to the non-linear and non-convex nature of the DNN, solving this problem is, in general, NP-hard [131, 283]. A recent line of works called linear relaxation-based perturbation analysis (LiRPA) algorithms [297, 291, 275, 290] proposes a formal analysis based on a sound linear relaxation of the DNN. The idea is to compute relaxations of all *sources of non-linearity* in the network so as to obtain two linear functions providing respectively a lower and an upper bound of the DNN's output, which are then used in the robustness certification. Nonetheless, these methods use overapproximation techniques to satisfy the worst-case setting, where no exceptions are allowed. While this strict approach ensures absolute safety within the certified input space, it faces crucial limitations. In fact, the efficiency of LiRPA approaches, and, in general, any FV methods, scales poorly with the size of the model. Additionally, formal methods fail to offer meaningful robustness information when input perturbations exceed the certified bounds. As discussed in [284], relying solely on formal verification can lead to overly conservative outcomes, particularly when adversarial examples are rare or when prior knowledge of the input distribution is available. This stems from the exact nature of formal solvers, which treat all violations equally: even a single adversarial input—such as a one-pixel change—invalidates the entire region, without distinguishing between isolated, low-probability cases and large, semantically significant unsafe regions.

Similar to recent approaches [284, 49], to overcome these critical limitations, we propose a probabilistic perspective that allows for an infinitesimal trade-off in certainty while providing the likelihood of such violations, enabling more nuanced and practical assessments of robustness that better reflect real-world risks. Importantly,

the goal of this work is not to compare probabilistic methods with provable ones, as they are fundamentally different in nature. Rather, our objective is to propose a novel, complementary solution that enhances formal methods by offering additional safety information about the models under evaluation for particular challenging instances to be verified.

Building on this foundation, we explore Xu et al. [291] speculation that estimates as tightly as possible reachable sets, i.e., the range of possible values at each hidden node given bounded inputs, could significantly enhance verification efficiency by yielding sharper final linear bounds. In this perspective, we address two key research questions: (i) *How can we compute (probabilistic) reachable sets that yield tighter intermediate and output bounds than existing methods, while remaining computationally efficient?* (ii) *What theoretical guarantees can we establish for linearized layers using (probabilistically sound) reachable set?*

Our Contributions. We propose **P**robabilistically **T**ightened **Li**near **R**elaxation-based **P**erturbation **A**nalysis (`PT-LiRPA`), a novel framework for computing tight linear lower and upper bounds combining existing LiRPA methods with a sampling-based reachable set estimation strategy. Unlike other related probabilistic works [284, 49], our approach does not rely on specific input distributions or attacks. Specifically, in this work, we start considering statistical results on tolerance limits [286] to provide probabilistic guarantees on estimating the minimum and maximum values of a neural network's output. Using the approach of Wilks [286], we are guaranteed that, for any $R, \psi \in (0, 1)$, with probability ψ, at most, a fraction $(1-R)$ of points from a possibly infinitely large sample in the perturbation region $\mathcal{P}$ may violate the estimated bounds obtained from an initial sample, whose size n is explicitly computable from the desired parameters ψ and R. Hence, our intuition is to extend this approach to also compute an estimation of reachable sets with a specified confidence level. Nonetheless, while the result of [286] quantitatively bounds the error of the sample-based procedure by predicting the fraction of potential violations in future samples, it results in "weaker" guarantees with respect to other known probabilistic approaches. In particular, this approach does not provide information on the magnitude of these violations with respect to the estimated bounds. Specifically, in any estimated reachable set for a possibly existing fraction $(1-R)$ of points drawn from $\mathcal{P}$, the probability of violating the bound might in principle be uncontrollably large. To address such an issue, following results on *extreme value theory* (EVT) [92, 58], we extend Wilks [286]' probabilistic guarantees and present two novel bounds on the magnitude of potential errors between the true minimum (Theorem 3.3) and the sample-based estimated one (Theorem 3.4). Notably, our final result proves that with negligible computational overhead, each node's reachable set computed using random samples represents with high probability the actual domain of that node for any $x \in \mathcal{P}$. Hence, we show that by employing this sampling-based procedure to compute probabilistically tight reachable set bounds in the neural network and integrating these into the linearization used by LiRPA-based formal verification methods, we are able to obtain significantly tighter lower and upper linear bounds of a neural network's output, while preserving the verification soundness for any input in the perturbation region and specified confidence level.

To assess the benefit and effectiveness of our novel framework, we perform an extensive empirical evaluation. We first compare our approach on neural networks trained on MNIST and CIFAR datasets with PROVEN [284] and Randomized Smoothing [49], the most closely related probabilistic verification approaches. In addition, as a ground truth, we also consider a set of state-of-the-art worst-case robustness verification approaches, namely CROWN[297], α-CROWN [291], β-CROWN [275], and GCP-CROWN [298]. This first set of experiments shows that with a very high confidence (i.e., $\geq$ 99%), `PT-LiRPA` improves the certified lower bound of ε perturbation tolerated by the models up to 3.31x and 2.26x compared with both the corresponding probabilistic approaches of [284, 49] and up to 3.62x w.r.t. the worst-case analysis results. Finally, in the second set of experiments, we demonstrate that for challenging instances from the International Verification of Neural Networks Competition (VNN-COMP) [198, 35], where state-of-the-art formal verification methods fail to produce a conclusive result, our `PT-LiRPA`, with a quantifiable level of confidence, represents a valuable resource in providing safety information.

3.2.1 Probabilistically Tightened LiRPA via Underestimation

In this section, we theoretically investigate whether and how it is possible to compute tight intermediate reachable sets, which directly impact the linear output bound computation. As highlighted in Alg. 3, the entire linearization process crucially depends on the bounds computed at line 3. However, computing exact values for such bounds is generally infeasible, as the problem has been shown to be NP-hard [131, 283]. Motivated by the speculation of Xu et al. [291], we therefore explore efficient alternatives for approximating the exact (unknown) values of these intermediate bounds as tightly as possible. To this end, we study the impact of a sampling-based approach and what type of probabilistic guarantees can be achieved with it. In detail, we begin employing the *statistical prediction of tolerance limits* results [286], which allows a closed-form derivation of the required sample size to achieve a desired confidence level. However, as we will show, this method only quantifies the fraction of the perturbation region where the guarantees may fail, without addressing the magnitude of potential violations relative to the estimated bounds. Consequently, the resulting probabilistic certificates are weaker than those provided by related approaches such as [284, 49], where guarantees hold across the entire perturbation region. To mitigate this limitation, we first introduce a qualitative extension of Wilks' guarantees adapted to our setting. While novel, this bound can become overly loose in high-dimensional scenarios. To address this issue, we then develop a new theoretical and practical result based on *extreme value theory* [92], which allows us to tightly bound the magnitude of possible violations with respect to the estimated bounds.

We now present all the theoretical and practical components to compute tightened bounds with probabilistic guarantees within our `PT-LiRPA` approach. Our approach is based on a statistical methodology that allows us to compute estimates on the neural network's output in the form $\min_{\boldsymbol{x}\in\mathcal{P}} f(\boldsymbol{x})$ and $\max_{\boldsymbol{x}\in\mathcal{P}} f(\boldsymbol{x})$, which hold with high confidence. In particular, we employ the tools of *statistical prediction of tolerance limits* of [286]. Fix a node $z_j^{(i)}$ in the neural network and, as before, let

$z_j^{(i)}(\boldsymbol{x})$ be the preactivation value for the node when the input to the network is a vector $\boldsymbol{x}$. Let $\boldsymbol{x}_1, \ldots \boldsymbol{x}_n$ be n points (vectors) independently and uniformly sampled from the perturbation set of interest $\mathcal{P}$. We compute $z_j^{(i)}$'s (estimated) pre-activated bounds as:

$$\overline{l}_j^{(i)} = \min_{k=1,\ldots,n} z_j^{(i)}(\boldsymbol{x}_k); \qquad \underline{u}_j^{(i)} = \max_{k=1,\ldots,n} z_j^{(i)}(\boldsymbol{x}_k),$$

i.e., the minimum and maximum value observed from the propagation of the n random points in that specific neuron $z_j^{(i)}$.

Let $l_j^{*(i)} = \min_{\boldsymbol{x}\in\mathcal{P}} z_j^{(i)}(\boldsymbol{x})$ and $u_j^{*(i)} = \max_{\boldsymbol{x}\in\mathcal{P}} z_j^{(i)}(\boldsymbol{x})$. Clearly, we have

$$\overline{l}_j^{(i)} \geq l_j^{*(i)}; \quad \underline{u}_j^{(i)} \leq u_j^{*(i)}. \tag{3.1}$$

However, based on the results of [286], for any $\psi, R \in (0,1)$, we can choose the sample size n that guarantees that the estimated reachable set is correct with probability ψ for at least a fraction R of points in the perturbation set $\mathcal{P}$. Crucially, this statistical result does not require any knowledge of the probability distribution governing the output of our function of interest. Formally, we have the following.

Lemma 3.1 (Probabilistically tightened reachable sets): *Let n the number of samples employed in the computation and the interval $[\overline{l}_j^{(i)}, \underline{u}_j^{(i)}]$, where $\overline{l}_j^{(i)}$ and $\underline{u}_j^{(i)}$ are the minimum and maximum pre-activation values observed in the sample, respectively. Fix $R \in (0,1)$, then for any further possibly infinite sequence of samples from $\mathcal{P}$, the probability that $[\overline{l}_j^{(i)}, \underline{u}_j^{(i)}]$ is correct[7] for at least a fraction R of points is at least $\psi = n \cdot \int_R^1 x^{n-1}\, dx = (1 - R^n)$.*

Hence, following Lemma 3.1, whose proof follows from citewilks, for any desired confidence level ψ, and lower bound fraction R, we can compute the number n of samples sufficient to obtain the provable probabilistic guarantees on the desired reachable set accuracy. Specifically, we have that if we use a sample of $n \geq \frac{\ln(1-\psi)}{\ln(R)}$ input points and with them we obtain an estimated reachable set $[\overline{l}_j^{(i)}, \underline{u}_j^{(i)}]$, then with probability ψ at most a fraction $(1 - R)$ of points in an indefinitely larger future sample could fall outside such reachable set. By taking into account the total number of neurons in the DNN, and using a independently chosen set of points for each neuron, we can extend the above result to compute reachable sets for each neuron that are simultaneously correct with probability ψ for at least a fraction R of the points in $\mathcal{P}$:

Proposition 3.4: *Consider an N-layer ReLU DNN with m neurons. Fix a confidence level ψ and coverage ratio $R \in (0,1)$. For each intermediate neuron $z_j^{(i)}$, collect n' i.i.d. samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ from the perturbation region $\mathcal{P}$, and compute the approximate reachable set as $\overline{l}_j^{(i)} = \min_{k=1,\ldots,n'} z_j^{(i)}(\boldsymbol{x}_k)$ and $\underline{u}_j^{(i)} = \max_{k=1,\ldots,n'} z_j^{(i)}(\boldsymbol{x}_k)$. If the number of samples used for each neuron satisfies $n' \geq \frac{\ln(1-\psi^{1/m})}{\ln(1-(1-R)/m)}$, then for any*

[7] In the sense that there exists $\mathcal{P}' \subseteq \mathcal{P}$ such that $|\mathcal{P}'|/|\mathcal{P}| \geq R$ and for all $\boldsymbol{x} \in \mathcal{P}'$ it holds that $z_j^{(i)}(\boldsymbol{x}) \in [\overline{l}_j^{(i)}, \underline{u}_j^{(i)}]$.

(possibly infinite) sequence X of inputs sampled independently and uniformly from $\mathcal{P}$, for each neuron $z_j^{(i)}$ with probability at least $\psi^{1/m}$ there is a subsequence X' of X of size $|X'| \geq \left(1 - \frac{1-R}{m}\right)|X|$ such that for each $\boldsymbol{x} \in X'$ the estimated reachable set is sound, i.e., $z_j^{(i)}(\boldsymbol{x}) \in [\underline{l}_j^{(i)}, \underline{u}_j^{(i)}]$.

We will now show that we can combine the (probabilistically valid) estimation computed on the reachable sets of individual neurons using a LiRPA approach, so as to obtain, first, a probabilistically valid over-approximation of any ReLU layer in the DNN and thus on the final network's lower and upper outputs. In detail, we begin by proving that the estimated reachable sets used to produce the vectors $\boldsymbol{A}$ and $\boldsymbol{b}$, together with the linearization applied to each ReLU layer of the network (as in Sec. 2.3.4), yield a probabilistically sound over-approximation that covers at least a fraction R of the perturbation region $\mathcal{P}$.

Lemma 3.2 (ReLU Layer Relaxation using `PT-LiRPA`): *Consider a ReLU DNN with m neurons distributed over N layers, and $\mathcal{P}$ a perturbation region of interest. Fix confidence and coverage parameter $\psi, R \in (0,1)$. Fix a layer $i \in \{1, \ldots, N-2\}$, and for each $j = 1, \ldots, d_i$ compute an estimated reachable set $[\underline{l}_j, \underline{u}_j]$ for neuron $z_j^{(i)}$, using $n' \geq m\frac{ln(1-\psi^{1/m})}{ln(1-(1-R)/m)}$ independently and uniformly sampled point from $\mathcal{P}$, as by Proposition 3.4. Let $\underline{\boldsymbol{l}} = (\underline{l}_1, \ldots, \underline{l}_{d_i})$ and $\underline{\boldsymbol{u}} = (\underline{u}_1, \ldots, \underline{u}_{d_i})$.*

Let $\boldsymbol{A}, \boldsymbol{b}$ be the vectors of the linear bounds coefficients and biases (inductively) computed for the ReLU layer $i+1$, and such that, with probability $\geq \psi^{\frac{d_{i+1}+d_{i+2}+\cdots+d_N}{m}}$, for any $n \in \mathbb{N}$ in any sequence of n points uniformly and independently sampled from $\mathcal{P}$, for at least $n \times \left(1 - \frac{1-R}{m}\sum_{j=i+1}^{N} d_j\right)$ points $\boldsymbol{x}$ in X it holds that

$$f(\boldsymbol{x}) \geq \boldsymbol{A}^T ReLU(\boldsymbol{v_x}) + \boldsymbol{b}, \tag{3.2}$$

where $\boldsymbol{v_x}$ is the vector of pre-activation values of the neurons in the layer i when the input is $\boldsymbol{x}$. Then,

1. *with probability $\geq \psi^{\frac{d_i}{m}}$, it holds that for any $n \in \mathbb{N}$ in any sequence X of n points uniformly and independently sampled from $\mathcal{P}$, for at least $n \times \left(1 - \frac{1-R}{m} d_i\right)$ points $\boldsymbol{x}$ in X it holds that:*

$$\boldsymbol{A}^T ReLU(\boldsymbol{v_x}) \geq \boldsymbol{A}^T(\underline{\boldsymbol{D}}^* \boldsymbol{v_x} + \underline{\boldsymbol{b}}^*) \tag{3.3}$$

where

$$\underline{\boldsymbol{D}}^* = \begin{cases} 1 & \underline{l}_j \geq 0, \\ 0 & \underline{u}_j \leq 0, \\ \alpha_j & \underline{u}_j > 0 > \underline{l}_j \text{ and } \boldsymbol{A}_j \geq 0, \\ \frac{\underline{u}_j}{\underline{u}_j - \underline{l}_j} & \underline{u}_j > 0 > \underline{l}_j \text{ and } \boldsymbol{A}_j < 0 \end{cases} \quad \underline{\boldsymbol{b}}^* = \begin{cases} 0 & \underline{l}_j > 0 \text{ or } \underline{u}_j \leq 0, \\ 0 & \underline{u}_j > 0 > \underline{l}_j \text{ and } \boldsymbol{A}_j \geq 0, \\ -\frac{\underline{u}_j \underline{l}_j}{\underline{u}_j - \underline{l}_j} & \underline{u}_j > 0 > \underline{l}_j \text{ and } \boldsymbol{A}_j < 0. \end{cases}$$

2. *with probability $\geq \psi^{\frac{d_i + d_{i+1} + \cdots + d_N}{m}}$, for vectors $\boldsymbol{A}' = \boldsymbol{A}^T \boldsymbol{D}^*$ and $\boldsymbol{b}' = \boldsymbol{b} + \boldsymbol{A}^T \boldsymbol{b}^*$ it holds that for any $n \in \mathbb{N}$ in any sequence X of n points uniformly and independently sampled from $\mathcal{P}$, for at least $n \times \left(1 - \frac{1-R}{m}\sum_{j=i}^{N} d_j\right)$ points $\boldsymbol{x}$ in X we have that*

$$f(\boldsymbol{x}) \geq \boldsymbol{A}' ReLU(\boldsymbol{v_x}) + \boldsymbol{b}'. \tag{3.4}$$

Proof. The proof closely follows the analogous result at the basis of the LiRPA approach [297]. The only (crucial) difference is that we construct $\underline{\boldsymbol{D}}^*$ and $\underline{\boldsymbol{b}}^*$, i.e., the diagonal matrix, and the bias vector meant to provide a linear lower bound on the post-activation values of layer i, using the vectors of estimated reachable sets of the nodes in the layer, instead of their actual reachable sets.

Let X be a sequence of n points independently and uniformly sampled from $\mathcal{P}$.

For each $j = 1, \ldots, d_i$, let

$$X_j = \{\boldsymbol{x} \in X \mid z_j^{(i)}(\boldsymbol{x}) \notin [\bar{l}_j, \underline{u}_j]\},$$

and let $\mathcal{E}_j$ be the event

$$\mathcal{E}_j = \{|X_j| \leq \frac{1-R}{m}|X|\}.$$

Because of the way we compute the estimated reachable sets $[\bar{l}_j, \underline{u}_j]$ (Proposition 3.4) we have that for each j

$$Pr[\mathcal{E}_j] \geq \psi^{1/m}.$$

Moreover these events are independent, hence

$$Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \cdots \wedge \mathcal{E}_{d_i}] \geq \psi^{d_i/m}.$$

Let

$$X_{OK} = X \setminus \left(\bigcup_{j=1}^{d_i} X_j \right) = \{\boldsymbol{x} \in X \mid \forall j, z_j^{(i)} \in [\bar{l}_j, \underline{u}_j]\}.$$

Then,

$$|X_{OK}| \geq |X| - \sum_{j=1}^{d_i} |X_j|,$$

and in particular, if for each j it holds that $|X_j| \leq \frac{1-R}{m}|X|$–i.e., when the event $\mathcal{E}_1 \wedge \cdots \wedge \mathcal{E}_{d_i}$ occurs–we have $|X_{OK}| \geq \left(1 - \frac{(1-R)}{m} d_i\right)|X|$.

We can conclude that the probability of the event $\mathcal{E}_{OK} = \{|X_{OK}| \geq (1 - \frac{(1-R)d_i}{m})|X|\}$ satisfies

$$Pr[\mathcal{E}_{OK}] \geq Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \cdots \wedge \mathcal{E}_{d_i}] \geq \psi^{d_i/m}.$$

In words, we have shown that with probability $\psi^{1/m}$ for each point $\boldsymbol{x}$ in a fraction of X of size $(1 - \frac{(1-R)d_i}{m})|X|$, for all nodes $z_j^{(i)}$ in layer i, the pre-activation value $z_j^{(i)}(\boldsymbol{x})$ induced by $\boldsymbol{x}$ is contained in the estimated reachable sets $[\bar{l}_j, \underline{u}_j]$.

In order to simplify the notation, let us fix an $\boldsymbol{x}$ from X_{OK} and let $\boldsymbol{v} = \boldsymbol{z}^{(i)}(\boldsymbol{x})$ be the vector of pre-activation values induce by $\boldsymbol{x}$ in the nodes of layer i.

We will show that for the $\underline{\boldsymbol{D}}^*$ and $\underline{\boldsymbol{b}}^*$ defined in the statement, we have that for each j it holds that $\boldsymbol{A}_j(ReLU(\boldsymbol{v}_j)) \geq \boldsymbol{A}_j(\underline{\boldsymbol{D}}^*_{j,j}\boldsymbol{v}_j + \underline{\boldsymbol{b}}^*_j)$, which immediately implies the desired result, since $\boldsymbol{A}^T ReLU(\boldsymbol{v}) = \sum_j \boldsymbol{A}_j(ReLU(\boldsymbol{v}_j)) \geq \boldsymbol{A}_j(\underline{\boldsymbol{D}}^*_{j,j}\boldsymbol{v}_j + \underline{\boldsymbol{b}}^*_j) = \boldsymbol{A}^T(\underline{\boldsymbol{D}}^*\boldsymbol{v}_{\boldsymbol{x}} + \underline{\boldsymbol{b}}^*)$.

We start by noticing that the following inequalities hold

$$\boldsymbol{l}_j \leq \boldsymbol{l}_j^* \leq \bar{\boldsymbol{l}}_j < 0 < \underline{\boldsymbol{u}}_j \leq \boldsymbol{u}_j^* \leq \boldsymbol{u}_j, \tag{3.5}$$

where $[\boldsymbol{l}_j^*, \boldsymbol{u}_j^*]$ is the actual reachable set for the j-th node of the layer under consideration.

We split the argument into three cases according to the sign of the estimated values $\bar{\boldsymbol{l}}_j$ and $\underline{\boldsymbol{u}}_j$. Moreover, we split each case into subcases according to the sign of $\boldsymbol{A}_j$ and $\boldsymbol{v}_j$.

Case 1. $\underline{\boldsymbol{u}}_j > 0 > \bar{\boldsymbol{l}}_j$

From inequality 3.5, we know that since we are underestimating true bounds $[\boldsymbol{l}_j^*, \boldsymbol{u}_j^*]$, the ReLU node is actually unstable, even for any LiRPA approach. Comparing the diagonal coefficients $\frac{\underline{\boldsymbol{u}}_j}{\underline{\boldsymbol{u}}_j - \bar{\boldsymbol{l}}_j}$ with $\frac{\boldsymbol{u}_j}{\boldsymbol{u}_j - \boldsymbol{l}_j}$ and the biases $-\frac{\underline{\boldsymbol{u}}_j \bar{\boldsymbol{l}}_j}{\underline{\boldsymbol{u}}_j - \bar{\boldsymbol{l}}_j}$ with $-\frac{\boldsymbol{u}_j \boldsymbol{l}_j}{\boldsymbol{u}_j - \boldsymbol{l}_j}$ of `PT-LiRPA` and any LiRPA cannot be helpful. The relation between the coefficients strongly depends on the quality of the bounds computed, and we cannot draw any direct conclusion since in some cases $\underline{\boldsymbol{D}} > \underline{\boldsymbol{D}}^*$ and in some cases not. Hence, we need to proceed by subcases.

Subcase 1.1. $\boldsymbol{A}_j < 0$ and $\boldsymbol{v}_j < 0$.

Since $\boldsymbol{v}_j < 0$ then $ReLU(\boldsymbol{v}_j) = 0$. Moreover, we have

$$(\underline{\boldsymbol{D}}_{j,j}^* \boldsymbol{v}_j + \underline{\boldsymbol{b}}_j^*) = \frac{\underline{\boldsymbol{u}}_j}{\underline{\boldsymbol{u}}_j - \bar{\boldsymbol{l}}_j} \boldsymbol{v}_j + \Big(- \frac{\underline{\boldsymbol{u}}_j \bar{\boldsymbol{l}}_j}{\underline{\boldsymbol{u}}_j - \bar{\boldsymbol{l}}_j} \Big) = \frac{\underline{\boldsymbol{u}}_j (\boldsymbol{v}_j - \bar{\boldsymbol{l}}_j)}{\underline{\boldsymbol{u}}_j - \bar{\boldsymbol{l}}_j} \geq 0,$$

where the last inequality holds since $\boldsymbol{v}_j \in [\bar{\boldsymbol{l}}_j, \underline{\boldsymbol{u}}_j]$ implies $\boldsymbol{v}_j - \bar{\boldsymbol{l}}_j \geq 0$.

Multiplying both sides by $\boldsymbol{A}_j < 0$ and using $ReLU(\boldsymbol{v}_j) = 0$, we get

$$\boldsymbol{A}_j (ReLU(\boldsymbol{v}_j)) = 0 \geq \boldsymbol{A}_j (\underline{\boldsymbol{D}}_{j,j}^* \boldsymbol{v}_j + \underline{\boldsymbol{b}}_j^*)$$

Subcase 1.2. $\boldsymbol{A}_j < 0$ and $\boldsymbol{v}_j \geq 0$.

Since $\boldsymbol{v}_j \geq 0$ then $ReLU(\boldsymbol{v}_j) = \boldsymbol{v}_j$. Moreover, we have

$$(\underline{\boldsymbol{D}}_{j,j}^* \boldsymbol{v}_j + \underline{\boldsymbol{b}}_j^*) = \frac{\underline{\boldsymbol{u}}_j}{\underline{\boldsymbol{u}}_j - \bar{\boldsymbol{l}}_j} \boldsymbol{v}_j + \Big(- \frac{\underline{\boldsymbol{u}}_j \bar{\boldsymbol{l}}_j}{\underline{\boldsymbol{u}}_j - \bar{\boldsymbol{l}}_j} \Big) \geq \boldsymbol{v}_j,$$

where the last inequality holds since under the standing hypothesis the coefficient of $\boldsymbol{v}_j$ in the left hand side is ≥ 1 and term $-\frac{\underline{\boldsymbol{u}}_j \bar{\boldsymbol{l}}_j}{\underline{\boldsymbol{u}}_j - \bar{\boldsymbol{l}}_j}$ is non-negative.

Multiplying both sides by $\boldsymbol{A}_j < 0$ and using $ReLU(\boldsymbol{v}_j) = \boldsymbol{v}_j$, we get

$$\boldsymbol{A}_j (ReLU(\boldsymbol{v}_j)) = \boldsymbol{A}_j \boldsymbol{v}_j \geq \boldsymbol{A}_j (\underline{\boldsymbol{D}}_{j,j}^* \boldsymbol{v}_j + \underline{\boldsymbol{b}}_j^*).$$

Subcase 1.3. $\boldsymbol{A}_j \geq 0$ and $\boldsymbol{v}_j < 0$.

We have

$$ReLU(\boldsymbol{v}_j) = 0 \geq \alpha_j \boldsymbol{v}_j + 0 = \underline{\boldsymbol{D}}_{j,j}^* \boldsymbol{v}_j + \underline{\boldsymbol{b}}_j^*$$

, since $0 < \alpha_j$. Hence, multiplying both sides by $\boldsymbol{A}_j \geq 0$ we obtain again the desired inequality.

Subcase 1.4. $\boldsymbol{A}_j \geq 0$ and $\boldsymbol{v}_j \geq 0$.

We have

$$ReLU(\boldsymbol{v}_j) = \boldsymbol{v}_j \geq \alpha_j \boldsymbol{v}_j + 0 = \underline{\boldsymbol{D}}^*_{j,j} \boldsymbol{v}_j + \underline{\boldsymbol{b}}^*_j$$

since $\alpha < 1$. Again, multiplying both sides by $\boldsymbol{A}_j \geq 0$ we obtain again the desired inequality.

This concludes the first case.

$z^{(i)}_j$ outside the reachable set $[\bar{l}_j, \underline{u}_j]$. Thus, with probability $\geq \psi$.

CASE 2. $\bar{\boldsymbol{l}}_j > 0$

Since $\boldsymbol{v}_j \in [\bar{l}_j, \underline{u}_j]$, with $\bar{l}_j > 0$ we have $ReLU(\boldsymbol{v}_j) = \boldsymbol{v}_j$.

Moreover, we have

$$\underline{\boldsymbol{D}}^*_{j,j} \boldsymbol{v}_j + \underline{\boldsymbol{b}}^*_j = 1 \cdot \boldsymbol{v}_j + 0 = \boldsymbol{v}_j = ReLU(\boldsymbol{v}_j)$$

from which, multiplying both sides by $\boldsymbol{A}_j$ yields the desired inequality $\boldsymbol{A}_j ReLU(\boldsymbol{v}_j) \geq \boldsymbol{A}_j(\underline{\boldsymbol{D}}^*_{j,j} \boldsymbol{v}_j + \underline{\boldsymbol{b}}^*_j)$.

CASE 3. $\underline{\boldsymbol{u}}_j < 0$

Since $\boldsymbol{v}_j \in [\bar{l}_j, \underline{u}_j]$ and $\underline{u}_j < 0$ we have $ReLU(\boldsymbol{v}_j) = 0$. Moreover, under the standing hypothesis, we also have $\underline{\boldsymbol{D}}^*_{j,j} \boldsymbol{v}_j + \underline{\boldsymbol{b}}^*_j = 0 = ReLU(\boldsymbol{v}_j)$. Again, multiplying both sides by $\boldsymbol{A}_j$ we have that the desired inequality holds also in this case.

We have proved that for each $\boldsymbol{x}$ in X_{OK} we have $\boldsymbol{A}^T ReLU(\boldsymbol{v}_j) \geq \boldsymbol{A}^T(\underline{\boldsymbol{D}}^* \boldsymbol{v}_x + \underline{\boldsymbol{b}}^*)$. Finally, recalling that with probability $\psi^{d_i/m}$ we have that $|X_{OK}| \geq |X|(1 - \frac{1-R}{m} d_i)$, we have that the previous inequality holds with the desired statistical guarantees, which completes the proof of claim (1).

For proving (2) it is enough to note that the guarantees on $\boldsymbol{A}, \boldsymbol{b}$ and (3.2) hold independently of the choice of the samples leading to the statistical guarantee on (3.3). Hence they simultaneously happen–yielding that (3.4) holds–with the product of the probabilities of (3.2) and (3.3), i.e., $\psi^{\frac{d_i + d_{i+1} + \cdots + d_N}{m}}$. In particular, with this probability, for any $n \in \mathbb{N}$ in any sequence X of n points uniformly and independently sampled from $\mathcal{P}$, neither of (3.2) and (3.3) fails on at least $n \times \left(1 - \frac{1-R}{m} \sum_{j=i}^{N} d_j\right)$ points.

□

As a direct implication of this result, we can show that the neural network's output linear bounds computed using `PT-LiRPA` remain, with high probability, an overestimation of the real lower and upper bound, respectively, for at least a fixed fraction R of the perturbation region under consideration.

Theorem 3.2 (PT-LiRPA weak probabilistic guarantees): *Fix an N-layer ReLU DNN with m neurons with f the function it computes. Then, for any $\psi, R \in (0,1)$ PT-LiRPA, using a total number of $n \geq m\frac{\ln(1-\psi^{1/m})}{\ln(1-(1-R)/m)}$ independent and uniformly distributed input points, computes a linear approximation $\boldsymbol{a}^T_{PT\text{-}LiRPA}(\boldsymbol{x}) + \boldsymbol{c}_{PT\text{-}LiRPA}$ of f such that with probability at least ψ for at least a fraction R of possibly infinite samples of input points, $\boldsymbol{x}$, it holds that*

$$f(\boldsymbol{x}) \geq \boldsymbol{a}^T_{PT\text{-}LiRPA}(\boldsymbol{x}) + \boldsymbol{c}_{PT\text{-}LiRPA}.$$

Proof. The proof then directly follows from Lemma 3.2 and the derivations of Zhang et al. [297]. □

3.2.2 A Qualitative Bound of Statistical Prediction of Tolerance Limit

The result of Wilks [286] allows us to bound the error of the sample-based procedure employed in our PT-LiRPA framework in terms of the number of potential violations in future samples. However, it does not provide any information on the magnitude of such possible violations with respect to the estimated bounds. Additionally, the probabilistic guarantees we provide so far are, broadly speaking, weaker than those offered by related probabilistic methods such as [284, 49], where the guarantees hold for any input $\boldsymbol{x} \in \mathcal{P}$, and not only for a predefined fraction of it. In this section, we aim to complement Wilks [286]' statistical result with a qualitative interpretation that bounds the potential error between the true minimum and its estimate based on samples. Specifically, we leverage known results on *extreme value theory* [92] to strengthen and extend our original guarantee to apply to the entire perturbation set $\mathcal{P}$, thus bringing the proposed solution in line with other state-of-the-art probabilistic approaches.

We start by noticing that the problem of finding an empirical minimizer close to a function's exact minimizer is well established in the statistical literature (see, e.g., [10]). Exploiting quantitative assumptions on the objective function—such as a Lipschitz condition that holds in our setting—can facilitate the bounding of the possible magnitude error in the sample-based minimizer computation approach. In this vein, we begin by providing a first worst-case qualitative bound on the maximum Δ error we can achieve when employing our sampling-based approach to compute the estimated reachable set with respect to the real (unknown) ones. For readability, in the following we simplify the notation $z_j^{(i)}$—which denotes the pre-activation value of the node in position j of layer i—by removing the indices and referring generically to a node's pre-activation value as z. We adopt the same simplification for the corresponding lower and upper bound estimates, denoted as $\overline{l}$ and $\underline{u}$, respectively.

We start with the following result.

Theorem 3.3 (Worst-case excess bound): *Fix a neuron in our N-layer DNN. Let z be the function mapping the input $\boldsymbol{x}$ to the network to the pre-activation value $z(\boldsymbol{x})$ of the neuron of interest. Let $\overline{l} = \min_{k=1,\dots,n} z(\boldsymbol{x}_k)$ be the minimum pre-activation value observed in a sample of n inputs, $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$ independently and uniformly drawn from $\mathcal{P} \subseteq \mathbb{R}^d$. Let $l^* = \min_{\boldsymbol{x}\in\mathcal{P}} z(\boldsymbol{x})$ be the actual minimum pre-activation value achievable for z over all $\boldsymbol{x} \in \mathcal{P}$. Then, if z is Lipschitz continuous, it holds that:*

$$Pr[|\bar{l} - l^*| \leq \Delta] \geq 1 - exp\left(-n\left(\frac{\Delta}{L}\right)^d \cdot \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}\right) \tag{3.6}$$

where L is the Lipschitz constant of the function $z(\cdot)$, d is the dimension of the perturbation region $\mathcal{P}$, Γ is the gamma function, and Δ the maximum error we are interested in bounding.

Proof. Since z is an L-Lipschitz function, then we have that for each $k = \{1, \ldots, n\}$, and $\boldsymbol{x}^*$ being a minimizer of z, i.e., $z(\boldsymbol{x}^*) = l^*$, it holds that

$$|z(\boldsymbol{x}_k) - l^*| \leq L||\boldsymbol{x}_k - \boldsymbol{x}^*||_2.$$

From the function's Lipschitz continuity property, we have

$$\begin{aligned} Pr[|\bar{l} - l^*| \geq \Delta] &= Pr[\forall k\ z(\boldsymbol{x}_k) - z(\boldsymbol{x}^*) \geq \Delta] \\ &\leq Pr[\forall k\ L||\boldsymbol{x}_k - \boldsymbol{x}^*||_2 \geq \Delta] \\ &= Pr[\forall k\ ||\boldsymbol{x}_k - \boldsymbol{x}^*||_2 \geq \frac{\Delta}{L}] \end{aligned}$$

Fix $S = \{\boldsymbol{x} \in \mathcal{P} : ||\boldsymbol{x} - \boldsymbol{x}^*||_2 \leq \Delta/L\}$. This is the sphere of radius Δ/L centered at $\boldsymbol{x}^*$. For a uniformly sampled point $\boldsymbol{x} \in \mathcal{P}$, let $\mu(S) = Pr[\boldsymbol{x} \in S]$. This is equal to the volume of the set S which is proportional to $(\Delta/L)^d \cdot c(d)$, where $c(d) = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}$ is a constant that depends on the dimension d. Hence, by the independence of the $\boldsymbol{x}_k$s, the probability $Pr[\forall k\ \boldsymbol{x}_k \notin S] = (1 - \mu(S))^n$.

$$\begin{aligned} Pr[|\bar{l} - l^*| \geq \Delta] &\leq Pr[\forall k\ ||\boldsymbol{x}_k - \boldsymbol{x}^*||_2 \geq \frac{\Delta}{L}] \\ &= Pr[\forall k\ \boldsymbol{x}_k \notin S] \\ &= (1 - \mu(S))^n \\ &\text{from the fact that } (1-x)^n \approx e^{-nx} \\ &\leq exp\left(-n\left(\frac{\Delta}{L}\right)^d \cdot \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}\right) \end{aligned}$$

Hence, $Pr[|\bar{l} - l^*| \leq \Delta] \geq 1 - Pr[|\bar{l} - l^*| \geq \Delta] \geq 1 - exp\left(-n\left(\frac{\Delta}{L}\right)^d \cdot \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}\right)$. □

Broadly speaking, in the theorem, we bound the probability that the estimated minimizer deviates from the true minimizer by more than a predefined threshold Δ. Owing to the Lipschitz properties of DNNs, this is equivalent to estimating the probability that a random sample of inputs includes at least one point within distance Δ/L of the true minimizer $\mathbf{x}^*$, i.e., an input $\boldsymbol{x}$ in $S = \{\mathbf{x} \in \mathcal{P} : \|\mathbf{x} - \mathbf{x}^*\|_2 \leq \Delta/L\}$. Geometrically, S corresponds to the intersection of the perturbation region $\mathcal{P}$ with a ball of radius Δ/L centered at $\mathbf{x}^*$. If $\mathbf{x}^*$ lies sufficiently far from the boundary of $\mathcal{P}$, then S is exactly such a ball. Otherwise, if $\mathbf{x}^*$ is near the boundary, S becomes a truncated ball, i.e., the intersection with $\mathcal{P}$. This is clearly represented in Fig. 3.11, where we depict a potential perturbation region $\mathcal{P}$ in 3d derived from the original input $\boldsymbol{x}_0$ and the minimizer $\boldsymbol{x}^*$ of f for $\mathcal{P}$.

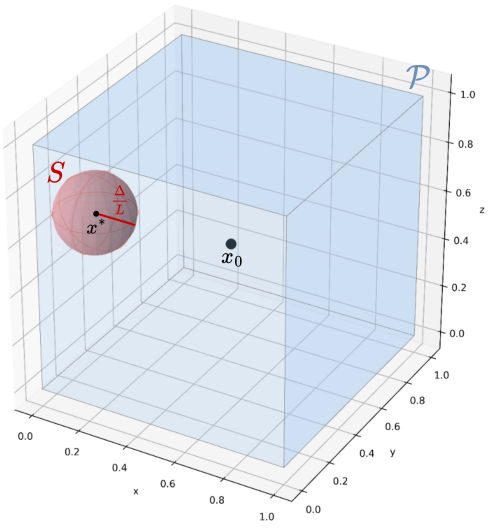


Fig. 3.11: Illustrative representation in 3D of Theorem 3.5.

Clearly, the bound given in Theorem 3.3 accounts for the worst-case scenario and assumes no specific property of the distribution on the input. When the perturbation region has high dimensionality or the DNN has a large Lipschitz constant, this bound may become too loose to offer meaningful insights. Following EVT, and particularly the results of [58], we can provide a tighter bound on the error in the estimation.

Lemma 3.3 (Tighter Qualitative bound): *Fix a neuron in our N-layer DNN and let z be the be the function mapping the input network to the pre-activation value of the neuron. Let $\bar{l} = \min_{k=1,\dots,n} z(\boldsymbol{x}_k)$ the minimum pre-activation value observed in a sample of n inputs, $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$ independently and uniformly drawn from $\mathcal{P} \subseteq \mathbb{R}^d$. Let $l^* = \min_{\boldsymbol{x}\in\mathcal{P}} z(\boldsymbol{x})$, the true minimum pre-activation value achievable over all $\boldsymbol{x} \in \mathcal{P}$. Let $Y_1, Y_2, \dots, Y_n$ be the order statistics for $z(\boldsymbol{x}_1), \dots, z(\boldsymbol{x}_n)$. For all $p \in (0,1)$, then it holds that:*

$$Pr\left[|\bar{l} - l^*| \leq \frac{Y_2 - Y_1}{(1-p)^{-a} - 1}\right] \geq 1 - p \tag{3.7}$$

with $a \approx log(\nu)/log\left(\frac{Y_\nu - Y_3}{Y_3 - Y_2}\right)$, with ν any integer valued function of n such that $\nu(n) \to \infty$ and $\nu(n)/n \to 0$.

Importantly, Lemma 3.3, holds under the assumption that the samples $Y_1, \dots, Y_n$ follow a nondegenerate limit distribution function in the form $1 - exp(-x^a)$ for some $a > 0$ [58]. In particular, this is true for uniformly differentiable f, which is the case for most common neural networks, e.g., when the activation functions are Sigmoid, Tanh, etc. For ReLU-based networks, where the differentiability requirement is not satisfied, we can still apply Lemma 3.3 by splitting the non-linearities, i.e., splitting all the ReLU nodes, achieving a linear system differentiable everywhere. Concretely, whenever a ReLU's interval bound crosses 0 (i.e., $\underline{z} < 0 < \overline{z}$) computed over the perturbation region $\mathcal{P}$, we split the computation into two linear branches enforcing $z(\boldsymbol{x}) \geq 0$ (active) and $z(\boldsymbol{x}) \leq 0$ (inactive). On each resulting subregion, the network is affine, so the neuron's pre-activation is an affine (hence differentiable) map of $\boldsymbol{x}$. Sampling i.i.d. uniformly from $\mathcal{P}$ and rejecting points that do not satisfy

a branch's linear constraints is equivalent to sampling i.i.d. uniformly from that constrained subregion. Hence, the order statistics restricted to the subregion obey the required extreme-value limit. In practice, in our experimental setting described in Sec. 3.2.6, where $\mathcal{P}$ is an ℓ_∞-ball and ReLU constraint splitting is applied, the rejection rate remains very low. Specifically, the fraction of discarded samples is always negligible (typically below 5% on average). This is because the probability that a uniformly sampled point violates the constraints decreases rapidly as the sample size increases, and the rejection step only eliminates points outside the feasible region while preserving the independence of the remaining samples.

3.2.3 Extending Wilks' Probabilistic Guarantees

Building on the results of Lemma 3.3, we can extend the theoretical guarantees of `PT-LiRPA` to the entire perturbation region $\mathcal{P}$. Specifically, Lemma 3.3 provides a bound on the maximum error in estimating the true minimum (or maximum) of a given intermediate node z. By adding this EVT-based error bound to the initial estimates $\bar{l}$ and $\underline{u}$, obtained from the first random sampling of n inputs from $\mathcal{P}$, we derive a probabilistically tight approximation of the reachable set for z that holds for any $\boldsymbol{x} \in \mathcal{P}$. Notably, following the asymptotic requirements in [92], we can choose the number of upper order statistics as $\nu(n) = \lfloor n^\xi \rfloor$, with $\xi \in (0, 1)$, which clearly satisfies the conditions $\nu(n) \to \infty$ and $\nu(n)/n \to 0$.

Hence, for any intermediate neuron in the network, we have:

Theorem 3.4 (Improved `PT-LiRPA` probabilistic guarantee on the estimated reachable set): *Fix a positive integer n and real values $p, \xi \in (0, 1)$, and let $\nu = \lfloor n^\xi \rfloor$. For any neuron z in an N-layer DNN, let $z(\boldsymbol{x})$ denote the pre-activation value of z when $\boldsymbol{x}$ is the input to the network. Let $\bar{l} = \min_{k=1,\dots,n} z(\boldsymbol{x}_k)$ and $\underline{u} = \max_{k=1,\dots,n} z(\boldsymbol{x}_k)$ as the minimum and maximum pre-activation values observed in a sample of n random inputs $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$ independently and uniformly drawn from $\mathcal{P} \subseteq \mathbb{R}^d$. Let $Y_1 \le Y_2 \le \cdots \le Y_n$ be the order statistics for the observed values $z(\boldsymbol{x}_k)$.*

Then for any $\boldsymbol{x} \in \mathcal{P}$ it holds that:

$$Pr\left[\bar{l} - \frac{Y_2 - Y_1}{(1-p)^{-a_l} - 1} \le z(\boldsymbol{x}) \le \underline{u} + \frac{Y_n - Y_{n-1}}{(1-p)^{-a_u} - 1}\right] \ge 1 - 2p.$$

with $a_l \approx \frac{\log(\nu)}{\log\left(\frac{Y_\nu - Y_3}{Y_3 - Y_2}\right)}, a_u \approx \frac{\log(\nu)}{\log\left(\frac{Y_{n-2} - Y_{n-\nu}}{Y_{n-1} - Y_{n-2}}\right)}$.

Proof. The proof directly follows from the union bound, exploiting Lemma 3.3. □

Therefore, in a network with m neurons, using the LiRPA approach employing the estimates $\hat{l}$ and $\hat{u}$ for the reachable set of neuron z as given by Theorem 3.4, i.e.,

$$\hat{l} = \bar{l} - \frac{Y_2 - Y_1}{(1-p)^{-a_l} - 1} \qquad \hat{u} = \underline{u} + \frac{Y_n - Y_{n-1}}{(1-p)^{-a_u} - 1}, \tag{3.8}$$

we can compute a linear lower bound function $\boldsymbol{a}^T_{\texttt{PT-LiRPA}}(\boldsymbol{x}) + \boldsymbol{c}_{\texttt{PT-LiRPA}}$ such that with probability at least $1 - 2mp$ for any $\boldsymbol{x} \in \mathcal{P}$ satisfies:

$$f(\boldsymbol{x}) \geq \boldsymbol{a}^T_{\texttt{PT-LiRPA}}(\boldsymbol{x}) + \boldsymbol{c}_{\texttt{PT-LiRPA}}. \tag{3.9}$$

We apply the sampling procedure independently for each of the m neurons, estimating the reachable set using n i.i.d. samples drawn from $\mathcal{P}$. By Theorem 3.4, each reachable set is correct with probability at least $1 - 2p$. Then we use union bound to estimate the probability that all reachable sets are simultaneously correct, yielding a lower bound $1 - 2mp$. We note that this analysis does not need independence between estimation errors across neurons. In fact, dependencies may arise due to the layered nature of DNNs, which may lead to compounding errors in deeper layers.

Some observations are in order. The result of Theorem 3.4 provides for any choice of n and p a guarantee holding for all input values $x \in \mathcal{P}$. However, there is a potential weakness in its practical use since we would also like to guarantee that the two sides of inequality (3.9) are as close as possible, i.e., that the linear lower bound is as tight as possible. For this, we would like to have that for each neuron z the values $\hat{l}$ (respectively, $\hat{u}$) and $\bar{l}$ (respectively, $\underline{u}$) are close. However, the tail index parameters a_l and a_u, ruling their difference, depend on the shape of the tail of the distribution of $z(\boldsymbol{x})$ over $\mathcal{P}$. This precludes the possibility of computing the minimum value of n yielding to achieve a desired precision for a given desired confidence p.

One possibility to address this issue is to guess the minimum number of samples by a standard doubling technique: keep on doubling the number of samples used until the estimated tail corrections fall below a desired threshold. Alternatively, we can start with a conservative sample size n inspired by Wilks' formula and our extension (Proposition 3.4), set $\nu = \lfloor n^{\xi} \rfloor$ with $\xi \in (0, 1)$, and compute the resulting values $\hat{l}$, $\hat{u}$ as by Theorem 3.4. While potentially suboptimal, the experiments show that this approach produces linear bounds that, besides satisfying the above guarantees, remain significantly tighter than those provided by the traditional LiRPA-based approach, especially in deeper layers where LiRPA errors tend to compound.

3.2.4 Example of `PT-LiRPA` linear bounds computation.

Recalling the example provided in Sec. 2.3.4, we show now the computation of the linear bounds employing CROWN enhanced with `PT-LiRPA`. In detail, the calculation is analogous to what we have seen above, except for the construction of the diagonal matrices and bias vectors. In the following, we will compute the linear bounds using both the theoretical results of Theorem 3.2 and Theorem 3.4.

We start by computing the estimated reachable sets from a sample-based approach in $\mathcal{P}$ using $n = 10k$ samples, which for the Proposition 3.4, with $R = 0.999$ and considering the number of neurons in the DNN, are sufficient to have a final confidence $\psi \geq 0.99$. We report in Fig. 3.12 highlighted in red the estimated reachable sets obtained from the propagation of n random samples drawn from $[[-2, 2], [-1, 3]]$. As we can notice, the bounds are slightly tighter than the overestimated ones obtained from the IBP process. Our intuition is thus that from the computation of

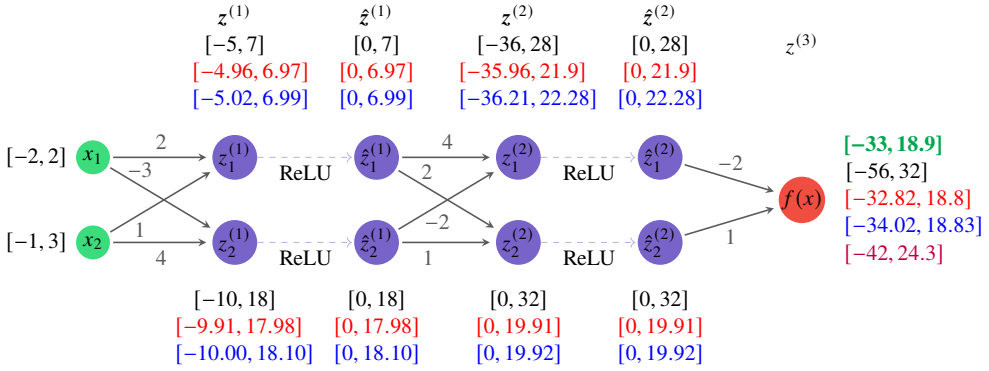


Fig. 3.12: Toy DNN used in this example. Intervals reported in green are the exact output reachable set computed via MIP, in black are the results of the IBP, and in purple the CROWN ones considering the input $[[-2, 2], [-1, 3]]$. In red are the reachable sets computed using a naive sampling-based approach of $n = 10k$ samples. Finally, in blue, the ones computed using a naive sampling-based approach combined with the EVT error estimation.

$\underline{\boldsymbol{D}}^{(i)}, \overline{\boldsymbol{D}}^{(i)}, \underline{\boldsymbol{b}}^{(i)}, \overline{\boldsymbol{b}}^{(i)}$ using these tightened bounds we can obtain more accurate lower and upper final linear bounds. For the diagonal matrices and bias vectors, we get:

$$\underline{\boldsymbol{D}}^{(2)} = \begin{bmatrix} \frac{u}{u-l} & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.3784 & 0 \\ 0 & 1 \end{bmatrix} \qquad \underline{\boldsymbol{b}}^{(2)} = \begin{bmatrix} \frac{-ul}{u-l} \\ 0 \end{bmatrix} = \begin{bmatrix} 13.61 \\ 0 \end{bmatrix}$$

$$\overline{\boldsymbol{D}}^{(2)} = \begin{bmatrix} \alpha & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \qquad \overline{\boldsymbol{b}}^{(2)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

and

$$\underline{\boldsymbol{D}}^{(1)} = \begin{bmatrix} \frac{u}{u-l} & 0 \\ 0 & \alpha \end{bmatrix} = \begin{bmatrix} 0.5839 & 0 \\ 0 & 0 \end{bmatrix} \qquad \underline{\boldsymbol{b}}^{(1)} = \begin{bmatrix} \frac{-ul}{u-l} \\ 0 \end{bmatrix} = \begin{bmatrix} 2.8986 \\ 0 \end{bmatrix}$$

$$\overline{\boldsymbol{D}}^{(1)} = \begin{bmatrix} \frac{u}{u-l} & 0 \\ 0 & \frac{u}{u-l} \end{bmatrix} = \begin{bmatrix} 0.5839 & 0 \\ 0 & 0.6448 \end{bmatrix} \qquad \overline{\boldsymbol{b}}^{(1)} = \begin{bmatrix} \frac{-ul}{u-l} \\ \frac{-ul}{u-l} \end{bmatrix} = \begin{bmatrix} 2.8986 \\ 6.3870 \end{bmatrix}.$$

Thus computing all the $\boldsymbol{A}$s and ds vectors,

$$\underline{\boldsymbol{A}}^{(2)} = \underline{\boldsymbol{A}}^{(3)}\underline{\boldsymbol{D}}^{(2)}\boldsymbol{W}^{(2)} = [-1.0275, 2.5138], \qquad \overline{\boldsymbol{A}}^{(2)} = \overline{\boldsymbol{A}}^{(3)}\overline{\boldsymbol{D}}^{(2)}\boldsymbol{W}^{(2)} = [2, 1],$$

$$\underline{\boldsymbol{A}}^{(1)} = \underline{\boldsymbol{A}}^{(2)}\underline{\boldsymbol{D}}^{(1)}\boldsymbol{W}^{(1)} = [-1.2, -0.6], \qquad \overline{\boldsymbol{A}}^{(1)} = \overline{\boldsymbol{A}}^{(2)}\overline{\boldsymbol{D}}^{(1)}\boldsymbol{W}^{(1)} = [0.4013, 3.7471],$$

$$\underline{d} = \underline{\boldsymbol{A}}^{(3)}\underline{\boldsymbol{b}}^{(2)} + \underline{\boldsymbol{A}}^{(2)}\underline{\boldsymbol{b}}^{(1)} = -30.1983, \qquad \overline{d} = \overline{\boldsymbol{A}}^{(3)}\overline{\boldsymbol{b}}^{(2)} + \overline{\boldsymbol{A}}^{(2)}\overline{\boldsymbol{b}}^{(1)} = 12.1841,$$

we obtain:

$$\underline{f}_{\text{CROWN w/ PT-LiRPA}} = \min_{\boldsymbol{x}\in\mathcal{P}} \underline{\boldsymbol{A}}^{(1)}(\boldsymbol{x}) + \underline{d} = -||\underline{\boldsymbol{A}}^{(1)}||_1 \cdot \varepsilon + \underline{\boldsymbol{A}}^{(1)}\boldsymbol{x}_0 + \underline{d}$$
$$= -3.6 - 0.6 - 30.1983 = -34.4.$$

$$\overline{f}_{\text{CROWN w/ PT-LiRPA}} = \max_{\boldsymbol{x}\in\mathcal{P}} \overline{\boldsymbol{A}}^{(1)}(\boldsymbol{x}) + \overline{d} = ||\underline{\boldsymbol{A}}^{(1)}||_1 \cdot \varepsilon + \overline{\boldsymbol{A}}^{(1)}\boldsymbol{x}_0 + \overline{d}$$
$$= 8.2968 + 3.7471 + 12.1841 = 24.23.$$

As we can notice, these bounds are significantly tighter than the ones computed using CROWN ($[-42, 24.3]$). However, the theoretical guarantees provided by Theorem 3.2 allow us to state that these bounds are probabilistically sound only for a fraction R of the perturbation region $\mathcal{P}$, thus resulting in a slightly weaker guarantee w.r.t. the one provided by existing probabilistic approaches. Nonetheless, we believe that if one accepts the assumption underlying this theoretical guarantee, this approach still presents a valuable and computationally efficient tool for computing probabilistically valid linear output bounds. In the following, we show how to practically extend these theoretical guarantees to the whole perturbation region, exploiting Theorem 3.4.

We start again from computing the estimated reachable sets from a sample-based approach in $\mathcal{P}$. For each estimated lower and upper bound, we compute and add the corresponding error using $\frac{Y_2 - Y_1}{(1-p)^{-a_l}-1}$ for the lower and $\frac{Y_n - Y_{n-1}}{(1-p)^{-a_u}-1}$ for the upper bound, respectively. We report in Figure 3.12 highlighted in blue the new estimated reachable sets obtained from the propagation of n random samples drawn from $[[-2, 2], [-1, 3]]$ with the addition of the corresponding error. Hence, we speculate that recomputing $\underline{\boldsymbol{D}}^{(i)}, \overline{\boldsymbol{D}}^{(i)}, \underline{\boldsymbol{b}}^{(i)}, \overline{\boldsymbol{b}}^{(i)}$ using these new estimated reachable sets we can still obtain more accurate lower and upper final linear bounds w.r.t. the LiRPA-based approaches. In fact, we obtain:

$$\underline{\boldsymbol{D}}^{(2)} = \begin{bmatrix} \frac{u}{u-l} & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.3809 & 0 \\ 0 & 1 \end{bmatrix} \qquad \underline{\boldsymbol{b}}^{(2)} = \begin{bmatrix} \frac{-ul}{u-l} \\ 0 \end{bmatrix} = \begin{bmatrix} 13.7919 \\ 0 \end{bmatrix}$$
$$\overline{\boldsymbol{D}}^{(2)} = \begin{bmatrix} \alpha & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \qquad \overline{\boldsymbol{b}}^{(2)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

and

$$\underline{\boldsymbol{D}}^{(1)} = \begin{bmatrix} \frac{u}{u-l} & 0 \\ 0 & \alpha \end{bmatrix} = \begin{bmatrix} 0.5820 & 0 \\ 0 & 0 \end{bmatrix} \qquad \underline{\boldsymbol{b}}^{(1)} = \begin{bmatrix} \frac{-ul}{u-l} \\ 0 \end{bmatrix} = \begin{bmatrix} 2.9219 \\ 0 \end{bmatrix}$$
$$\overline{\boldsymbol{D}}^{(1)} = \begin{bmatrix} \frac{u}{u-l} & 0 \\ 0 & \frac{u}{u-l} \end{bmatrix} = \begin{bmatrix} 0.5820 & 0 \\ 0 & 0.6441 \end{bmatrix} \qquad \overline{\boldsymbol{b}}^{(1)} = \begin{bmatrix} \frac{-ul}{u-l} \\ \frac{-ul}{u-l} \end{bmatrix} = \begin{bmatrix} 2.9219 \\ 6.4427 \end{bmatrix}.$$

We can now compute all the $\boldsymbol{A}$s and ds vectors.

$$\begin{aligned}
\underline{\boldsymbol{A}}^{(2)} &= \underline{\boldsymbol{A}}^{(3)}\underline{\boldsymbol{D}}^{(2)}\boldsymbol{W}^{(2)} = [-1.0471, 2.5235] \\
\overline{\boldsymbol{A}}^{(2)} &= \overline{\boldsymbol{A}}^{(3)}\overline{\boldsymbol{D}}^{(2)}\boldsymbol{W}^{(2)} = [2, 1] \\
\underline{\boldsymbol{A}}^{(1)} &= \underline{\boldsymbol{A}}^{(2)}\underline{\boldsymbol{D}}^{(1)}\boldsymbol{W}^{(1)} = [-1.2187, -0.6094] \\
\overline{\boldsymbol{A}}^{(1)} &= \overline{\boldsymbol{A}}^{(2)}\overline{\boldsymbol{D}}^{(1)}\boldsymbol{W}^{(1)} = [0.3957, 3.7402] \\
\underline{d} &= \underline{\boldsymbol{A}}^{(3)}\underline{\boldsymbol{b}}^{(2)} + \underline{\boldsymbol{A}}^{(2)}\underline{\boldsymbol{b}}^{(1)} = -30.6434 \\
\overline{d} &= \overline{\boldsymbol{A}}^{(3)}\overline{\boldsymbol{b}}^{(2)} + \overline{\boldsymbol{A}}^{(2)}\overline{\boldsymbol{b}}^{(1)} = 12.2865
\end{aligned}$$

Finally we have

$$\begin{aligned}
\underline{f}_{\text{PT-LiRPA}} = \min_{\boldsymbol{x}\in\mathcal{P}} \underline{\boldsymbol{A}}^{(1)}(\boldsymbol{x}) + \underline{d} &= -||\underline{\boldsymbol{A}}^{(1)}||_1 \cdot \varepsilon + \underline{\boldsymbol{A}}^{(1)}\boldsymbol{x}_0 + \underline{d} \\
&= -3.6562 - 0.6094 - 30.6434 = -34.91.
\end{aligned}$$

$$\begin{aligned}
\overline{f}_{\text{PT-LiRPA}} = \max_{\boldsymbol{x}\in\mathcal{P}} \overline{\boldsymbol{A}}^{(1)}(\boldsymbol{x}) + \overline{d} &= ||\underline{\boldsymbol{A}}^{(1)}||_1 \cdot \varepsilon + \overline{\boldsymbol{A}}^{(1)}\boldsymbol{x}_0 + \overline{d} \\
&= 8.2717 + 3.7402 + 12.2865 = 24.3.
\end{aligned}$$

Although the upper bound is equivalent to the original CROWN approach, we can notice that our procedure produces a tighter lower bound. This toy example provides a preliminary insight into the potential of the proposed solution. Our speculation on the impact of `PT-LiRPA` on realistic verification instances will be confirmed by the experiments presented in Sec. 3.2.6.

EVT-based approach to directly bound the output?

A natural question that arises is whether the results of Theorem 3.4 can be used directly to obtain a tight estimation of the output reachable set, without relying on the LiRPA combination. Although this sampling-based method offers a probabilistic estimate that, with high confidence, contains the entire perturbation region, it may still underestimate the true output bounds due to its reliance on a finite number of samples. For example, the MIP result yields bounds of $[-33, 18.9]$, and as we can notice in Fig. 3.12 highlighted in blues, the output reachable set only applying a forward computation of Theorem 3.4 still underestimates the exact upper bound $[-34.02, 18.83]$. In contrast, since our method integrates a sampling-based approach with any LiRPA method—which inherently provides sound overestimations—the final computed bounds will always be at least as tight as those obtained through estimation based on a finite number of samples and are likely (with a confidence at least $1 - 2mp$) to produce valid linear bounds, i.e., not discarded by potential adversarial attacks (in fact, we obtain as final result $[-34.4, 24.23]$). Additionally, as emphasized in prior work [290], combining forward and backward analysis typically yields tighter bounds compared to using a simple forward bound computation. This observation further motivated our investigation into how tighter reachable sets can enhance the linearization approaches for verification efficiency.

3.2.5 PT-LiRPA framework for Neural Network Verification

Based on the theoretical results of Sec. 3.2.1, we now present in Algorithm 3 the PT-LiRPA approach for the verification process. For the sake of clarity and without loss of generality, we present the procedure applied to the parallel BaB as shown for the optimized LiRPA approach proposed in [291].

Algorithm 3: PT-LiRPA on parallel BaB

Input: A DNN f with N layers and m neurons, an original input $\boldsymbol{x}_0$, a maximum ε perturbation to create a perturbation region $\mathcal{P}$, maximum error in the confidence p, sample size n, $\xi \in (0,1)$ for $v(n)$ and a batch size t.
Output: *robust/not-robust*
▷ as in Alg. 4
if PGD_attack$(f, \mathcal{P})$ **then**
  return *not robust*
interm_bounds $\leftarrow$ get_interm_bounds$(f, \mathcal{P}, n, p, \xi)$ ▷ as in Alg. 5
$\underline{f}_{\mathcal{P}}, \overline{f}_{\mathcal{P}} \leftarrow$ LiRPA$(f, \mathcal{P}$, *interm_bounds*$)$ ▷ as in Alg. 1 where $\mathcal{P}$ contains $\boldsymbol{x}_0, \varepsilon$
$\mathcal{B} \leftarrow (\underline{f}_{\mathcal{P}}, \overline{f}_{\mathcal{P}})$
while $\mathcal{B} \neq \emptyset$ **do**
  $\mathcal{P}_1, \ldots, \mathcal{P}_t \leftarrow$ split$(\mathcal{B}, t)$
  interm_bounds$_{\mathcal{P}_1,\ldots,\mathcal{P}_t} \leftarrow$ get_interm_bounds$(f, [\mathcal{P}_1, \ldots, \mathcal{P}_t], n, p, \xi)$
  $(\underline{f}_{\mathcal{P}_1}, \overline{f}_{\mathcal{P}_1}), \ldots, (\underline{f}_{\mathcal{P}_t}, \overline{f}_{\mathcal{P}_t}) \leftarrow$ LiRPA$(f, [\mathcal{P}_1, \ldots, \mathcal{P}_t]$,*interm_bounds*$_{\mathcal{P}_1,\ldots,\mathcal{P}_t})$ ▷ parallel exec. of Alg. 1 on $\mathcal{P}_{1,\ldots,t}$
  $\mathcal{B}' \leftarrow (\underline{f}_{\mathcal{P}_1}, \overline{f}_{\mathcal{P}_1}), \ldots, (\underline{f}_{\mathcal{P}_t}, \overline{f}_{\mathcal{P}_t})$
  if $\exists\, \mathcal{P}_i \in \mathcal{B}'$ *s.t.* $\overline{f}_{\mathcal{P}_i} < 0$ or PGD_attack(f, C_i) **then**
    return *not robust*
  $\mathcal{B} \leftarrow \mathcal{B}' \setminus$ get_robust_domains$(\mathcal{B}')$
return *robust*

Given a DNN f with m neurons and a region of interest $\mathcal{P}$, the verification process typically involves a projected gradient descent (PGD) attack [158]. This attack, reported in Alg. 4 for the sake of completeness, can be employed before, after, or during the BaB procedure to search for potential adversarial inputs within the input region under consideration. In detail, we report in our PT-LiRPA algorithm (Alg. 3), a potential employment of PGD during the verification process. Specifically, the attack is performed before and during the BaB, performing a projected gradient descent search in the L_∞ ball $\mathcal{P} = \{\boldsymbol{x} : \|\boldsymbol{x} - \boldsymbol{x}_0\|_\infty \leq \varepsilon\}$ to find an adversarial input $\boldsymbol{x}_{\text{adv}}$ that makes the scalar model output non-positive, i.e., $f(\boldsymbol{x}_{\text{adv}}) \leq 0$. Starting from either the clean original input or a random uniform perturbation in $[-\varepsilon, \varepsilon]$ (i.e., a random input vector in the $\mathcal{P}$) the method iteratively evaluates the scalar output, computes its gradient with respect to the input, and takes an L_∞-constrained descent step using the elementwise sign of the gradient. After each update, the perturbation is projected back onto the L_∞ ball and the input is clipped to the valid data range; the procedure stops early if a negative output is obtained and otherwise runs for at most T iterations.

Algorithm 4: `PGD_Attack`[158]

Input Original input $\boldsymbol{x}_0$, neural network f, maximum perturbation ε to create data range $[x_{\min}, x_{\max}]$, step size α, iterations T, *random_start*
Output adversarial example $\boldsymbol{x}_{\text{adv}}$ with $\|\boldsymbol{x}_{\text{adv}} - \boldsymbol{x}_0\|_\infty \leq \varepsilon$ or original input $\boldsymbol{x}_0$

if *random_start* **then**
  $\boldsymbol{x} \leftarrow \boldsymbol{x}_0 + \text{Uniform}(-\varepsilon, \varepsilon)$
else
  $\boldsymbol{x} \leftarrow \boldsymbol{x}_0$
$\boldsymbol{x}_{\text{adv}} \leftarrow \text{clip}(\boldsymbol{x}, x_{\min}, x_{\max})$
for $t \in \{1, \ldots, T\}$ **do**
  if $f(\boldsymbol{x}_{\text{adv}}) \leq 0$ **then**
    return $\boldsymbol{x}_{\text{adv}}$
  $g \leftarrow \nabla_{\boldsymbol{x}_{\text{adv}}} f(\boldsymbol{x}_{\text{adv}})$ ▷ gradient of scalar output w.r.t. adversarial input
  $\boldsymbol{x}_{\text{adv}} \leftarrow \boldsymbol{x}_{\text{adv}} - \alpha \cdot \text{sign}(g)$ ▷ descent step for minimizing s
  $\Delta \leftarrow \text{clip}(\boldsymbol{x}_{\text{adv}} - \boldsymbol{x}_0, -\varepsilon, \varepsilon)$ ▷ project perturbation onto L_∞ ball
  $\boldsymbol{x}_{\text{adv}} \leftarrow \text{clip}(\boldsymbol{x}_0 + \Delta, x_{\min}, x_{\max})$
return $\boldsymbol{x}_0$

The main hyperparameters are the maximum perturbation ε, the step size α (typically chosen on the order of ε/T), the maximum iterations T, and the optional random start; multiple restarts or momentum can be used to increase attack strength. Success provides a concrete counterexample to robustness within the prescribed L_∞ radius, while failure is only a heuristic indication and does not constitute a formal certificate of robustness.

Hence, Alg. 3 begins with a PGD attack (lines 3-4), and if no adversarial is found, we proceed with the Branch-and-Bound process. We compute the estimated reachable sets using the `get_interm_bounds` method (line 5), which exploits the results of Theorem 3.4 and is reported here below in Alg. 5 for clarity.

Algorithm 5: `get_interm_bounds`

interm_bounds $\leftarrow \{\}$
$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \leftarrow \text{UniformSampling}(\mathcal{P}, n)$ ▷ collect n random i.i.d inputs from $\mathcal{P}$
for each intermediate layer **do**
  $\hat{\boldsymbol{l}}, \hat{\boldsymbol{u}} \leftarrow \{\}$
  for each node z in layer nodes **do**
    $Y_1, \ldots, Y_n \leftarrow \text{Sort}(z(\boldsymbol{x}_1), \ldots, z(\boldsymbol{x}_n))$ ▷ with $z(\cdot)$ as in Theorem 3.4
    $\overline{l}, \underline{u} \leftarrow Y_1, Y_n$
    $a_l, a_u \leftarrow \frac{\log(\nu)}{\log\left(\frac{Y_\nu - Y_3}{Y_3 - Y_2}\right)}, \frac{\log(\nu)}{\log\left(\frac{Y_{n-2} - Y_{n-\nu}}{Y_{n-1} - Y_{n-2}}\right)}$
    $\hat{l}, \hat{u} \leftarrow \overline{l} - \frac{Y_2 - Y_1}{(1-p)^{-a_l} - 1}, \underline{u} + \frac{Y_n - Y_{n-1}}{(1-p)^{-a_u} - 1}$ ▷ as in Eq. 3.8
    $\hat{\boldsymbol{l}}, \hat{\boldsymbol{u}} \leftarrow \hat{\boldsymbol{l}} \cup \hat{l}, \hat{\boldsymbol{u}} \cup \hat{u}$
  interm_bounds $\leftarrow$ *interm_bounds* $\cup [\hat{\boldsymbol{l}}, \hat{\boldsymbol{u}}]$ ▷ store the vector of lower and upper bounds for the specific layer
return *interm_bounds*

We then use these bounds in the linear bounds computation on any existing `LiRPA` approach (line 6), following Alg. 1 of Sec. 2.3.4 and the computation shown in the toy example of Sec. 3.2.4. We store the resulting output bounds $\underline{f}$ and $\overline{f}$ for the region $\mathcal{P}$, namely $\underline{f}_{\mathcal{P}}$ and $\overline{f}_{\mathcal{P}}$ in a set $\mathcal{B}$ of unverified regions (lines 7). We then continue the BaB process by splitting (using the `split` method) the original region from $\mathcal{B}$ into t sub-regions (line 9). Notably, we can perform the parallel selection and splitting into sub-domains using information on unstable ReLU nodes, as shown in [37, 275], or just on the perturbation region $\mathcal{P}_i$ [274]. Once we have the new sub-domains, we recompute the estimated reachable sets in parallel and use these bounds for the new computation of the linear lower and upper bounds for each sub-region, and we update $\mathcal{B}$ with the resulting unverified sub-domains (lines 10-12).

Algorithm 6: `get_robust_domains`

```
1: robust_domains ← {}
2: for (f_{P_i}, f̄_{P_i}) ∈ B do
3:     if f_{P_i} > 0 then
4:         robust_domains ←robust_domains ∪ P_i          ▷ following Def. 2.3
5: return robust_domains
```

At each iteration, the process can end either because there is at least a single sub-domain $\mathcal{P}_i \in \mathcal{B}$ that presents $\overline{f} < 0$, or a PGD attack succeeds, thus returning *not robust* as the answer (lines 13-14). Otherwise, the process continues updating $\mathcal{B}$ with the unverified domains using the procedure `get_robust_domains` (line 15) reported in Alg. 6. Following Theorem 3.4, if we reach the emptiness of $\mathcal{B}$, thus no adversarial examples are found during the verification process and all the sub-domains are evaluated as *robust*, we can state that, with a confidence $\geq 1-2mp$, the DNN is robust for the whole perturbation region $\mathcal{P}$.

3.2.6 Empirical Evaluation

Our empirical evaluation consists of three main experiments to answer the following questions:

Q1. *How does the hyperparameter ξ impact the lower bound computation? How does the number of samples employed in the computation process impact the tightening process?*

Q2. *How much `PT-LiRPA` improves the robustness bounds certificates w.r.t. other probabilistic and worst-case methods? What is the computational overhead of the proposed solution with respect to a worst-case certification approach?*

Q3. *What is the general impact of `PT-LiRPA` in the verification process of challenging instances such as the one employed in the VNN-COMP [198, 35]?*

All data are collected on a cluster running Rocky Linux 9.34 equipped with Nvidia RTX A6000 (48 GiB) and a CPU AMD Epyc 7313 (16 cores).

Answers to Q1.

To address the first question, we consider the original pre-trained models on MNIST dataset. Specifically, we focus on MLP models with varying depths and activation functions. For consistency and ease of comparison, we adopt the same notation as [297] and [284]: a model is denoted by the dataset name, followed by the number of layers i, the number of neurons per layer j, formatted as $i \times [j]$, and the activation function used. We begin by analyzing the mean percentage error in estimating the lower bounds of the intermediate reachable sets (using Eq. 3.10) for a fixed input image, using `PT-LiRPA` on various neural networks trained on the MNIST dataset.

$$error = \frac{Y_2 - Y_1}{(1-p)^{-a} - 1} \tag{3.10}$$

with $a \approx \frac{\log(\nu)}{\log\left(\frac{Y_\nu - Y_3}{Y_3 - Y_2}\right)}$, where $\nu = \lfloor n^\xi \rfloor$, $\xi \in (0,1)$.

In particular, we first study the impact of the $\xi \in (0,1)$ hyperparameter, which controls the number of order statistics $\nu = \lfloor n^\xi \rfloor$ used to estimate the tail distribution and compute the mean distance error across all intermediate nodes, and thus the neural network. Our results reported in Fig. 3.13 show that, using a fixed sample size of $n = 10k$, for small values of ξ, the number of extreme samples is limited, leading to high variance and bias in the tail modeling, and consequently to considerable estimation errors. As ξ increases in the interval $(0.2, 0.6)$, the error decreases significantly due to the improved reliability of the extreme value statistics. For larger ξ values, around $[0.6, 1)$, the error stabilizes below the overall mean error across all different ξ values, indicating that the estimator becomes robust and further improvements are marginal. We highlight that, even without access to the true lower (or upper, respectively) bound, one can, in principle, fine-tune the ξ parameter as desired to minimize the error with respect to the target value for a given application. Based on these observations, we set $\xi = 0.85$ for the following experiments, as it provides a good trade-off between low error estimation and stability across different models.

Hence, for the fixed value ξ and perturbed input, we investigate the impact of three sample sizes, namely 10000, 100000, and 350000, in the lower bound estimation of each estimated reachable set. For each sample size, we compute the mean distance

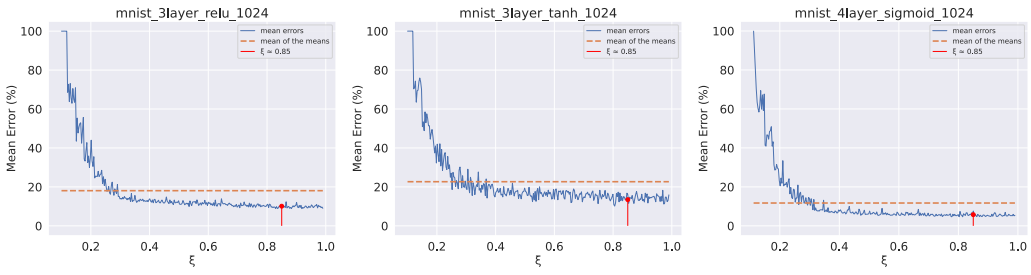


Fig. 3.13: Mean distance error (%) achieved on each intermediate node using PT-LiRPA with EVT-based error computation, for networks with ReLU (left), Tanh (middle), and Sigmoid (right) activations. In red we report the value $\xi = 0.85$ selected for the following experiments.

error across the entire network setting $p = 0.01$, thus ensuring a confidence level of at least 99% in the final results. For the distance error achieved on each intermediate node of the network using Eq. 3.10 with $\xi = 0.85$. We then compute the percentage error relative to the smallest observed value in the sample. Specifically, for each neuron, the error is normalized by the absolute value of the smallest pre-activation observed value.

Name model	# samples	mean distance error (%) 1-p=0.99	# samples	mean distance error (%) 1-p=0.99	# samples	mean distance error (%) 1-p=0.99
MNIST 2×[1024], ReLU	10000	12.91	100000	9.35	350000	8.78
MNIST 3×[1024], ReLU	10000	9.63	100000	8.49	350000	7.09
MNIST 4×[1024], ReLU	10000	11.89	100000	9.34	350000	8.85
MNIST 2×[1024], Tanh	10000	11.31	100000	12.22	350000	10.49
MNIST 3×[1024], Tanh	10000	10.67	100000	11.01	350000	10.34
MNIST 4×[1024], Sigmoid	10000	5.03	100000	6.21	350000	4.36
	mean error	10.24%	**mean error**	9.44%	**mean error**	8.32%

Table 3.4: Mean maximum error in estimating the lower bound of the intermediate reachable set for a fixed input image, using `PT-LiRPA` on various neural networks trained on the MNIST dataset with different sample sizes.

The results of Tab 3.4 demonstrate that with high confidence (i.e., at least 99%) across all tested networks, increasing the sample size, the percentage error in the estimation decreases. Importantly, even with a limited sample size (e.g., 10000 samples), we note that the mean error across all the networks in the lower bound estimation is 10.24% from the estimated one, while increasing the sample size, we reach a maximum error of 8.32%. To assess the practical impact of the estimation error on the final output bounds, we consider the same perturbed input of the previous experiments and fix the sample size to $350k$.

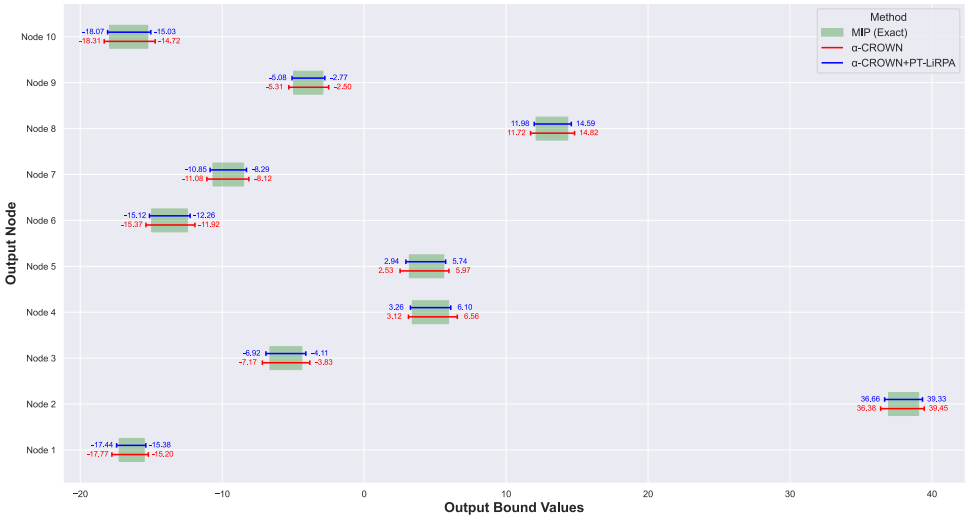


Fig. 3.14: Comparison of output bounds on the *MNIST_2x[1024]_ReLU* network using α-CROWN [291] reported in red,α-CROWN [291] with `PT-LiRPA` in blue, and exact MIP verification [252] in green.

We then compare the final output bounds obtained by an exact MIP-based verification [252], the state-of-the-art α-CROWN [291] method, and α-CROWN enhanced with our `PT-LiRPA`. As a representative case, we select the *MNIST_2x[1024]_ReLU* network. The reason for selecting this network is to enable a comparison with MIP [252], which provides exact output bounds. Since MIP solvers are inherently designed for ReLU (and more generally, piecewise-linear) activations, they are not directly applicable to networks with Sigmoid or Tanh activations. Among the models listed in Table 3.4, the ones with larger estimation errors under ReLU activations with confidence $1-p=0.99$ are *MNIST_2x[1024]_ReLU* and *MNIST_4x[1024]_ReLU*. We select the former because MIP scalability becomes a limiting factor on larger architectures, making *MNIST_2x[1024]_ReLU* the most suitable candidate for this analysis. The results, reported in Fig. 3.14, show that despite the estimation error, `PT-LiRPA` produces tighter output bounds than α-CROWN, while soundly overapproximating the exact MIP bounds. Similar trends were consistently observed across all evaluated networks, which we do not report here for readability reasons.

We also perform an additional experiment to analyze the impact of the confidence level on the tightness of the bounds. Specifically, we consider the same model of Fig. 3.14, i.e., *MNIST_2x[1024]_ReLU*, where we have the possibility of employing the exact MIP solver and evaluated a range of increasing confidence levels, namely $1-p \in \{0.8, 0.9, 0.95, 0.99, 0.995, 0.996, 0.997, 0.998, 0.999\}$. For each confidence level, we compute the bounds using α-CROWN, α-CROWN enhanced with our `PT-LiRPA`, and MIP (which provides the exact bounds). To measure the tightness, we calculate for each of the 10 output nodes the distance between the two bounds computed. For example, if MIP returns output bounds $[-2.2, 3.5]$ and `PT-LiRPA` returns $[-2.4, 3.7]$, the distance is given by the sum of the absolute differences between the lower and upper bounds, i.e., $|-2.2+2.4|+|3.7-3.5|=0.4$. This procedure is repeated for all 10 nodes, collecting, for each confidence level, the mean and standard deviation of these distances.

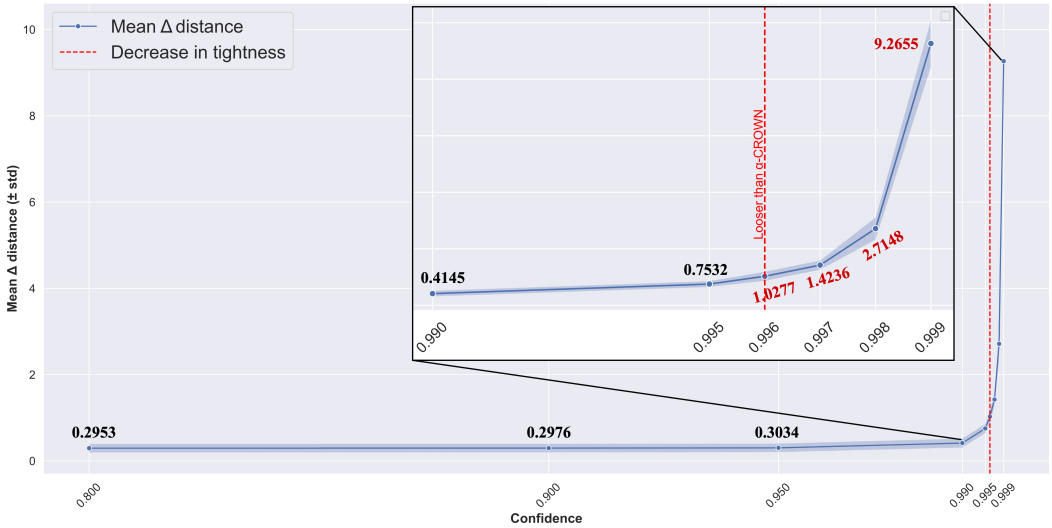


Fig. 3.15: Mean tightness level of `PT-LiRPA` for increasing confidence level on *MNIST_2x[1024]_ReLU* model.

The results in Fig. 3.15 clearly show that for moderate confidence levels, i.e., $1 - p \in [0.8, 0.995]$, `PT-LiRPA` consistently produces sound and tighter bounds compared to α-CROWN. As the confidence level approaches 1 (i.e., as $p \to 0$), the distance between `PT-LiRPA` and the exact bounds increases, leading to looser bounds than α-CROWN. This trend is perfectly in line with the probabilistic nature of our approach. In fact, considering, for instance, the formula used to compute the probabilistic lower bound, $\hat{l} = Y_1 - \frac{Y_2 - Y_1}{(1-p)^{-a_l} - 1}$ where $a_l \approx \frac{\log(\nu)}{\log\left(\frac{Y_\nu - Y_3}{Y_3 - Y_2}\right)}$ ($\nu = \lfloor n^\xi \rfloor = 350k^{0.85} \simeq 50k$), we observe that as $p \to 0$ (corresponding to requiring very high confidence), the denominator $(1-p)^{-a_l} - 1$ tends to 0 faster than the numerator $(Y_2 - Y_1)$. As a result, the error term $\frac{Y_2 - Y_1}{(1-p)^{-a_l} - 1}$ becomes large, and a correspondingly larger margin must be subtracted from the observed minimum. Consequently, the bounds become looser as the confidence level approaches 1. This behavior is mathematically unavoidable since we cannot guarantee arbitrarily high confidence levels while simultaneously maintaining very tight bounds. The plot therefore not only confirms the soundness of our method but also highlights the expected trade-off between confidence and tightness, which is especially relevant for safety-critical applications. For the sake of completeness, we also tested additional models such as *MNIST_2x[1024]_Tanh* and *MNIST_4x[1024]_Sigmoid*. Although MIP was not applicable in these cases, we compared `PT-LiRPA` only against α-CROWN and observed a similar trend, i.e., a loss of tightness above confidence 0.995, further confirming the generality of our findings.

Answers to Q2.

For the second question, we consider the models trained on MNIST and CIFAR datasets, as provided in [297]. Hence, we evaluate the performance of our `PT-LiRPA`-based probabilistic verifier, alongside PROVEN [284], Randomized Smoothing [49], and CROWN [297], by testing for each model 10 random images from the corresponding test datasets. Specifically, we compare the maximum input perturbation ε that can be certified for each method. CROWN [297] serves as the baseline for worst-case robustness certification, as employed in [284]. Based on the previous results, we set a sample size of $n = 350k$, $\xi = 0.85$ and $1 - 2mp \geq 0.99$ in our `PT-LiRPA`-based verifier. This confidence level is consistent with the settings used by [284] in their experiments.

Certification method	Worst-case (CROWN)	PROVEN†	Rand. Smoothing	CROWN w/ `PT-LiRPA`	`PT-LiRPA` certification bound increase
Confidence	100%	≥ 99%	≥ 99%	≥ 99%	w.r.t. CROWN, PROVEN, Rand. Smoothing
MNIST 2×[1024], ReLU	0.03566±0.011	0.0556	0.0461±0.0106	**0.0558±0.02068**	**1.6X**, **1.004X**, **1.21X**
MNIST 3×[1024], ReLU	0.03112 ± 0.01076	0.03524	0.03452±0.0169	**0.06652 ±0.02501**	**2.14X**, **1.9X**, **1.93X**
MNIST 2×[1024], Tanh	0.01827 ±0.01331	0.02915	0.02301±0.0115	**0.02949 ± 0.02515**	**1.61X**, **1.01X**, **1.28X**
MNIST 3×[1024], Tanh	0.01244 ±0.00468	0.01360	0.01294±0.0073	**0.0257 ± 0.01205**	**2.07X**, **1.89X**, **1.99X**
MNIST 4×[1024], Sigmoid	0.01975 ±0.0111	0.02170	0.02439±0.019	**0.05506±0.04035**	**2.79X**, **2.54X**, **2.26X**
CIFAR 5×[2048], ReLU	0.002412 ± 0.00184	0.00264	0.00778±0.0104	**0.00874 ± 0.00107**	**3.62X**, **3.31X**, **1.12X**
CIFAR 7×[1024], ReLU	0.001984 ± 0.00089	0.00209	**0.006264±0.0061**	0.00471 ± 0.00273	**2.37X**, **2.25X**, 0X

Table 3.5: Comparison of `PT-LiRPA` with worst-case bound CROWN [297] and probabilistic approaches PROVEN [284], Randomized Smoothing [49] on different neural networks MNIST and CIFAR models. † results taken from the original paper [284] due to the code's unavailability to reproduce the results.

Tab.3.5 reports the results obtained. The *"Worst-case"* column indicates the mean and standard deviation of the maximum ε perturbation tolerated and provably certified using CROWN [297] for that model under consideration in the 10 random images tested, as in [284]. Instead, the PROVEN, Rand. Smoothing and `PT-LiRPA` columns report the certified mean ε with standard deviation we achieve for the same images, using the three probabilistic approaches, sacrificing only a 10^{-2} of confidence. In general, we observe that the `PT-LiRPA`-based verifier can certify robustness levels up to 3.62 times higher than the worst-case baseline CROWN [297] and up to 3.31 and 2.26 times higher compared to PROVEN [284] and Rand. Smoothing [49], respectively. This demonstrates the significant advantage of our approach over other existing probabilistic methods.

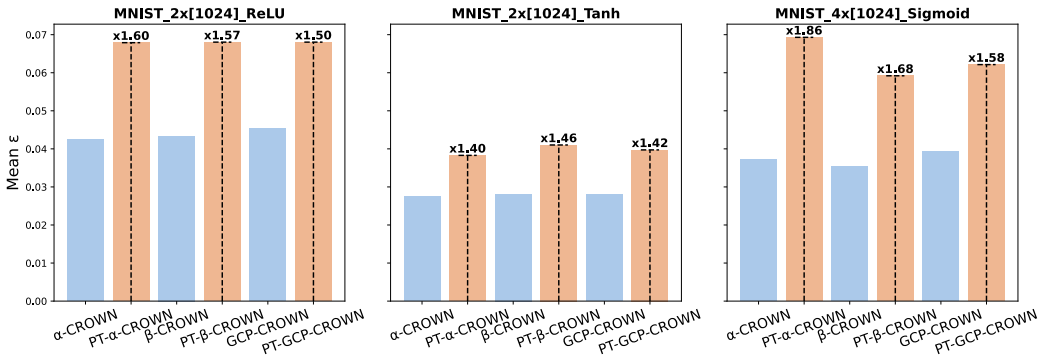


Fig. 3.16: Comparison `PT-LiRPA` with worst-case bound CROWN [297], α-CROWN [291], β-CROWN [275], GCP-CROWN [298] on MNIST_2×[1024]_ReLU, MNIST_2×[1024]_Tanh, and MNIST_4×[1024]_Sigmoid models. On the x-axis, we report the original worst-case method and the corresponding probabilistic version using our `PT-LiRPA` framework. On the y-axis, we report, for each method, the mean maximum input perturbation ε that can be certified on 10 random images.

Since the worst-case setting employed in this comparison is one of the first LiRPA-based approaches proposed in the literature, in the following, we conduct further analysis on the level of tightness we can achieve with respect to more recent LiRPA approaches such as α-CROWN [291], β-CROWN [275], and GCP-CROWN [298]. For each of these approaches, as well as their probabilistic counterparts based on our `PT-LiRPA` framework, we compute the mean input perturbation ε that can be certified across 10 random images. As shown in Fig. 3.16, our probabilistic framework consistently certifies robustness levels at least 1.5 times higher than the worst-case baseline, even when using more recent LiRPA techniques. Notably, the results indicate that as the estimated reachable sets become more precise through over-approximation, the impact of our approach diminishes. Nonetheless, our framework can provide interesting safety information on the model's robustness level even with very tight provable reachable sets.

Finally, to assess the computational overhead introduced by the proposed approach, we perform an ablation study analyzing its impact on the total certification time, considering $n = 350k$ samples and progressively larger network sizes. We

highlight that the overall time complexity of the algorithm is polynomial in the network size, specifically $O\left(n \sum_{i=1}^{N} d_i d_{i-1}\right)$, where the term $\sum_{i=1}^{N} d_i d_{i-1}$ represents the complexity for a single sample, where N is the total number of layers in the network, d_{i-1} is the number of neurons in the preceding layer, and d_i is the number of neurons in the current layer. The product $d_i d_{i-1}$ therefore indicates the number of multiplications required for the processing of layer i. Consequently, the total complexity is proportional to the number of samples multiplied by the cost of a single forward propagation through the entire network. This aspect is well highlighted in the results of Tab. 3.6, where the time overhead of the sampling-based approach in `PT-LiRPA` is negligible (i.e., less than one second) in the overall certification time, even when employing a significantly large number of samples, thanks to the GPU acceleration employed in the certification process. Clearly, the total computation time of CROWN enhanced with `PT-LiRPA` is greater than that of using CROWN alone, as the probabilistic certification of a larger tolerable input perturbation results in a longer verification process.

Certification method	Worst-case (CROWN)	CROWN w/ PT-LiRPA				
	Total certification time	Total certification time	# computations of interm. bounds	# samples	Total intem. bounds computation time	Mean interm. bounds computation time
MNIST 2×[1024], ReLU	35.7s	47.4s	253	350k	0.27s	0.001s
MNIST 3×[1024], ReLU	37.73s	54.85s	248	350k	0.34s	0.0014s
MNIST 2×[1024], Tanh	23.02s	36.73s	165	350k	0.22s	0.0013s
MNIST 3×[1024], Tanh	31.47s	59.82s	186	350k	0.26s	0.0014s
MNIST 4×[1024], Sigmoid	40.26s	64.61s	209	350k	0.33s	0.0016s
CIFAR 5×[2048], ReLU	26.3s	82.42s	120	350k	0.48s	0.004s
CIFAR 7×[1024], ReLU	21.02s	73.9s	154	350k	0.37s	0.0024s

Table 3.6: Time comparison between CROWN and CROWN enhanced with `PT-LiRPA` for the certification of the models in Table 3.5. The *Total certification time* column reports the overall time required to compute the average ε perturbation that the model can tolerate across 10 random test images. The *# computations of interm. bounds* column indicates how many times the intermediate bound computation procedure is invoked to determine the mean ε, while the *# samples* column specifies the number of samples used in each instance of intermediate bound computation. Finally, the last two columns present the total overhead and the average time per call of the sampling-based approach used to compute the probabilistically optimal intermediate bounds.

Importantly, the time comparison is conducted only against CROWN as the goal of this experiment is to show that the additional cost introduced by `PT-LiRPA` is negligible in the overall verification time, while still producing tighter output bounds. Importantly, once the LiRPA baseline is fixed (e.g., α-CROWN, β-CROWN, GCP-CROWN), the subsequent linearization procedure is identical whether using the original method or our `PT-LiRPA` variant. The only difference lies in the way intermediate bounds are computed, which are then used to construct the diagonal matrices and bias terms. Consequently, we argue that measuring the overhead against CROWN is representative, since the additional cost introduced by `PT-LiRPA` does not depend on which LiRPA baseline is employed.

Answers to Q3.

To answer the last question, we integrate our `PT-LiRPA` in the α, β-CROWN toolbox (`https://github.com/Verified-Intelligence/alpha-beta-CROWN`) and perform a final experiment on different benchmarks of the VNN-COMP 2022 and 2023 [198, 35]. We point out that the comparison between probabilistic and provable verifiers is performed to have a ground truth (when possible), and to highlight the valuable help that probabilistic approaches can have in solving challenging instances to be verified.
To keep the section self-contained, we report below a brief overview of the selected benchmarks.

- *ACAS xu* [125, 131] benchmark 2023: includes ten properties evaluated across 45 neural networks designed to provide turn advisories for aircraft to prevent collisions. Each neural network consists of 300 neurons distributed over six layers, using ReLU activation functions. The networks take five inputs representing the aircraft's state and produce five outputs, with the advisory determined by the minimum output value. Here, we verified only property 3, which returns unsafe if the clear of conflict (COC) output is minimal, with a max computation time of 116s.
- *TllVerifyBench* benchmark 2023: this benchmark features Two-Level Lattice (TLL) neural networks with two inputs and one single output. These models are then transformed into MLP ReLU networks where the output properties consist of a randomly generated real number and a randomly generated inequality direction to be verified. Here we verify all 32 instances of the VNN-COMP 2023 with a timeout of 600s for each property.
- *CIFAR_biasfield* benchmark 2022: this benchmark focuses on verifying a Cifar-10 network under bias field perturbations. These perturbations are modeled by creating augmented networks that reduce the input space to just 16 parameters. For each image to be verified, a distinct bias field transform network is generated, consisting of a fully connected transform layer followed by the Cifar CNN with 8 convolutional layers with ReLU activations. Each bias field transform network has 363k parameters and 45k nodes. Here, we test all 72 properties with a timeout set to 300s for each one.
- *TinyImageNet* benchmark 2022: consists of CIFAR100 image classification ($56\times 56\times 3$) with Residual Neural Networks (ResNet). Here, we consider the medium network size composed of 8 residual blocks, 17 convolutional layers, and 2 linear layers. For *TinyImageNet-ResNet-medium*, we verify all 24 properties with a timeout of 200 seconds for each property.

In general, we selected benchmarks where the state-of-the-art α, β-CROWN method is unable to solve some of the instances within the time constraints. This set of experiments aims to confirm our hypothesis regarding the effectiveness of having tighter estimated reachable sets for verification purposes. In detail, our intuition is that even though our procedure requires a little initial overhead, with tighter estimated reachable sets, we can achieve more precise final output bounds, potentially reducing the cases where the verification approach can not make a decision and must resort to a split in the BaB process, thus resulting in more efficient overall verification time.

Before starting to verify these instances, we explore the effect of varying incremental sample sizes for a fixed $\xi = 0.85$ on the computation of estimated reachable sets with neural networks of significant size.

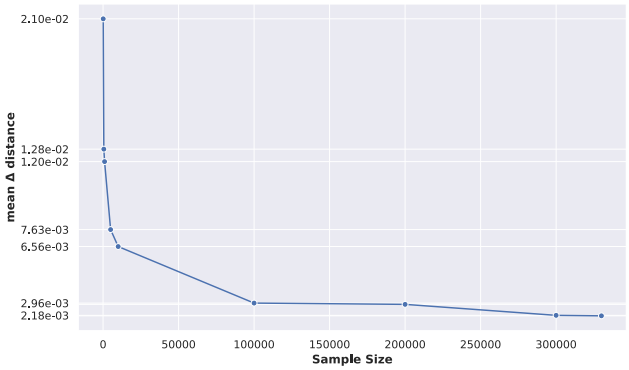


Fig. 3.17: Estimated reachable sets at convergence for the increasing sample size in *CIFAR_biasfield* benchmark. y-axis reports the mean distance between estimated reachable sets using $350k$ samples (as reference) and the one using $[100, 500, 1k, 5k, 10k, 100k, 200k, 300k, 330k]$, respectively.

In detail, we focus on the *CIFAR_biasfield* benchmark and set a confidence level of $1 - 2mp \geq 99\%$. In detail, we compute the mean distance between estimated reachable sets using $350k$ samples as reference and the one using progressively increasing the sample size until the difference between successive estimated reachable sets exceeds the threshold of $\Delta = 0.001$. Specifically, we begin with 100 samples and progressively increase the sample size until the difference between successive estimated reachable sets exceeds the threshold. Our results, detailed in Fig. 3.17, indicate that stable estimated reachable sets, in this scenario, can be obtained with sample sizes ranging from $300k$ to $350k$ as the mean distance between estimated bounds is strictly less than $\Delta = 0.001$. Hence, employing a sample size of $350k$ samples results in a valid choice even for considerably large networks. We recall once again that propagating a large number of samples, such as $350k$, requires a negligible computational effort and time (as shown in the results of Tab. 3.6) due to batch processing and GPU acceleration. The principal limitation arises from GPU memory capacity, since larger sample sizes may increase the risk of memory errors relative to CPU-based propagation.

In Tab. 3.7 we report our results on the VNN-COMP, where we consider an increased difficulty for the verification process. We start with the simpler benchmark *ACASxu* [125, 131], and we test property 3. This property is particularly interesting as it holds for 42 of the 45 models tested, thus allowing us to verify the improvement in terms of time and verification accuracy. In the first row of Tab. 3.7, we can notice that by sacrificing only a 0.01% of confidence, α, β-CROWN enhanced with `PT-LiRPA` achieves the same verified accuracy in less verification time, thus confirming our intuition. Interestingly, we observe that tighter bounds are not always beneficial in general. Specifically, in cases where a PGD attack succeeds despite loose bounds, using tighter bounds does not lead to further improvements.

Benchmark	Method	Confidence	Verified accuracy	#safe (unsat)	#unsafe (sat)	#unkwown	Tot verification time
ACASxu	α,β-CROWN	100%	93.33%	42	3	0	26s
	α,β-CROWN w/PT-LiRPA	$\geq$ 99%	93.33%	42	3	0	**16.37s**
tllVerifyBench	α,β-CROWN	100%	46.875%	15	17	0	90.2s
	α,β-CROWN w/ PT-LiRPA	$\geq$ 99%	46.875%	15	17	0	92s
CIFAR_biasfield	α,β-CROWN	100%	95.83%	69	1	2	1553.5s
	α,β-CROWN w/ PT-LiRPA	$\geq$ 99%	**98.61%**	**71**	1	**0**	**408.7s**
CIFAR_tinyimagenet	α,β-CROWN	100%	62.5%	15	3	6	1429.6s
	α,β-CROWN w/ PT-LiRPA	$\geq$ 99%	**87.5%**	**21**	3	**0**	**425.6s**

Table 3.7: Results on VNN-COMP 2022-2023 benchmarks. Results marked in **bold** report improved performance in terms of verified accuracy (% sat instances/all instances) and total verification time for the specific benchmark tested, wr.r.t. a worst-case verification approach.

Additionally, in some scenarios, less accurate bounds from vanilla LiRPA methods could be quickly refined by BaB, still resulting in efficient verification time. This is exemplified by the *tllVerifyBench* experiments, where even sacrificing a 0.01% of confidence, PT-LiRPA produced tight estimated reachable sets but achieved the same verified accuracy with a minor overhead in bounds computation.
Crucially, the real benefit of our PT-LiRPA arises on more challenging verification benchmarks such as *CIFAR_biasfield* and *CIFAR_tinyimagenet*. Both these benchmarks are image-based verification tasks and thus allow us to show the scalability and the impact of tightened estimated reachable sets on large networks using the proposed approach. Crucially, in these two last benchmarks, sacrificing a 0.01% of confidence, we obtain significant improvements in verification results with respect to worst-case α, β-CROWN. In detail, in both *CIFAR_biasfield* and *CIFAR_tinyimagenet*, we achieved higher verified accuracy without incurring any *unknown* answer and with significantly less verification time. These final results demonstrate the effectiveness and the potential impact of using PT-LiRPA for verification purposes, showing the advantage of incorporating estimated reachable sets in handling challenging instances that are difficult to solve with provable solvers.

3.2.7 Assumptions and Limitations

The proposed PT-LiRPA framework builds on several assumptions that define its applicability and theoretical guarantees. Our probabilistic framework relies on uniform random sampling within the perturbation region $\mathcal{P}_{\mathbf{x}_0,\epsilon}$ to estimate probabilistically tight reachable sets. Consequently, our theoretical probabilistic guarantees, derived from Wilks [286]'s tolerance limit theorem and extended using extreme value theory, hold under the assumption that the samples are independent and representative of the true input distribution within $\mathcal{P}$.

Importantly, by accepting certificates that hold for a fraction R of the perturbation region, the theoretical and practical tool derived from Wilks [286]'s results, i.e., Theorem 3.2, remains a valuable solution, as it provides a closed-form expression to compute the number of samples required for any desired confidence level ψ and coverage ratio R. To address this limitation and extend the analysis to probabilistic certificates valid over the entire perturbation region, thus aligning with the probabilistic verification literature, we further based our theoretical derivation on extreme value theory. In this case, the result of Theorem 3.4 provides, for any choice of sam-

ple size n and confidence error p, a guarantee that holds for all input values $\boldsymbol{x} \in \mathcal{P}$. However, unlike Wilks' approach, EVT does not yield a closed-form expression to determine the minimum number of samples required to achieve a desired precision and confidence level. Consequently, the balance between probabilistic soundness and the tightness of the resulting output bounds becomes more empirical, as confirmed in our experiments. In fact, the tightness of the computed bounds and the computational efficiency of the method depend on the chosen sample size n, and the estimate of the tail distribution $\nu = \lfloor n^{\xi} \rfloor$, which must balance verification accuracy and cost.

Despite these limitations, `PT-LiRPA` complements deterministic verification methods by offering practical, quantifiable robustness guarantees for challenging instances where worst-case formal verification is either overly conservative or computationally infeasible.

Summary. In this section, we introduced `PT-LiRPA`, a novel probabilistic framework that combines LiRPA-based formal verification of deep neural network approaches with a sampling-based technique. We provide a rigorous theoretical derivation of the correctness of our approach, complementing, for the first time, statistical results on the tolerance limit, with qualitative bounds on the error magnitude of a sampling-based approach employed to estimate reachable set domains. Our approach provides tighter linear bounds, significantly improving both the accuracy and verification efficiency while maintaining provable probabilistic guarantees on the soundness of the verification result. Empirical results demonstrate that `PT-LiRPA` outperforms related probabilistic methods, particularly in robustness certification, decreasing the confidence in the result by infinitesimal amounts. Additionally, we show the potential of our probabilistic approach for verifying challenging instances where the formal approaches fail. Although this novel solution provides promising results in terms of scalability, it focuses on standard verification, which may not be enough to fully understand the safety level of a model. Traditional robustness verification only determines whether at least one adversarial or unsafe input exists within a given region, but it does not quantify how widespread such unsafe inputs are. In many safety-critical domains, we argue that we also need to understand the probability or fraction of the input domain that leads to failures, so as to assess the practical risk and guide mitigation strategies. This motivates a shift toward the counting of unsafe inputs, formalized in the next section as the #DNN-VERIFICATION problem, which extends robustness verification from a Boolean decision problem to a quantitative measure of the unsafe input space.

3.3 The #DNN-Verification Problem: Counting Unsafe Inputs for Deep Neural Networks

Given a DNN and a safety property, a DNN verification tool should ideally either ensure that the property is satisfied for all the possible input configurations or identify a specific example (e.g., adversarial configuration) that violates the requirements. Given these complex functions' non-linear and non-convex nature, verifying even simple properties has been proven to be an NP-complete problem [131]. In literature, several works try to solve the problem efficiently either by *satisfiability modulo theories* (SMT) solvers or by *reachability analysis* methods as described in Chapter 2.

Although these methods show promising results, the current formulation, widely adopted for almost all the approaches, considers only the decision version of the formal verification problem, with the solution being a binary answer whose possible values are typically denoted `SAT` or `UNSAT`. `SAT` indicates that the verification framework found a specific input configuration, as a counterexample, that caused a violation of the requirements. `UNSAT`, in contrast, indicates that no such point exists, and then the safety property is formally verified in the whole input space. While an `UNSAT` answer does not require further investigations, a `SAT` result hides additional information and questions. For example, how many of such adversarial configurations exist in the input space? How likely are these misbehaviors to happen during a standard execution? Can we estimate the probability of running into one of these points? These questions can be better dealt with in terms of the problem of *counting the number of violations* of a safety property, a problem that might be important also in other contexts: (i) *model selection*: a counting result allows ranking a set of models to select the safest one. This model selection is impossible with a `SAT` or `UNSAT` type verifier, which does not provide any information to discriminate between two models that have both been found to violate the safety condition for at least one input configuration. (ii) *guide the training*: knowing the total count of violations for a particular safety property can help guide the training of a deep neural network in a more informed fashion, for instance, minimizing this number over the training process. (iii) *estimating the probability of error*: the ratio of the total number of violations over the size of the input space provides an estimate of the probability of committing an unsafe action given a specific safety property.

Furthermore, by enumerating the violation points, it is possible to perform additional safety-oriented operations, such as: (iv) *shielding*: given the set of input configurations that violate a given safety property, we could adopt a more informative shielding mechanism that prevents unsafe actions. (v) *enrich the training phase*: if we can enumerate the violation configurations, we could add these configurations to the training (or to a memory buffer in a deep reinforcement learning setup) to improve the training phase in a safe-oriented fashion. Motivated by the above questions and applications, previous works [17, 307, 80] propose a *quantitative* analysis of neural networks, focusing on a specific subcategory of these functions, i.e., Binarized Neural Networks (BNN). However, violation points are generally not preserved in the binarization of a DNN to a BNN nor conversely in the relaxation of a BNN to a DNN [307].

To this end, in this section, we introduce the #DNN-VERIFICATION problem, which is the extension of the decision DNN-Verification problem to the corresponding counting version. Given a general deep neural network (with continuous values) and a safety property, the objective is to count the exact number of input configurations that violate the specific requirement. We analyze the complexity of this problem and propose two solution approaches. In particular, in the first part of the section, we propose an algorithm that is guaranteed to find the *exact* number of unsafe input configurations, providing a detailed theoretical analysis of the algorithm's complexity. The high-level intuition behind our method is to recursively shrink the domain of the property, exploiting the `SAT` or `UNSAT` answer of a standard formal verifier to drive the expansion of a tree that tracks the generated subdomains. Interestingly, our method can rely on any formal verifier for the decision problem, taking advantage of all the improvements to state-of-the-art and, possibly, to novel frameworks. As the analysis shows, our algorithm requires multiple invocations of the verification tool, resulting in significant overhead and becoming quickly unfeasible for real-world problems. For this reason, inspired by the work of [84] on #SAT, we propose an approximation algorithm for #DNN-VERIFICATION, providing provable (probabilistic) bounds on the correctness of the estimation.

In more detail, in this section, we make the following contribution to the state-of-the-art:

- We propose an exact count formal algorithm to solve #DNN-VERIFICATION, that exploits state-of-the-art decision tools as backends for the computation.
- We present `CountingProVe`, a novel approximation algorithm for #DNN-VERIFICATION, that provides a bounded confidence interval for the results.
- We evaluate our algorithms on a standard benchmark, ACAS Xu, showing that `CountingProVe` is scalable and effective also for real-world problems.

To the best of our knowledge, this is the first study to present #DNN-VERIFICATION, the counting version of the decision problem of the formal verification for general neural networks without converting the DNN into a CNF.

3.3.1 Preliminaries

In this section, we first provide a formal definition of the #DNN-VERIFICATION problem. Throughout the presentation, we adopt the *satisfiability formulation* of DNN-VERIFICATION, where $\mathcal{X}$ and $\mathcal{Y}$ denote Boolean predicates specifying the pre- and post-conditions, respectively. In the subsequent sections, we introduce an algorithm that solves the problem by leveraging any existing formal verification tool. We then present a theoretical analysis of the problem's complexity, which underscores the necessity of approximate methods.

3.3.2 Problem Formulation

Given a tuple $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$, as in Definition 3.1, we let $\Gamma(\mathcal{T})$ denote the set of *all* the input configurations for f satisfying the property defined by $\mathcal{X}$ and $\mathcal{Y}$, i.e.

$$\Gamma(\mathcal{T}) = \left\{ \boldsymbol{x} \;\middle|\; \mathcal{X}(\boldsymbol{x}) \wedge \mathcal{Y}(f(\boldsymbol{x})) \right\}$$

Then, the #DNN-Verification consists of computing the cardinality of $\Gamma(\mathcal{T})$.

Definition 3.6 (*#DNN-Verification Problem*):
Input: *A tuple $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$, as in Definition 2.1.*
Output: $|\Gamma(\mathcal{T})|$

For the purposes discussed in the introduction, rather than the cardinality of $\Gamma(\mathcal{T})$, it is more useful to define the problem in terms of the ratio between the cardinality of Γ and the cardinality of the set of inputs satisfying $\mathcal{X}$. We refer to this ratio as the *violation rate (VR)*, and study the result of the #DNN-Verification problem in terms of this equivalent measure.

Definition 3.7 (*Violation Rate (VR)*): *Given an instance of the DNN-Verification problem $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$ we define the violation rate as*

$$VR = \frac{|\Gamma(\mathcal{T})|}{|\{x \mid \mathcal{X}(\boldsymbol{x})\}|}$$

Although, in general, DNNs can handle continuous spaces, in the following sections (and for the analysis of the algorithms), without loss of generality, we assume the input space to be discrete. We remark that for all practical purposes, this is not a limitation since we can assume that the discretization is made to the maximum resolution achievable with the number of decimal digits a machine can represent. It is crucial to point out that discretization is not a requirement of the approaches proposed in this work. In fact, supposing to have a backend that can deal with continuous values, our solutions would not require such discretization.

3.3.3 Exact Count Algorithm for #DNN-Verification

We now present an algorithm to solve the exact count of #DNN-Verification.

The algorithm recursively splits the input space into two parts of equal size as long as it contains both a point that violates the property (i.e., $(f, \mathcal{X}, \mathcal{Y})$ is a SAT-instance for DNN-Verification problem) and a point that satisfies it (i.e., $(f, \mathcal{X}, \neg\mathcal{Y})$ is a SAT-instance for DNN-Verification problem).[8] The leaves of the recursion tree of this procedure correspond to a partition of the input space into parts where the violation rate is either 0 or 1. Therefore, the overall violation rate is easily computable by summing up the sizes of the subinput spaces in the leaves of violation rate 1. Fig. 3.26 shows an example of the execution of our algorithm for the DNN and the

[8] Any state-of-the-art sound and complete verifier for the decision problem can be used to solve these instances. In fact, our method works with any state-of-the-art verifiers, although using a verifier that is not complete can lead to over-approximation in the final count.

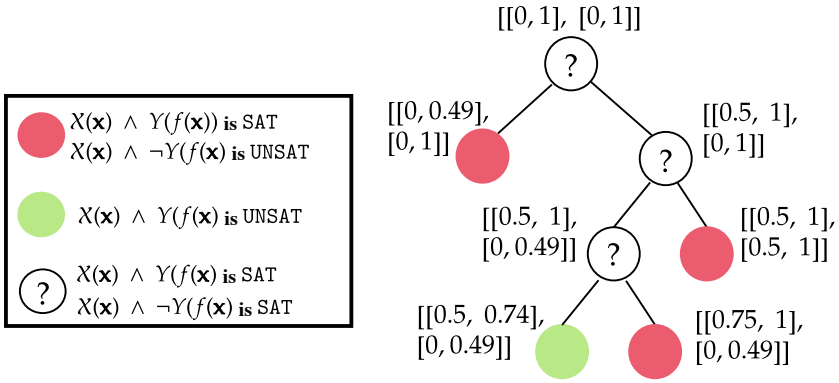


Fig. 3.18: Example execution of exact count for a particular f and safety property (assuming a discretization factor of 0.01).

safety property $\mathcal{X}(\boldsymbol{x})$ true if $\boldsymbol{x} \in [0,1] \times [0,1]$ and $\mathcal{Y}(f(\boldsymbol{x}))$ true if $f(\boldsymbol{x}) \geq 0$. We want to enumerate the total number of input configurations $\mathbf{x} = (x_1, x_2)$ where both x_1 and x_2 satisfy $\mathcal{X}$ (i.e., they lie in the interval $[0,1]$) and such that the output $z^{(3)} = f(\mathbf{x})$ is a value strictly less than 0, i.e., $\mathbf{x}$ violates the safety property.

The algorithm starts with the whole input space and checks if at least one point exists that outputs a value strictly less than zero. The exact count method checks the predicate $\exists \boldsymbol{x} \mid \mathcal{X}(\boldsymbol{x}) \wedge \mathcal{Y}(f(\boldsymbol{x}))$ with a verification tool for the decision problem. If the result is `UNSAT`, then the property holds in the whole input space, and the algorithm returns a VR of 0%. Otherwise, if the verification tool returns `SAT`, at least one single violation point exists, but we cannot assert how many similar configurations exist in the input space. To this end, the algorithm checks another property equivalent to the original one, thus negating the postcondition (i.e., $\neg\mathcal{Y} = z^{(3)} \geq 0$). Here, we have two possible outcomes:

- $\mathcal{X}(\boldsymbol{x}_0) \wedge \neg\mathcal{Y}(f(\boldsymbol{x}_0))$ is `UNSAT` implies that all possible $\boldsymbol{x}_0 = (x_1, x_2)$ that satisfy $\mathcal{X}$, output a value strictly less than 0, violating the safety property. Hence, the algorithm returns a 100% of *VR* in the input area represented by $\mathcal{X}$. This situation is depicted with the red circle in Fig. 3.26.
- $\mathcal{X}(\boldsymbol{x}_0) \wedge \neg\mathcal{Y}(f(\boldsymbol{x}_0))$ is `SAT` implies that there is at least one input configuration $\mathbf{x}$ satisfying $\mathcal{X}$ and such that $f(\mathbf{x}) \geq 0$. Therefore, the algorithm cannot assert anything about the violated portion of the area since there are some input points on which f generates outputs greater or equal to zero and some others that generate a result strictly less than zero. Hence, the procedure splits the input space into two parts, as depicted in Fig. 3.26.

This process is repeated until the algorithm reaches one of the base cases, such as a situation in which either $\mathcal{X}(\boldsymbol{x}) \wedge \mathcal{Y}(f(\boldsymbol{x}))$ is not satisfiable (i.e., all the current portion of the input space is safe), represented by a green circle in Fig.3.26), or $\mathcal{X}(\boldsymbol{x}) \wedge \neg\mathcal{Y}(f(\boldsymbol{x}))$ is not satisfiable (i.e., the whole current portion of the input space is unsafe), represented by a red circle. Note that the option of obtaining `UNSAT` on both properties is not possible.

Finally, the algorithm of exact count returns the *VR* as the ratio between the unsafe areas (i.e., the red circles) and the original input area. In the example of

Fig. 3.26, assuming a 2-decimal-digit discretization, we obtain a total number of violation points equal to 8951. Normalizing this value by the initial total number of points (10201), and considering the percentage value, we obtain a final *VR* of 87.7%. Moreover, the *Violation Rate* can also be interpreted as the probability of violating a safety property using a particular DNN f. In more detail, uniformly sampling 10 random input vectors for our DNN f in the intervals described by $\mathcal{X}$, 8 over 10 with high probability violates the safety property.

3.3.4 Hardness of #DNN-Verification

It is known that finding even just one input configuration (also referred to as a violation point) that violates a given safety property is computationally hard since the DNN-Verification problem is NP-complete [131]. Hence, counting and enumerating all the violation points is expected to be an even harder problem.

Theorem 3.5: *The #DNN-Verification is #P-hard.*

Proof. The proof of #P-hardness is similar and follows the one of NP-hardness; the main difference is in the concept of polynomial time *counting reduction*. As stated in [265] and [85], many NP-complete problems are *parsimonious*, meaning that for almost all pairs of NP-complete problems, there exist polynomial transformations between them that preserve the number of solutions. Hence, the reduction between two NP-complete problems can be directly taken as part of a counting reduction, thus providing an easy path to proving #P-hardness.

For our purpose, we follow the reduction between 3-SAT and DNN-Verification provided in the work of Katz et al. [131]. In more detail, we assume that the input nodes take the discrete values in $\{0, 1\}$ for simplicity. Note that this limitation can be relaxed using an ε discretization to consider a range between $[a, b]$ for the input space.

Recalling the hardness proof, we know that any 3-SAT formula Φ can be transformed into a DNN f (with ReLUs activation functions) and a property ϕ, such that ϕ is satisfiable on f if and only if Φ is satisfiable. Specifically, Katz et al. [131] provided three useful gadgets to perform the reduction:

1. **disjunction gadget** that maps a disjunction of three literals in a 3-CNF formula to the same result for a group of three nodes in a DNN. Formally this gadget performs the following transformation: $y_i = 1 - \max(0, 1 - \sum_{j=1}^{3} q_i^j)$. Where y_i is the node that collects the result of the linear combination and subsequent $ReLU$ activation of up to 3 nodes (q_i^j) from the previous layer. Hence, y_i will be 1 if at least one input variable is set to 1 (or true), and y_i will be 0 if all input variables are set to 0. In words, this gadget maps a disjunction of literals in a 3-CNF to a combination of nodes in a DNN, such that there is a one-one correspondence between the output of the nodes on a 0-1 input and the truth value computed by the disjunction over the equivalent truth values.
2. **negation gadget** that on input $x_i \in \{0, 1\}$ produces the output value $y_i = 1 - x_j$, hence modelling the exact behaviour of a logical negation.
3. **conjuction gadget** which maps the satisfiability of a 3-CNF Φ into a ϕ satisfiability for a DNN. In particular, Φ is satisfied only if all clauses $C_1, \ldots, C_n$

are simultaneously satisfied. Hence, if all the nodes are in the domain $\{0, 1\}$, for satisfiability, we want the resulting output of a forward propagation equal to n, i.e., the number of clauses. This gadget maps the conjunction of n clauses in a 3-CNF, i.e., the satisfiability, into the linear combination of n nodes to produce an output value. Therefore, $C_1 \wedge C_2 \wedge \cdots \wedge C_n = true$ if and only if the output of the gadget is n.

From the combination of these three gadgets, we obtain a reduction transforming a 3-CNF formula ϕ into a DNN $\mathcal{N}$. Let us consider the instance to the DNN-Verification problem asking to check whether there exists an input configuration on which $\mathcal{N}$ outputs a value different from n. Then, we have that the formula ϕ is satisfiable, i.e., there is a truth assignment to the input variables if and only if for the DNN $\mathcal{N}$ there exists an input configuration (in fact necessarily only using values in $\{0, 1\}$) that induces the output $y = n$, i.e., if and only if, there exists a violation.

As observed, this reduction also shows that each distinct satisfying assignment for ϕ is mapped to a distinct input configuration producing output n and vice versa, each input configuration on which $\mathcal{N}$ outputs n must be $\{0, 1\}$-valued and corresponds to a truth assignment that satisfies ϕ.

Therefore, counting the number of satisfying assignments for ϕ is equivalent to counting the number of violations for $\mathcal{N}$. Hence, from the #P-hardness of #3-SAT [265] it follows (via the above reduction) that also #DNN-Verification is #P-hard. □

3.3.5 CountingProVe for Approximate Count

In view of the #P-hardness of the #DNN-Verification problem, it is not surprising that the time complexity of the algorithm for the exact count worsens very fast when the size of the instance increases. In fact, moderately large networks are also intractable with this approach. To overcome these limitations while still providing guarantees on the quality of the results, we propose a randomized-approximation algorithm, `CountingProVe` (presented in Algorithm 7).

To understand the intuition behind our approach, consider Fig. 3.26. If we could assume that each split distributed the violation points evenly in the two subinstances it produces, for computing the number of violation points in the whole input space, it would be enough to compute the number of violation points in the subspace of a single leaf and multiplying it by 2^s (which represents the number of leaves). Since we cannot guarantee a perfectly balanced distribution of the violation points among the leaves, we propose to use a heuristic strategy to split the input domain, balancing the number of violation points in expectation. This strategy allows us to estimate the count and a provable confidence interval on the actual violation rate. In more detail, our algorithm receives as input the tuple for a DNN-Verification problem (i.e., $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$) and to obtain a precise estimate of the *VR*, performs t iterations of the following procedure.

Initially, we assume that, in the whole input domain, the safety property does not hold, i.e., we have a 100% of *Violation Rate*. The idea is to repeatedly simplify the input space by performing a sequence of multiple splits, where each i-th split implied a reduction of the input space of a factor α_i. At the end of s simplifications, the goal is

Algorithm 7: `CountingProVe`

1: **Input:** $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$, t (# of repetitions), m (# of violation points sampled per iteration), $\beta > 0$ (error tolerance factor).
2: **Output:** lower bound of the violation rate.
3: **for** t=1 to t **do**
4: $VR_t \leftarrow 100\%$, $s \leftarrow 0$
5: **while** Timeout(`ExactCount`($\mathcal{T}$)) **do**
6: $S \leftarrow$ `SampleViolationPoints`($\mathcal{T}, m$)
7: $median \leftarrow$ `ComputeMedian`($S, node_i$)
8: $node_{i,0}, node_{i,1} \leftarrow$ `SplitInterval`($node_i, median$)
9: $side \leftarrow$ a random value chosen uniformly from {0,1}
10: $\mathcal{X} \leftarrow$ `UpdateX`($\mathcal{X}, node_{i,side}$)
11: $s \leftarrow s + 1$
12: $VR_t \leftarrow 2^{s-\beta} \cdot$ `ExactCount`($\mathcal{T}$) $\cdot \prod_{i=1}^{s} \alpha_i$
13: $VR \leftarrow min(VR_t, VR)$
14: **return** VR

to obtain an exact count of unsafe input configurations that can be used to estimate the VR of the entire input space. Specifically, Algorithm 7 presents the pseudo-code for the heuristic approach. Before splitting, the algorithm attempts to use the exact count but stops if not completed within a fixed timeout which we set to just a fraction of a second (line 5). Inside the loop, the procedure `SampleViolationPoints`($\mathcal{T}, m$) samples m violation points from the subset of the input space described in $\mathcal{X}$ (using a uniform sampling strategy) and saves them in S.

After line 6, the algorithm has an estimation of how many violation points there are in the portion of the input space under consideration.[9] The idea is to simplify one dimension of the hyperrectangle cyclically while keeping the number of violation points balanced in the two portions of the domain we aim to split (i.e., as close to $|S|/2$ as possible). Specifically, `ComputeMedian`($S, node_i$) (line 7) computes the median of the values along the dimension of the node chosen for the split. `SplitInterval`($node_i, median$) splits the node into two parts $node_{i,0}$ and $node_{i,1}$ according to the median computed. For instance, suppose we have an interval of $[0, 1]$ and the median value is 0.4. The two parts obtained by calling `SplitInterval`($node_i, median$) are $[0, 0.4]$ and $(0.4, 1]$. The algorithm proceeds randomly, selecting the side to consider from that moment on and discarding the other part. Hence, it updates the input space intervals to be considered for the safety property $\mathcal{X}$ (lines 9-10). Finally, the variable s that represents the number of splits made during the iteration is incremented. At the end of the while loop (line 12), the VR is computed by multiplying the result of the exact count by $2^{s-\beta}$, and for $\prod_{i=1}^{s} \alpha_i$. The first term ($2^{s-\beta}$) considers the fact that we balanced the VR in our tree, hence selecting one path and multiplying by 2^s, we get a representative estimate of the whole starting input area. β is a tolerance error factor, and finally, $\prod_{i=1}^{s} \alpha_i$ describes

[9] if the m solutions are not found, the algorithm proceeds by considering only the violation points discovered or splitting at random.

how we simplified the input space during the s splits made (line 13). Finally, the algorithm refines the lower bound (line 14).

The correctness of our algorithm is independent of the number of samples used to compute the median value. However, using a poor heuristic (i.e., too few samples or a suboptimal splitting technique), the bound's quality can change, as the lower bound may get farther away from the actual *Violation Rate*. We report in Fig. 3.19 an example of the execution of `CountingProVe`.

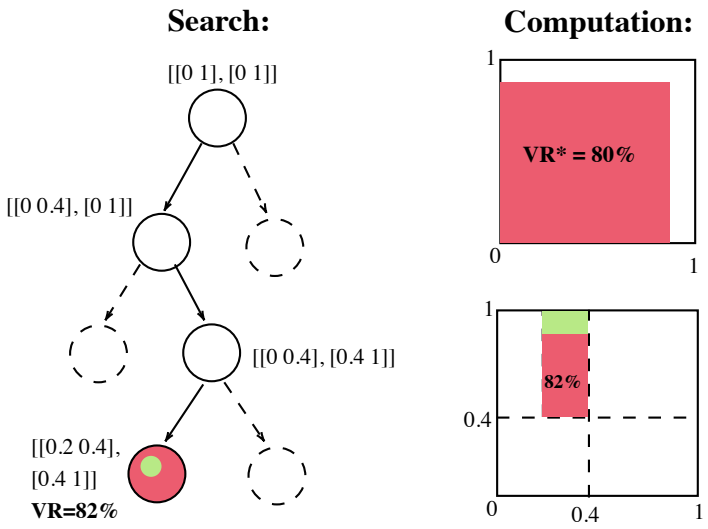


Fig. 3.19: Example of computation with `CountingProVe`.

The crucial aspect of `CountingProVe` is that even when the splitting method is arbitrarily poor, the bound returned is provably correct (from a probabilistic point of view), see the next section for more details.

A Provable Lower Bound

In this subsection, we show that the randomized-approximation algorithm `CountingProVe` returns a correct lower bound for the *Violation Rate* with a probability greater (or equal) than $(1-2^{-\beta t})$. In more detail, we demonstrate that the error of our approximation decreases exponentially to zero and that it does not depend on the number of violation points sampled during the simplification process nor on the heuristic for the node splitting.

Theorem 3.6: *Given the tuple $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$, the Violation Rate returned by the randomized-approximation algorithm* `CountingProVe` *is a correct lower bound with a probability* $\geq (1 - 2^{-\beta t})$.

Proof. Let $VR^* > 0$ be the actual violation rate. Then, `CountingProVe` returns an incorrect lower bound, if for each iteration, $VR > VR^*$. The key of this proof is to show that for any iteration, $Pr(VR > VR^*) < 2^{-\beta}$.

Fix an iteration t. The input space described by $\mathcal{X}$ and under consideration within the while loop is repeatedly simplified. Assume we have gone through a sequence

of s splits, where, as reported in Sec.3.3.5, the i-th split implied a reduction of the solution space of a factor α_i. For a violation point σ, we let Y_σ be a random variable that is 1 if σ is also a violation point in the restriction of the input space that has been (randomly) obtained after s splits (and that we refer to as the solution space of the leaf ℓ).

Hence, the *VR* obtained using an exact count method for a particular ℓ is $VR_\ell = \frac{\sum_{\sigma\in\Gamma} Y_\sigma}{A_\ell}$, i.e., the ratio between the number of violation points and the number of points in ℓ (A_ℓ). Then our algorithm returns as an estimate of the total *VR* computed by:

$$VR = 2^{s-\beta} \cdot VR_\ell \cdot \prod_{i=1}^{s} \alpha_i = 2^{s-\beta} \cdot \frac{\sum_{\sigma\in\Gamma} Y_\sigma}{A_\ell} \cdot \prod_{i=1}^{s} \alpha_i \tag{3.11}$$

Let $\mathbf{s}$ denote the sequence of s random splits followed to reach the leaf ℓ. We are interested in $\mathbb{E}[VR] = \mathbb{E}[\mathbb{E}[VR|\mathbf{s}]]$. The inner conditional expectation can be written as:

$$\mathbb{E}[VR \mid \mathbf{s}] = \mathbb{E}\Big[2^{s-\beta} \cdot \frac{\sum_{\sigma\in\Gamma} Y_\sigma}{A_\ell} \cdot \prod_{i=1}^{s} \alpha_i \,\Big|\, \mathbf{s}\Big] \tag{3.12}$$

$$= \mathbb{E}\Big[2^{s-\beta} \cdot \frac{\sum_{\sigma\in\Gamma} Y_\sigma}{A_{Tot}} \,\Big|\, \mathbf{s}\Big] \tag{3.13}$$

$$= \frac{2^{s-\beta}}{A_{Tot}} \sum_{\sigma} \mathbb{E}[Y_\sigma = 1 \mid \mathbf{s}] \tag{3.14}$$

$$= \frac{2^{s-\beta}}{A_{Tot}} \sum_{\sigma} 2^{-s} \tag{3.15}$$

$$= 2^{-\beta} \frac{\sum_{\sigma\in\Gamma} 1}{A_{Tot}} = 2^{-\beta} VR^* \tag{3.16}$$

where the equality in (2) follows from rewriting *VR* using (1). Equality (3) follows from (2) using the relation $A_{Tot} = \frac{A_\ell}{\prod_{i=1}^{s} \alpha_i}$. Then we have (4) that follows from (3) by using the linearity of the expectation. (5) follows from (4) since, in each split, we choose the side to take independently and with probability $1/2$. Finally, (6) follows by recalling that $\sum_{\sigma\in\Gamma} 1$ is the total number of violations in the whole input space, hence $VR^* = \frac{\sum_{\sigma\in\Gamma} 1}{A_{Tot}}$. Therefore, we have

$$\mathbb{E}[VR] = \mathbb{E}[\mathbb{E}[VR|\mathbf{s}]] = \mathbb{E}[2^{-\beta} VR^*] = 2^{-\beta} VR^*.$$

Finally, by using Markov's inequality, we obtain that

$$Pr(VR > VR^*) < \frac{\mathbb{E}[VR]}{VR^*} = \frac{2^{-\beta} VR^*}{VR^*} = 2^{-\beta}.$$

Repeating the process t times, we have a probability of overestimation equal to $2^{-\beta t}$. This proves that the *Violation Rate* returned by `CountingProVe` is correct with a probability $\geq (1 - 2^{-\beta t})$. □

A Provable Confidence Interval of the VR

This section shows how a provable confidence interval can be defined for the *VR* using `CountingProVe`. In particular, recalling the definition of *Violation Rate* (Def. 4.2), it is possible to define a complementary metric, the *safe rate (SR)*, counting all the input configurations that do not violate the safety property (i.e., provably safe). In particular, we define:

Definition 3.8: *(Safe Rate (SR)) Given an instance of the DNN-VERIFICATION problem $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$ we define the Safe Rate as the ratio between the number of safe points and the total number of points satisfying $\mathcal{X}$. Formally:*

$$SR = \frac{|\{x \mid \mathcal{X}(\boldsymbol{x})\} \setminus \Gamma(\mathcal{T})|}{|\{x \mid \mathcal{X}(\boldsymbol{x})\}|} = \frac{|\Gamma(f, \mathcal{X}, \neg\mathcal{Y})|}{|\{x \mid \mathcal{X}(\boldsymbol{x})\}|}$$

where the numerator indicates the sum of non-violation points in the input space. From the second expression, it is easy to see that `CountingProVe` can be used to compute a lower bound of the *SR*, by running Alg. 7 on the instance $(f, \mathcal{X}, \neg\mathcal{Y})$.

Theorem 3.7: *Given a Deep Neural Network f and a safety property $\langle \mathcal{X}, \mathcal{Y} \rangle$, complementing the lower bound of the Safe Rate obtained using `CountingProVe`, which is correct with a probability $\geq (1-2^{-\beta t})$, we obtain an upper bound for the Violation Rate with the same probability.*

Proof. Suppose we compute the *Safe Rate* with `CountingProVe`. From Theorem 3.6, we know that the probability of overestimating the real *SR* tends exponentially to zero as the iterations t grow. Hence, $Pr(SR > SR^*) < 2^{-\beta t}$. We now consider the *Violation Rate* as the complementary metric of the *Safe Rate*, and we write $VR = 1 - SR$ (as the *SR* is a value in the interval $[0, 1]$). We want to show that the probability that the *VR*, computed as $VR = 1 - SR$, underestimates the real *Violation Rate* (VR^*) is the same as theorem 3.6.

Suppose that at the end of t iterations, we have a $VR < VR^*$. This would imply by definition that $1 - SR < 1 - SR^*$, i.e., that $SR > SR^*$. However, from Theorem 3.6, we know that the probability that `CountingProVe` returns an incorrect lower bound at each iteration is $2^{-\beta t}$. Hence, we obtained that the *VR* computed as $VR = 1 - SR$ is a correct upper bound with the probability (1 - $2^{-\beta t}$) as desired. □

These results allow us to obtain a confidence interval for the *Violation Rate*,[10] namely, from Theorems 3.6 and 3.7 we obtain the following result.

Lemma 3.4: `CountingProVe` *can compute both a correct lower and upper bound, i.e., a correct confidence interval for the VR, with a probability $\geq (1 - 2^{-\beta t})$.*

[10] Notice that the same formulation holds for the complementary metrics (i.e., the *Safe Rate*).

CountingProVe is Polynomial

To conclude the analysis of our randomized-approximation algorithm, we discuss some additional requirements to guarantee that our approach runs in polynomial time. Let N denote the size of the instance. It is easy to see that by choosing both the timeout applied to each run of the exact count algorithm used in line 5 and the number m of points sampled in line 6 to be polynomially bounded in N then each iteration of the for loop is also polynomial in N. Moreover, if each split of the input area encoded by $\mathcal{X}$ performed in lines 7-10 is guaranteed to reduce the size of the instance by at least some fraction $\gamma \in (0, 1)$, then after $s = \Theta(\log N)$ splits the instance in the leaf ℓ has size $O(\frac{N}{2^s}) = O(1)$. Hence, we can exploit an exponential time formal verifier to solve the instance in the leaf ℓ, and the total time will be polynomial in N.

3.3.6 Experimental Results

In this subsection, we guide the reader to understand the importance and impact of this novel encoding for the verification problem of deep neural networks. In particular, in the first part of our experiments, we show how the problem's computational complexity impacts the run time of exact solvers, motivating the use of an approximation method to solve the problem efficiently. In the second part, we analyze a concrete case study, ACAS Xu [131], to explain why finding all possible unsafe configurations is crucial in a realistic safety-critical domain. All the data are collected on a commercial PC running Ubuntu 22.04 LTS equipped with Nvidia RTX 2070 Super and an Intel i7-9700k. In particular, for the exact counter, we rely as backend on the formal verification tool *BaB* [38] available on "NeuralVerification.jl" and developed as part of the work of [150]. While, as exact count for `CountingProVe`, we rely on *ProVe* [51] given its key feature of exploiting parallel computation on GPU. In our experiments with `CountingProVe`, we set $\beta = 0.02$ and $t = 350$ in order to obtain a correctness confidence level greater or equal to 99% (refer to Theorem 3.6).

Table 3.25 summarizes the results of our experiments, evaluating the proposed algorithms (i.e., the exact counter and `CountingProVe`) on different benchmarks. Our results show the advantage of using an approximation algorithm to obtain a provable estimate of the portion of the input space that violates a safety property. We discuss the results in detail below. The code used to collect the results and several additional experiments and discussions on the impact of different hyperparameters and backends for our approximation are available in the supplementary material (available here).

Scalability Experiments

In the first two blocks of Tab. 3.25, we report the experiments related to the scalability of the exact counters against our approximation method `CountingProVe`, showing how the #DNN-Verification problem becomes immediately infeasible, even for small DNNs. In more detail, we collect seven different random models (i.e., using random seeds) with different levels of violation rates for the same safety property, which consists of all the intervals of $\mathcal{X}$ in the range $[0, 1]$, and a postcondition $\mathcal{Y}$ that encodes a strictly positive output. In the first block, all the models have two hidden

Instance	Exact Count		CountingProVe (confidence ≥ 99%)		
	BaB				
	Violation Rate	Time	Interval confidence VR	Size	Time
Model_2_20	20.78%	234 min	[19.7%, 22.6%]	2.9%	42 min
Model_2_56	55.22%	196 min	[54.13%, 57.5%]	3.37%	34 min
Model_2_68	68.05%	210 min	[66.21%, 69.1%]	2.89%	45 min
Model_5_09	–	24 hrs	[8.42%, 13.2%]	4.78%	122 min
Model_5_50	–	24 hrs	[48.59%, 52.22%]	3.63%	124 min
Model_5_95	–	24 hrs	[91.73%, 96.23%]	4.49%	121 min
Model_10_76	–	24 hrs	[74.25%, 77.23%]	3.98%	300 min
ϕ_2 ACAS Xu_2.1	–	24 hrs	[0.45%, 5.01%]	4.56%	246 min
ϕ_2 ACAS Xu_2.3	–	24 hrs	[1.23%, 4.21%]	2.98%	241 min
ϕ_2 ACAS Xu_2.4	–	24 hrs	[0.74%, 3.43%]	2.68%	243 min
ϕ_2 ACAS Xu_2.5	–	24 hrs	[1.67%, 4.10%]	2.42%	240 min
ϕ_2 ACAS Xu_2.7	–	24 hrs	[2.35%, 5.22%]	2.87%	240 min

Table 3.8: Comparison of CountingProVe and exact counter on different benchmark setups. The first block shows the results on our benchmark properties, where every instance is in the form Model_ρ_ψ, where ρ is the size of the input space and ψ is the id of the specific DNN. The last block reports the results on the Acas Xu ϕ_2 benchmark. Full results on ϕ_2 and another property in the Appendix3.3.6.

layers of 32 nodes activated with $ReLU$ and two, five, and ten-dimensional input space, respectively. Our results show that for the models with two input neurons, the exact counter returns the violation rate in about 3.3 hours, while our approximation in less than an hour returns a provable tight ($\sim 3\%$) confidence interval of the input area that presents violations. Crucially, as the input space grows, the exact counters reach the timeout (fixed after 24 hours), failing to return an exact answer. CountingProVe, on the other hand, in about two hours, returns an accurate confidence interval for the violation rate, highlighting the substantial improvement in the scalability of this approach. In the supplementary material, we report additional experiments and discussions on the impact of different hyperparameters for the estimate of CountingProVe.

ACAS Xu Experiments

The ACAS Xu system is an airborne collision avoidance system for aircraft considered a well-known benchmark and a standard for formal verification of DNNs [125, 131]. It consists of 45 different models composed of an input layer taking five inputs, an output layer generating five outputs, and six hidden layers, each containing 50 neurons activated with $ReLU$. To show that the count of all the violation points can be extremely relevant in a safety-critical context, we focused on the property ϕ_2, on which, for 34 over 45 models, the property does not hold. In detail, the property ϕ_2 describes the scenario where if the intruder is distant and is significantly slower than the ownship, the score of a Clear of Conflict (COC) can never be maximal (more

detail are reported here [131]). We report in the last block of Tab. 3.25 the results of this evaluation. To understand the impact of the #DNN-VERIFICATION problem, let us focus, for example, on the *ACAS Xu_2.7* model. As reported in Tab. 3.25, `CountingProVe` returns a provable lower bound for the violation rate of at least a 2.35%. This means that assuming a 3-decimal-digit discretization, our approximation counts at least 23547 violation points compared to a formal verifier that returns a single counterexample (i.e., a single violation point). Note that a state-of-the-art verifier that returns only `SAT` or `UNSAT` does not provide any information about the possible amount of unsafe configurations considered by the property. Fig. 3.20 shows a 3d representation of the second property of the ACAS Xu benchmark (i.e., ϕ_2), comparing the possible outcome of a standard formal verifier, namely DNN-VERIFICATION, and the problem presented in this section #DNN-VERIFICATION.

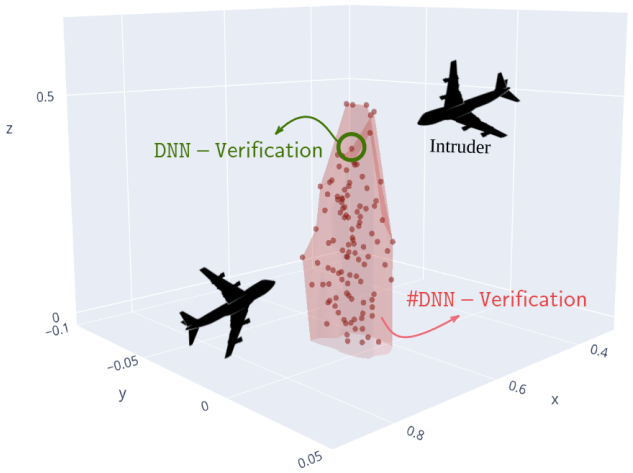


Fig. 3.20: Explanatory image of the possible impact of #DNN-VERIFICATION in safety-critical contexts. A standard verifier returns only a violation point (highlighted in the image with a green circle), limiting the interpretability of the results. In contrast, our approach paves the way to estimate the entire dangerous area (depicted in red in the figure).

Hyperparameters and Ablation Study

We report in this section the hyperparameters used to collect the previous results. Regarding the heuristic presented in the Alg. 7, we want to point out to the reader that a possible optimization is to perform a fixed number of s simplifications before calling the exact count method. In fact, as shown above, given the complexity of the problem, calling the exact count too frequently when input space is still considerably large typically results in a timeout, thus causing a waste of computation and time. For this reason, we decided to perform a fixed number of preliminary simplifications before calling the exact count. In more detail, we set $s = 17$ for the first row of Tab. 3.25, and $s = 45$ for the remaining part. The value s for the preliminary simplifications can be obtained assuming any discretization of the initial input space N, described by $\mathcal{X}$. Hence, relying on the considerations discussed in the Sec. 3.3.5

for the polynomiality of the approximation, we set $s = \lfloor \log N \rfloor - 1$ to ensure the termination of an exponential time exact counter. Moreover, to collect the data of Tab. 3.25 and 3.14, we set $\beta = 0.02, t = 350$ obtaining a confidence level of 99% (see theorem 3.6). Finally, regarding the number of samples to compute the median, we set $m = 1.5M$ for the scalability experiment of Sec. 3.3.6 and $m = 3M$ for the ACAS Xu experiments.

We performed additional experiments to highlight the impact of different hyperparameters on the quality of the estimate returned by `CountingProVe`. Crucially, as specified in section 3.3.5, the correctness of the algorithm is independent of the heuristic used by the algorithm. We now analyze the impact on the estimate of different parameters such as β, t, and m.

Experiments on Different β and t

Confidence	**Hyperparameters**		**`CountingProVe`**		
	β	t	Interval confidence VR	Size	Time
99%	0.02	350	[54.13%, 57.5%]	3.37%	34 min
	0.1	70	[51.36%, 59.17%]	7.81%	8 min
	1.5	5	[19.39%, 84.35%]	64.96%	32 sec
90%	0.02	170	[52.99%, 56.68%]	3.69%	17 min
	0.1	34	[51.37%, 58.93%]	7.55%	4 min
	1.5	3	[19.53%, 84.32%]	64.79%	18 sec
85%	0.02	137	[53.52%, 57.01%]	3.48%	14 min
	0.1	27	[51.47%, 58.67%]	7.2%	3 min
	1.5	2	[19.56%, 84.24%]	64.67%	12 sec

Table 3.9: Comparison of different hyperparameters for `CountingProVe` on *Model_2_56*. The true VR is equal to 55.22%.

Tab. 3.9 shows the comparison results between different hyperparameters for `CountingProVe`. In detail, all the experiments are performed on the same model "*Model_2_56*", using $s = 17$ preliminary simplifications before calling the exact count. Moreover, we use the same number of $m = 1.5M$ samples to compute the median value in the heuristic. We test three confidence levels at 85%, 90%, and 99%, respectively, setting three possible value pairs for β and t.

Regarding the impact of the error tolerance factor β, as expected, as this value increases, the confidence interval deteriorates.We justify this as the tolerance factor appears in the formula for calculating the violation at the end of each while loop ($2^{s-\beta} \cdot ExactCount$). Hence, a larger β strongly impacts the value of the estimate, thus also potentially deteriorating the lower and upper bounds. However, we want to emphasize once again as for any value tested at a high confidence level, the estimate returned by `CountingProVe` is correct, i.e., the lower bound does not overestimate, and the upper bound never underestimates the value returned by the true count (equals to 55.22%).

Interestingly, we note that by setting the same value for the error tolerance factor ($\beta = 0.02$), the estimate for the three confidence levels is quite similar. Our

approximation thus allows choosing the desired confidence level while obtaining a good estimate and potentially saving time. In fact, by choosing a confidence of 85%, we obtain an estimate very close to the best estimate obtained with 99% confidence, halving the computation time.

Impact of the m Samples in CountingProVe

Although the correctness of the approximation does not depend on the number of violation samples to compute the median value (as shown in Theorem 3.6), we performed an additional experiment to understand its impact on the quality of the estimation. The experiment was performed on the "*Model_2_56*" model with parameters $\beta = 0.02, t = 350, s = 17$. We report in Tab 3.10 the comparison of four different sample values, $500k, 1M, 1.5M, 3M$, and finally $5M$. As we can notice, increasing the sampling size m to find the violation points to compute the median leads to a more accurate estimate of the true violation rate. Intuitively, we obtain a (theoretically) higher probability of finding violation points in the input space by increasing the number of samples. Hence, the more violation points we randomly sample, the more information we obtain to compute an accurate median. However, using more samples results in more time to compute the median and, consequently, the confidence interval of the violation rate.

m **samples**	`CountingProVe` (confidence ≥ 99%)		
	Interval confidence VR	Size	Time
$500k$	[52.59%, 66.36%]	13.8%	13 min
$1M$	[51.45%, 57.2%]	5.74%	24 min
$1.5M$	[54.13%, 57.5%]	3.37%	34 min
$3M$	[53.41%, 56.63%]	3.21%	40 min
$5M$	[54.17%, 56.42%]	2.24%	60 min

Table 3.10: Comparison of different m for `CountingProVe`

Alternative Backends for `CountingProVe`

To show that the correctness of our approximation is independent of the backend chosen, we conducted additional experiments using *BaB* [38] and *NSVerify* [154] as the exact counters instead of *ProVe* [51] for the final count on the leaf in `CountingProVe`. To perform a fair analysis, given the stochastic nature of our approximation, we set the same seed for all methods tested, only changing some hyperparameters and network sizes. We report in Tab. 3.11 the results of our experiments. As expected, the resulting interval of confidence for the VR is the equivalent using any exact counters or hyperparameters in all the tests performed. In more detail, in the first two rows of Tab. 3.11, we use the same model (*Model_2_56*), only varying the hyperparameters for the confidence (i.e., β and t), and the number of preliminary splits s. We can notice as long as the network size is still small, the use of the GPU (used in *ProVe*) does not bring much benefit. In fact, there is a slight difference in the time to compute the interval of confidence of the VR in both tests with the model with only two input nodes. Moreover, while performing multiple

preliminary splits (s) can take more time, it also slightly improves the confidence interval. Crucially, notice that in the last two rows of Tab. 3.11 a little improvement of the interval confidence of VR w.r.t the results presented in Tab. 3.25 and 3.9.

Instance	**Hyperparameters**				**CountingProVe**			
	β	t	s	m	Backend	Interval VR	Size	Time
Model_2_56	0.1	70	15	1.5M	`BaB`	[51.36%, 59.17%]	7.81%	10 min
Model_2_56	0.1	70	15	1.5M	`ProVe`	[51.36%, 59.17%]	7.81%	8 min
Model_2_56	0.02	350	22	1.5M	`BaB`	[53.77%, 57.03%]	3.26%	60 min
Model_2_56	0.02	350	22	1.5M	`ProVe`	[53.77%, 57.03%]	3.26%	50 min
Model_5_95	0.02	350	79	1.5M	`BaB`	[92.42%, 96.22%]	3.8%	185 min
Model_5_95	0.02	350	79	1.5M	`NSVerify`	[92.42%, 96.22%]	3.8%	180 min
Model_5_95	0.02	350	79	1.5M	`ProVe`	[92.42%, 96.22%]	3.8%	150 min

Table 3.11: Comparison of different backends for `CountingProVe`

Regarding the last row of Tab. 3.11, we used a different model (*Model_5_95*) to test the scalability of other exact counters in combination with `CountingProVe`. The interesting thing to point out in this experiment is that *BaB* (or any different DNN-verification tool) used as a backend for the exact count on the same model results in timeout (i.e., after 24 hours, it does not return a result) as reported in Tab 3.25. However, using it as a backend in our approximation, in about 3 hours, can return a very tight confidence interval of the amount of the input space that presents violations. This shows that the intuition behind our approximation brings significant scalability improvements.

Finally, in this last experiment, we confirm what we mentioned above. As the network grows, having GPU support brings significant improvements in timing, as *ProVe*, in this experiment, saves 30 minutes of computation. Hence, this result motivates us to use it as the “default" backend for our approximation. Moreover, *ProVe* can verify any DNNs, i.e., with any activation function, which is not typically possible with any state-of-the-art DNN-verification tool. However, this experiment clearly shows that any verifier (perhaps that exploits GPUs) can be employed in `CountingProVe`, so potentially future improvements or new methods can be easily integrated into our approximation.

Discretization

It is crucial to point out that discretization is not a requirement of our counting approach. In fact, supposing to have a backend that can deal with continuous values, `CountingProve` would not require such discretization. Nevertheless, the discretization factor might be a parameter of the algorithm. To this end, we performed an analysis of the impact of this parameter, reporting the results in Tab. 3.12. Our experiments demonstrate that using a less fine-grained discretization produces less accurate outcomes, but it enhances the efficiency of the process in terms of time. In the previous results, we opted for a discretization value of 3 (i.e., 0.001) that provides a good balance between time and accuracy.

Rounding (decimal digit)	**CountingProVe** (confidence ≥ 99%) Interval Size	Time
1	64.8 ±2.2%	∼213 min
3	2.83 ±0.19%	∼242 min
5	2.2±0.38%	∼315 min

Table 3.12: Different discretization test on property ϕ_2 of ACAS Xu.

Single Check Verification

In Tab. 3.13, we provide the results of our additional experiments on the single check verification using α-β-$CROWN$[297, 291, 275]. We point out that this does not provide the same information as our proposed approach. Specifically, running a decision verifier multiple times does not provide information about the actual number of violations. Nevertheless, the UNSAT case can be interpreted as a counting result, where the answer is *zero* violations.

	Models				
	2_5	2_6	2_7	3_3	4_2
Result	SAT	SAT	SAT	UNSAT	UNSAT
Time (s)	8.2	8.1	8.12	74.23	85.1

Table 3.13: Single Check Verification on property ϕ_2 of ACAS Xu.

Full Experimental Results ACAS Xu

We report in Tab 3.14 the full results of comparing CountingProVe and the exact counter on ϕ_2 of the ACAS Xu benchmark discussed in Sec.3.3.6. We consider only the model for which the property ϕ_2 does not hold (i.e., the models that present at least one single input configuration that violate the safety property). From the results of Tab 3.14, we can see that our approximation returns a tight interval confidence (mean of 2.83%) of the VR for each model tested in about 4 hours. We want to underline that these obtained results do not exploit any particular optimization of our approximation, and therefore the times to compute these intervals can be greatly improved. For example, a simple optimization would compute the various t iterations in parallel, significantly reducing computation times.

Instance	**Exact Count**		**CountingProVe** (confidence $\geq$ 99%)		
	BaB				
	Violation Rate	Time	Interval confidence VR	Size	Time
ϕ_2 ACAS Xu_2.1	–	24 hrs	[0.45%, 5.01%]	4.56%	246 min
ϕ_2 ACAS Xu_2.2	–	24 hrs	[1.06%, 4.81%]	3.75%	246 min
ϕ_2 ACAS Xu_2.3	–	24 hrs	[1.23%, 4.21%]	2.98%	241 min
ϕ_2 ACAS Xu_2.4	–	24 hrs	[0.74%, 3.43%]	2.68%	243 min
ϕ_2 ACAS Xu_2.5	–	24 hrs	[1.67%, 4.10%]	2.42%	240 min
ϕ_2 ACAS Xu_2.6	–	24 hrs	[1.01%, 3.59%]	2.58%	248 min
ϕ_2 ACAS Xu_2.7	–	24 hrs	[2.35%, 5.22%]	2.87%	240 min
ϕ_2 ACAS Xu_2.8	–	24 hrs	[1.77%, 4.68%]	2.92%	248 min
ϕ_2 ACAS Xu_2.9	–	24 hrs	[0.18%, 2.77%]	2.59%	239 min
ϕ_2 ACAS Xu_3.1	–	24 hrs	[1.62%, 4.98%]	3.36%	242 min
ϕ_2 ACAS Xu_3.2	–	24 hrs	[0%, 2.50%]	2.5%	243 min
ϕ_2 ACAS Xu_3.3	–	24 hrs	[0%, 2.54%]	2.54%	245 min
ϕ_2 ACAS Xu_3.4	–	24 hrs	[0.26%, 3.08%]	2.82%	244 min
ϕ_2 ACAS Xu_3.5	–	24 hrs	[0.92%, 3.60%]	2.68%	244 min
ϕ_2 ACAS Xu_3.6	–	24 hrs	[1.71%, 4.48%]	2.77%	251 min
ϕ_2 ACAS Xu_3.7	–	24 hrs	[0.14%, 2.64%]	2.49%	213 min
ϕ_2 ACAS Xu_3.8	–	24 hrs	[0.75%, 3.28%]	2.54%	216 min
ϕ_2 ACAS Xu_3.9	–	24 hrs	[2.11%, 5.20%]	3.09%	242 min
ϕ_2 ACAS Xu_4.1	–	24 hrs	[0.33%, 3.04%]	2.71%	246 min
ϕ_2 ACAS Xu_4.3	–	24 hrs	[1.3%, 3.61%]	2.31%	243 min
ϕ_2 ACAS Xu_4.4	–	24 hrs	[0.79%, 3.57%]	2.79%	247 min
ϕ_2 ACAS Xu_4.5	–	24 hrs	[0.71%, 4.03%]	3.33%	240 min
ϕ_2 ACAS Xu_4.6	–	24 hrs	[1.65%, 4.72%]	3.08%	244 min
ϕ_2 ACAS Xu_4.7	–	24 hrs	[1.67%, 4.33%]	2.66%	248 min
ϕ_2 ACAS Xu_4.8	–	24 hrs	[1.68%, 4.17%]	2.49%	241 min
ϕ_2 ACAS Xu_4.9	–	24 hrs	[0.10%, 2.61%]	2.51%	247 min
ϕ_2 ACAS Xu_5.1	–	24 hrs	[1.06%, 3.76%]	2.7%	240 min
ϕ_2 ACAS Xu_5.2	–	24 hrs	[0.86%, 3.58%]	2.72%	248 min
ϕ_2 ACAS Xu_5.4	–	24 hrs	[0.75%, 3.25%]	2.5%	239 min
ϕ_2 ACAS Xu_5.5	–	24 hrs	[1.66%, 4.35%]	2.68%	247 min
ϕ_2 ACAS Xu_5.6	–	24 hrs	[1.81%, 4.45%]	2.64%	240 min
ϕ_2 ACAS Xu_5.7	–	24 hrs	[1.75%, 5.15%]	3.40%	246 min
ϕ_2 ACAS Xu_5.8	–	24 hrs	[1.96%, 4.65%]	2.70%	241 min
ϕ_2 ACAS Xu_5.9	–	24 hrs	[1.62%, 4.40%]	2.77%	241 min
			Mean	2.83%	242 min

Table 3.14: Comparison of CountingProVe and exact counters on different benchmark setups.

Experiments on a Different Property

To validate the correctness of our approximation, we also performed a final experiment on property ϕ_3 of the Acas Xu benchmark. In particular, this property encodes a scenario in which if the intruder is directly ahead and is moving towards the ownship, the score for COC will not be minimal (we refer to [131] for further details). This property is particularly interesting as it holds for all 45 models, i.e., we expect a 0% of violation rate for each model tested. For simplicity, in Tab. 3.15 we report only the first 5 models since the results were very similar for all the DNNs tested. As expected, we empirically confirmed that also for this particular situation, the lower and upper bounds computed with `CountingProVe` never overestimate and underestimate, respectively, the true value of the violation rate, in this case, 0%.

Instance	**`CountingProVe`** (confidence $\geq$ 99%)		
	Interval confidence VR	Size	Time
ϕ_3 ACAS Xu_1.1	[0%, 2.26%]	2.26%	215 min
ϕ_3 ACAS Xu_1.2	[0%, 2.88%]	2.88%	216 min
ϕ_3 ACAS Xu_1.3	[0%, 2.30%]	2.30%	215 min
ϕ_3 ACAS Xu_1.4	[0%, 2.47%]	2.47%	218 min
ϕ_3 ACAS Xu_1.5	[0%, 2.48%]	2.48%	214 min

Table 3.15: `CountingProVe` on ACAS Xu ϕ_3 property

Summary. In this section, we first introduced the #DNN-Verification, the problem of counting the number of input configurations that generate a violation of a given safety property. We analyzed the complexity of this problem, proving the #P-hardness and highlighting why it is relevant for the community. Furthermore, we proposed an exact count approach that, however, inherits the limitations of the formal verification tool exploited as a backend and struggles to scale on real-world problems. Crucially, we presented an alternative probabilistic approach, `CountingProVe`, which provides an approximated solution with formal guarantees on the confidence interval. However, the backend used in `CountingProVe` still needs to perform exact verification over a subset of the input space under analysis. Consequently, the overall efficiency of our approach strongly depends on the performance of these exact solvers. To address this limitation, in the next section, we introduce a method designed to improve their efficiency and, in turn, enhance the scalability of our solution to the #DNN-Verification problem.

3.4 Scaling #DNN-Verification Tools with Efficient Bound Propagation and Parallel Computing

The main solutions of formal verification state-of-the-art tools for both DNN-Verification and #DNN-Verification, presented in the previous sections, are strongly based on the Branch-And-Bound approach [38]. However, unlike the methods to solve the DNN-Verification problem, which potentially explore only part of the tree generated through BaB, since they seek for a single counterexample, to solve in a sound and complete fashion the #DNN-Verification problem, we have to explore every single node, which exponentially increases the complexity, resulting in a prohibitive computational demand.

Hence, inspired by the recent BaB-based solutions employed to solve efficiently the DNN-Verification [275], in this work, we first investigate possible optimizations to solve the #DNN-Verification in an exact fashion. Moreover, we employ these optimizations to `CountingProVe`, the approximate solution described in the previous section, to assess the scalability and efficiency improvements. We argue that by integrating similar optimization of standard FV tools on previously proposed exact and approximate solutions for the #DNN-Verification problem, we are able to enhance the scalability and efficiency of these solutions. More specifically, our intuitions rely on the fact that every time we develop a new level of the BaB tree, instead of computing the result on each node iteratively, we can employ the power computation of GPUs to check multiple scenarios simultaneously. In addition, in order to maintain an optimal branching tree size, we can employ in our optimization the *symbolic linear relaxation* (SLR) [273], one of the most recent and efficient techniques to compute a tight bound propagation through the DNN.

The empirical evaluation on standard FV benchmarks as ACAS xu [125] and on realistic robotic DRL mapless navigation domain confirm our hypothesis. In particular, we empirically show the improvement in scalability even in complex robotic scenarios, where previous approaches for the #DNN-Verification problem struggled to scale, making this new type of verification applicable also in different robotic real-world applications.

Summarizing, the main contributions of this section are the following:

- we proposed an efficient bound propagation method based on BaB and SLR for solving the #DNN-Verification efficiently.
- inspired by existing FV tools that exploit GPU to speed up the verification process [275] we employ the parallel computation of each BaB layer to enhance the performance of existing exact count approaches.
- we empirically show the effectiveness and the scalability improvement of the solution proposed on standard FV benchmark (ACAS xu) and realistic robotic scenarios.

3.4.1 Background

Our solution exploits two main components common to most recent state-of-the-art verifiers: interval analysis with BaB approach that integrates symbolic linear relaxation to compute tight output reachable sets and parallel computation [273, 275]. In the following, we briefly recall these strategies.

Efficient Bound Estimation via Symbolic Interval Propagation and Linear Relaxation

In Fig. 3.21, we report a representation of the reachability analysis combined with a branching method called *iterative refinement*. In particular, the idea is to exploit the fact that any neural network with a finite number of layers is Lipschitz continuous. Hence, leveraging the fact that the dependency error for Lipschitz continuous functions decreases as the width of intervals decreases, the iterative refinement process allows us to bisect the input interval by evenly dividing the interval into the union of two consecutive sub-intervals obtaining more tight reachable sets w.r.t. the naive propagation [274].

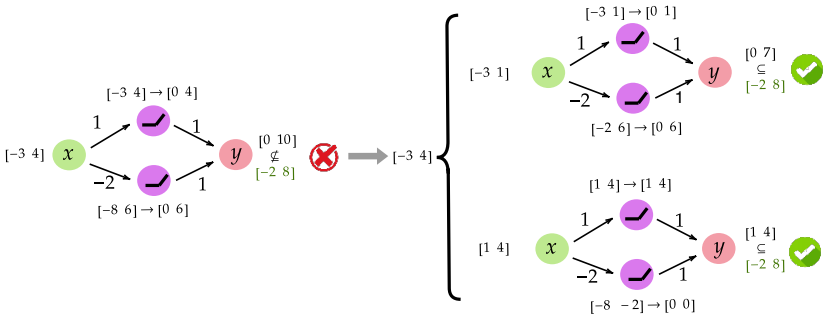


Fig. 3.21: Explanatory image of reachability analysis combined with branching method called *iterative refinement*[274]. Exploiting Moore's interval algebra [196], the lower and the upper bounds of each interval are propagated through the DNN layer-by-layer. In the left part, naive interval analysis produces a very large overestimation of the reachable set. Using input split refinements (right part) we are able to reduce the over-approximation and provably verify the property.

However, another source of over-approximation is given by the fact that naive interval propagation–even when combined with iterative refinement– does not take into account all the inter-dependencies with the previous layers. To this end, [274] proposes the symbolic interval propagation (SIP) as a novel solution to preserve as much dependency information as possible while propagating the bounds through the DNNs layers. In particular, simarly to what we have seen in the previous chapters, Wang et al. [274] consider DNNs with ReLU activation functions, and they seek to maintain linear equations for all the propagation, simplifying the non-linear nature of the neural networks and significantly speeding up the verification process. However, ReLU nodes can demonstrate non-linear behavior for a given input interval and when the interval in input to the ReLU node produces a result with a negative lower bound and a positive upper bound, a concretization process is required, thus losing all the inter-dependences preserved up to that specific node. To address such an issue, [273] proposed the symbolic linear relaxation of ReLU nodes, which allows in these scenarios to partially keep the input dependencies during interval propagation by maintaining symbolic equations (the same used in the linearization of ReLU layers presented in Sec. 3.2) in the form:

$$z = \left[\frac{u}{u-\ell}Eq_\ell,\ \frac{u}{u-\ell}(Eq_u - \ell)\right] \tag{3.17}$$

where z is the ReLU node, and ℓ, u denote the concrete lower and upper bounds for Eq_ℓ and Eq_u, respectively. More specifically, in Fig. 3.22, we report an explanatory example of interval propagation using SIP and SLR. Focusing on the left part of the image and considering the upper ReLU node, we have to perform a concretization since, considering the input bounds, we have a possible negative lower and a positive upper bound. This process leads to the loss of all the input dependencies preserved until that moment. In contrast, on the right part of the figure, using Eq. 3.17 on the same ReLU node, we are able to partially preserve the information and thus obtain a tighter output reachable set.

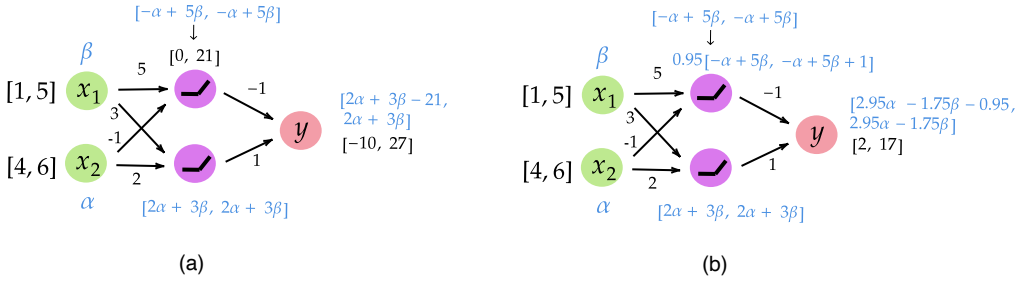


Fig. 3.22: Explanatory image of (a) Symbolic interval propagation proposed in [274] and (b) Symbolic linear relaxation [273].

3.4.2 Efficient #DNN-Verification via Symbolic Linear Relaxation and Parallel Computing

In this subsection, we present our main contributions. Considering the fact that the #DNN-Verification problem is #P-hard, the time complexity of the algorithm for exact counting the violations in the property's domain deteriorates rapidly as the instance input size of the network grows. In fact, even moderately small networks with only five input nodes become unmanageable using this approach. In particular, what makes the problem so complex is having to solve an instance of the DNN-Verification problem on each node of the BaB. As the size of the input increases, we have a larger number of potential splits to perform, significantly increasing the complexity of the problem. In fact, suppose to consider a network of only 5 input nodes, where on each dimension, we perform 12 splits to be able to get an exact count of the *violation rate*, thus getting $5 \cdot 2^{12}$ nodes to verify. Assuming that each node of the BaB requires an average time of 5 seconds to get an answer, it would require about $5 \cdot 2^{12} \cdot 5s = 102400s \sim 28$ hours in order to compute the total count of violations in the property's domain.

To address this challenging problem, inspired by the existing optimizations of *search-reachability* methods for the DNN-Verification problem, we propose to combine the symbolic linear relaxation [273] to reduce the number of splits required during the computation, and the parallel computation to make the #DNN-Verification problem manageable. We implement our exact count for ReLU DNNs,

Algorithm 8: `Exact Count`

Input: $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$
Output: Violation Rate (VR)
$VR \leftarrow 0$, unknown $\leftarrow \mathcal{X}$
while unknown $\neq \emptyset$ **do**
  $\mathcal{X}' \leftarrow$ `Pop`(unknown)
  $\mathcal{R} \leftarrow$ `ComputeReachableSet`$(\mathcal{X}', f)$
  if $\mathcal{R} \subseteq \mathcal{Y}$ **then**
    $VR \leftarrow VR + |\mathcal{R}|$
  else if $\mathcal{R} \cap \mathcal{Y} = \emptyset$ **then**
    unknown $\leftarrow$ unknown $\setminus \mathcal{X}'$
  else
    unknown $\leftarrow$ unknown $\cup$ `IterativeRefinement`$(\mathcal{X}')$
return VR

even though our approach is easily extendable to different DNNs such as Tanh or Sigmoid, using similar solutions proposed in [100] to deal with different non-linear activation functions.

We report in Alg. 16 the complete pipeline of our exact count method. We start by considering the entire property's domain $\mathcal{X}$ which encodes the property's precondition, and we check for the negated property's postcondition $\mathcal{Y}$ (as discussed in Sec.2). Hence, for each sub-portion $\mathcal{X}'$ of the original input space under consideration, we use the method `ComputeReachableSet` to compute the output reachable set of that portion. In detail, for each node of each hidden layer, we store two linear equations, one for the lower and one for the upper bound of the interval. Once we reach a ReLU node, in case concretization is needed, we use the symbolic linear relaxation, i.e., Eq. 3.17, described above, to preserve the interdependence with the previous layer, obtaining tight output bounds and thus speeding up the verification process. At the end of the computation, we check if the corresponding output reachable set of $\mathcal{R}(\mathcal{X}, f)$ lies in the undesired reachable set $\mathcal{Y}$, using Moore's interval algebra [196] and, in particular, this relation:

$$\mathcal{R}(\mathcal{X}, \mathcal{N}) \subseteq \mathcal{Y} \iff (\mathcal{R}_\ell \geq \mathcal{Y}_\ell \wedge \mathcal{R}_u \leq \mathcal{Y}_u)$$

where $\mathcal{R}_\ell$, ,$\mathcal{Y}_\ell$ are the lower bound and $\mathcal{R}_u$, $\mathcal{Y}_u$ are the upper bound of the reachable sets $\mathcal{R}(\mathcal{X}, f)$ and $\mathcal{Y}$, respectively. Since we start verifying the negation of the postcondition, if the $\mathcal{R}(\mathcal{X}, f) \subseteq \mathcal{Y}$ this implies that all $\mathcal{X}$ is unsafe, that is, there exists at least one and possibly an infinite number of violation points.[11] On the other hand, if $\mathcal{R}(\mathcal{X}, f) \cap \mathcal{Y} = \emptyset$, we can state that there are no violation points in $\mathcal{X}$, thus the original safety property holds. Finally, if we are in the case where the two reachable sets overlap only on one side, i.e., a situation in which $(\mathcal{R}_\ell < \mathcal{Y}_\ell \wedge \mathcal{R}_u \leq \mathcal{Y}_u)$ or the symmetric case, $(\mathcal{R}_\ell \geq \mathcal{Y}_\ell \wedge \mathcal{R}_u > \mathcal{Y}_u)$, we cannot state whether the property is respected or not, and we have to proceed with the iterative refinement process [274].

[11] Since we operate with a continuous input space hence we could have an infinite number of violations.

This process is repeated until, in all the partitions of the input space created with the iterative refinement, we have complete safety or violation.

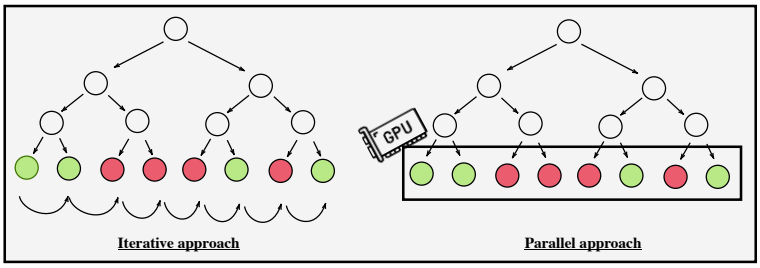


Fig. 3.23: Left: the original approach proposed in the previous section for obtain the exact count for the *#DNN-Verification* problem. Each node of the BaB is explored iteratively, thus resulting in poor sample efficiency and scalability as the size of the tree grows. Right: part of the optimization proposed in this work. Exploiting GPUs acceleration we can verify each layer of the BaB in parallel, thus enhancing the performance of exact count tools.

Even though the application of SLR improves the speed up of the verification process, the main difference with the DNN-Verification problem is that we have to check every single node of the BaB tree. Hence, an iterative approach like the one represented in the left part of Fig. 3.23, can be very inefficient as the number of branches increases. In contrast, the idea is to fully exploit the power of modern accelerators, i.e., GPUs, to compute in a parallel fashion the result of the FV verification of each layer of the BaB tree, as previously shown in Sec. 3.2 and reported in the right Fig. 3.23. In particular, each node located in the same layer t of the BaB tree represents a separate sub-instance to analyze and thus can be threaded independently. We represent the total number of sub-instances at the same layer t with a value n and the cardinality of each sub-instance with k. Thus, each layer t of the BaB tree can be represented using a matrix of size $(n, k, 2)$, where the number 2 represents the lower and upper bound of each of the n intervals of cardinality k to be verified. To clarify, referring to Fig.3.23 and assuming we want to parallel verify the last level of the BaB tree shown on the right side of the image, this would be encoded with a matrix $(8, k, 2)$, with k the dimension of the input space.

The main difficulties in implementing SLR on GPUs in CUDA are related to the management of memory cost. When comparing SLR with naive propagation, we can notice that the propagation of the coefficients of two equations for SLR, as compared to the propagation of the upper and lower bound values for naive, requires more memory in proportion to the number of input nodes in the network. In fact, given m the maximum layer size and k the number of input nodes, naive propagation requires $m \cdot 2$ floats in memory to propagate the upper and lower bound on each node in the layer, while SLR requires $m \cdot k \cdot 2$ floats in memory to propagate the coefficients associated with each input node through the DNN. Since both the global and shared memory of each block in the GPU is generally limited, performing the verification on DNNs with either a considerable number of input nodes or particularly large hidden layers can be challenging on GPUs. To address this issue, we implement a parallel

version of symbolic linear relaxation via a CUDA kernel, which concurrently runs interval analysis on each of the n sub-instances by assigning each one to a separate CUDA thread. In order to further improve the computational efficiency, we load the lower and upper equations for symbolic linear relaxation onto the shared memory, which provides faster access time compared to the global memory, vectorizing every data structure into one-dimensional arrays. Moreover, to address the memory limitation discussed above, we maintain a small number of threads per block within limits allowed by the ratio of shared memory and equation size for the network we want to verify. The resulting output bounds for each of the n sub-instances verified are then saved on the specific indices of an output vector and analyzed to compute the next layer $t + 1$ of the BaB tree.

3.4.3 Empirical Evaluation

In this subsection, we empirically analyze the improvement in scalability and efficiency of the methods proposed in this section. In particular, our goal is to show that the combination of efficient techniques employed for standard FV can be extended for the #DNN-Verification, enhancing the computational efficiency of existing tools and thus allowing the adoption of these methodologies also in realistic robotic applications.

To this end, we applied the optimization discussed above on `ProVe`[51] and we refer to this new method as `ProVe_SLR` and `CountingProVe` (presented in the previous section) referred as `CountingProVe_SLR`. Moreover, we compare our methods against the `Exact Count` presented in Alg. 16, which uses naive interval propagation and no parallelization. All the data have been collected on a commercial PC running Ubuntu 22.04 LTS equipped with Intel i5-13600KF and Nvidia GeForce RTX 4070 Ti, reporting for each method the violation rate and the computation time.

We show the results of our experiments on three different case studies. In the first block of Tab. 3.25, we analyze ACAS Xu[125, 131], a realistic safety-critical domain in which finding all possible unsafe configurations is crucial. In more detail, ACAS Xu is an airborne collision avoidance system for aircraft, a well-known benchmark for FV of DNNs. It consists of 45 models with five input nodes, five output nodes, and six hidden layers containing 50 $ReLU$ nodes each. In our experiments, we compare the performance of each method on three different models for which we have a `SAT` answer (i.e., at least a violation point) on the property ϕ_2, which describes a scenario where, if the intruder is distant and significantly slower than the ownship, the score of a Clear of Conflict (CoC) can never be maximal.

We then analyze a set of DRL mapless navigation models [8, 168]. In particular, in the second block of Tab.3.25, we consider an input space composed of nine nodes, two hidden layers of 16 nodes activated with ReLU, and three output nodes that encode discrete actions that the agent can perform in the environment.

Finally, in order to test the scalability, in the last block of Tab. 3.25, we increase the complexity and test a model with 13 inputs representing more dense laser scans, two hidden layers of 64 ReLU nodes, and six output nodes that encode, once again, discrete actions for the robot. The safety properties tested in both of these mapless navigation scenarios are behavioral properties. We provide the reader with a pictorial representation of the property tested in Fig. 4.11.

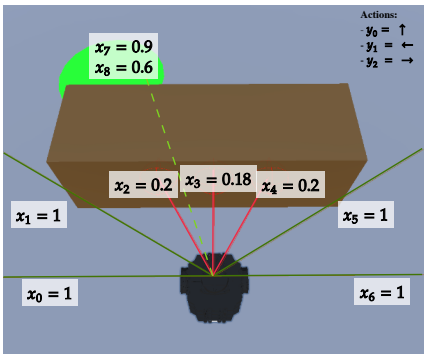


Fig. 3.24: Explanatory image of the safety property tested on different models in the second block of Tab 3.25. Each input $x_0, \ldots, x_6$ represents a lidar scan value between $[0, 1]$, while x_7, x_8 are heading and distance from the goal, respectively. Hence, the safety property ensures that in this situation, a forward movement (y_0) is not chosen by the agent.

Focusing on the first block in Tab. 3.25, due to the scalability issues discussed in Sec.3.3, the `Exact Count` method reaches the timeout (fixed after 24 hours), failing to return an exact answer for all the tests performed.

On the other hand, `ProVe_SLR` is able to return the exact VR in about 8 hours and 34 minutes (mean computation time), which corresponds to a $\sim$ 64.3% mean time reduction to compute the VR rate. The time required by `ProVe_SLR` is still considerable, primarily due to the need to write partial results to disk during the verification process to address memory constraints. We specify that with more powerful hardware, the verification times can be improved further, but we argue that it is interesting to show how the optimizations proposed in this work allow the use of these verifiers in safety-critical robotic scenarios, even using commercial hardware.

A slight minor improvement is also obtained when we use the `CountingProVe_SLR` approximation that uses `ProVe_SLR` as backend over the original `CountingProVe`[173]. Recalling the intuitions of this latter approximation provided in Sec. 3.3, given the

	Exact methods				**Approximations (confidence >99%)**			
Instance	`Exact Count`		`ProVe_SLR`		`CountingProVe`		`CountingProVe_SLR`	
	VR	Time	VR	Time	Lower Bound VR	Time	Lower Bound VR	Time
ϕ_2 ACAS Xu_4.3	✗	> 24h	1.43%	**8h46m**	1.32%	2h4m	1.28%	**45m**
ϕ_2 ACAS Xu_4.9	✗	> 24h	0.15%	**12h21m**	0.093%	2h23m	0.10%	**59m**
ϕ_2 ACAS Xu_5.8	✗	> 24h	2.2%	**4h35m**	1.96%	2h10m	1.76%	**51m**
Model_9_1	✗	> 24h	27.98%	**3h46m**	26.47%	2h17m	25.98%	**42m**
Model_9_2	✗	> 24h	20.88%	**4h20m**	19.73%	2h31m	19.95%	**48m**
Model_9_3	✗	> 24h	25.62%	**3h12m**	24.64%	2h2m	24.26%	**47m**
Model_13_64	✗	> 24h	22.13%	**5h41m**	20.99%	3h21m	21.24%	**1h30m**

Table 3.16: Comparison on different case studies of `Exact Count` without parallelization, `ProVe_SLR` that combines parallelization and SLR, `CountingProVe` and `CountingProVe_SLR` which exploits the efficiency improvements of `ProVe_SLR` as backend. The first block shows the results on the ACAS Xu ϕ_2 FV benchmark. The second and third blocks report the results of the robotic mapless navigation scenario.

improvement in the scalability of ProVe_SLR, we are able to reduce the number of splits required before employing the exact count on the leaf, thus reducing the computational time required to compute the lower bound of the VR. In particular, focusing on the results, given the randomized nature of CountingProVe, the VR reported by this latter and CountingProVe_SLR is slightly different. Nonetheless, both the results are correct as we obtain a lower bound of the real violation rate computed by ProVe_SLR. The significant difference in this first block is that we obtained a $\sim 61.4\%$ mean reduction in the computation time between CountingProVe and CountingProVe_SLR.

Regarding the experiments of robotic mapless navigation, also in this realistic scenario, we notice a substantial improvement in computation time when applying the combination of SLR and parallel computing. Crucially, in both these sets of experiments, we obtain a mean computational improvement of $\sim 80\%$ when using ProVe_SLR over a naive exact count approach and a $\sim 61.6\%$ when employing CountingProVe_SLR over CountingProVe.

Summary. In this section, we presented an efficient optimizations to the existing solutions for the #DNN-Verification problem. Crucially, we show that the combination of symbolic linear relaxation for tight output reachability sets estimation and parallel computation on GPUs enhances the scalability and performance of existing #DNN-Verification tools. This optimization paves the way for the application of different types of FV in very complex, realistic robotic scenarios, such as mapless navigation. We show how naive sample-based approximations can fail to capture a minimal fraction of the violation rate in the property's domain, emphasizing the necessity for more sophisticated approximation methods for counting violation points such as CountingProVe_SLR.

Nonetheless, solutions to the #DNN-Verification problem allow for estimating the probability that a DNN violates a given property, but they do not provide information on the actual input configurations that are safe or violations for the property of interest. The knowledge of the distribution of safe and unsafe areas in the input space is a key element to devise approaches that can enhance the safety of DNNs, e.g., by patching unsafe areas through re-training. To address this aspect in the next section, we introduce the AllDNN-Verification problem.

3.5 Enumerating Safe Regions in Deep Neural Networks with Provable Probabilistic Guarantees

For the sake of clarity, in this first part of the section, we recall the existing FV approaches and related key concepts on which our approach is based. In contrast to the standard robustness and adversarial attack literature [41, 158, 299], FV of DNNs seeks formal guarantees on the safety aspect of the neural network given a specific input domain of interest. Broadly speaking, if a DNN-Verification tool states that the region is provably verified, this implies that there are no adversarial examples – violation points – in that region. We recall in the next section the formal definition of the satisfiability problem for DNN-Verification [131].

DNN-Verification

In the DNN-Verification, we have as input a tuple $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$, where f is a DNN, $\mathcal{X}$ is precondition on the input, and $\mathcal{Y}$ a postcondition on the output. In particular, $\mathcal{X}$ denotes a particular input domain or region for which we require a particular postcondition $\mathcal{Y}$ to hold on the output of f. Since we are interested in discovering a possible counterexample, $\mathcal{Y}$ typically encodes the negation of the desired behavior for f. Hence, the possible outcomes are `SAT` if there exists an input configuration that lies in the input domain of interest, satisfying the predicate $\mathcal{X}$, and for which the DNN satisfies the postcondition $\mathcal{Y}$, i.e., at least one violation exists in the considered area, `UNSAT` otherwise.

To provide the reader with a better intuition, we discuss a toy example.

Example 2: (DNN-Verification) Suppose we want to verify that for the toy DNN f depicted in Fig. 3.25 given an input vector $\boldsymbol{x} = (x_1, x_2) \in [0, 1] \times [0, 1]$, the resulting output should always be $y \geq 0$. We define $\mathcal{X}$ as the predicate on the input vector $\boldsymbol{x} = (x_1, x_2)$ which is true iff $\boldsymbol{x} \in [0, 1] \times [0, 1]$, and $\mathcal{Y}$ as the predicate on the output y which is true iff $y = f(\boldsymbol{x}) < 0$, that is, we set $\mathcal{Y}$ to be the negation of our desired property. As reported in Fig. 3.25, given the vector $\boldsymbol{x} = [1, 0]^T$ we obtain $y < 0$, hence the verification tool returns a `SAT` answer, meaning that a specific counterexample exists and thus the original safety property does not hold.

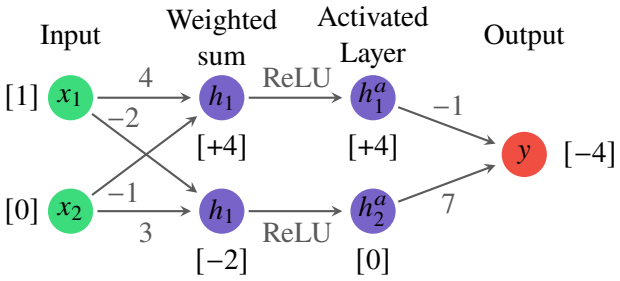


Fig. 3.25: A counterexample for a toy DNN-Verification problem.

#DNN-VERIFICATION problem

Despite the provable guarantees and the advancement that formal verification tools have shown in recent years [150], the binary nature of the result of the DNN-VERIFICATION problem may hide additional information about the safety aspect of the DNNs. To address this limitation in the previous section we discuss the #DNN-VERIFICATION, i.e., the extension of the decision problem to its counting version. In this problem, the input is the same as the decision version, but we denote as $\Gamma(\mathcal{T})$ the set of *all* the input configurations for f satisfying the property defined by $\mathcal{X}$ and $\mathcal{Y}$, i.e.

$$\Gamma(\mathcal{T}) = \left\{ \boldsymbol{x} \;\middle|\; \mathcal{X}(\boldsymbol{x}) \wedge \mathcal{Y}(f(\boldsymbol{x})) \right\} \tag{3.18}$$

Then, the #DNN-VERIFICATION consists of computing $|\Gamma(\mathcal{T})|$.

The approach (reported in Fig.3.26) solves the problem in a sound and complete fashion where any state-of-the-art FV tool for the decision problem can be employed to check each node of the Branch-and-Bound [38] tree recursively.

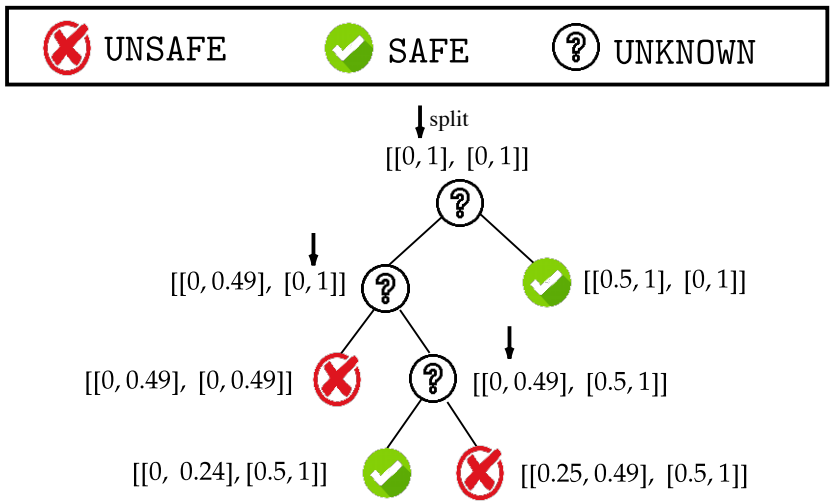


Fig. 3.26: Explanatory image execution of exact count for a particular f and safety property.

In detail, each node produces a partition of the input space into two equal parts as long as it contains both a point that violates the property and a point that satisfies it. The leaves of this recursion tree procedure correspond to partitioning the input space into parts where we have either complete violations or safety. Hence, the provable count of the safe areas is easily computable by summing up the cardinality of the subinput spaces in the leaves that present complete safety. Since our setting is in the continuum, the number of points in any non-empty set is infinite. Hence, we consider the cardinality as a proxy for the volume of the corresponding set. Nonetheless, if we assume some discretization of the space (to the maximum resolution allowed by the machine precision), $\Gamma(\mathcal{T})$ becomes a finite countable set.

Clearly, by using this method, it is possible to exactly count (and even to enumerate) the safe points. However, due to the necessity of solving a DNN-VERIFICATION

instance at each node (an intractable problem that might require exponential time), this approach becomes soon unfeasible and struggles to scale on real-world scenarios. In fact, it turns out that under standard complexity assumption, no efficient and scalable approach can return the exact set of areas in which a DNN is provably safe (as detailed in the next section).

To address this concern, after formally defining the AllDNN-Verification problem and its complexity, we propose first of all a relaxation of the problem, and subsequently, an approximate method that exploits the analysis of underestimated output reachable sets obtained using *statistical prediction of tolerance limits* [286, 216] providing a tight underapproximation of the safe areas with strong probabilistic guarantees.

The AllDNN-Verification Problem

The AllDNN-Verification problem asks for the set of all the safe points for a particular tuple $\langle f, \mathcal{X}, \mathcal{Y} \rangle$. Formally:

Definition 3.9 (AllDNN-Verification Problem):
Input: *A tuple* $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$.
Output: *the set of safe points* $\Gamma(\mathcal{T})$, *as given in (3.18).*

Considering the example of Fig. 3.26, solving the AllDNN-Verification problem for the safe areas consists in returning the set:

$$\Gamma = \left\{ \Big[[0.5, 1] \times [0, 1] \Big] \cup \Big[[0, 0.24] \times [0.5, 1] \Big] \right\}.$$

Hardness of AllDNN-Verification

From the #P-hardness of the #DNN-Verification problem proved in the previous section and the fact that exact enumeration also provides exact counting, it immediately follows that the AllDNN-Verification is #P-hard, which essentially states that no polynomial algorithm is expected to exist for the AllDNN-Verification problem.

3.5.1 ϵ-ProVe: a Provable (Probabilistic) Approach

In view of the structural scalability issue of any solution to the AllDNN-Verification problem, due to its #P-hardness, we propose to resort to an approximate solution. More precisely, we define the following approximate version of the problem:

Definition 3.10 (*ϵ-Rectilinear Under-Approximation of safe areas for DNN (ϵ-RUA-DNN)*):
Input: *A tuple* $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$.
Output: *a family* $\mathcal{R} = \{r_1, \ldots, r_m\}$ *of disjoint rectilinear* ϵ*-bounded hyperrectangles such that* $\bigcup_i r_i \subseteq \Gamma(\mathcal{T})$ *and* $|\Gamma(\mathcal{T}) \setminus \bigcup_i r_i|$ *is minimum.*

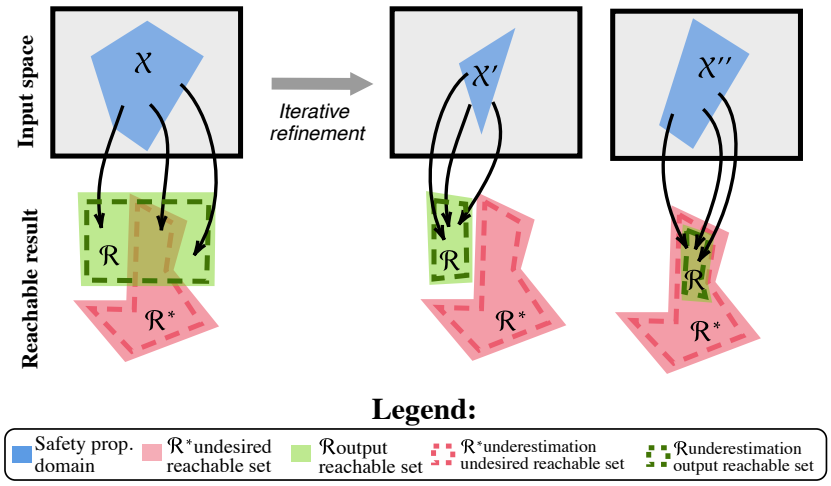


Fig. 3.27: Explanatory image of how to exploit reachable set result for solving the AllDNN-Verification problem.

A *rectilinear ϵ-bounded hyperrectangle* is defined as the cartesian product of intervals of size at least ϵ. Moreover, for $\epsilon > 0$, we say that a rectilinear hyperrectangle $r = \times_i [\ell_i, u_i]$ is *ϵ-aligned* if for each i, both extremes ℓ_i and u_i are a multiple of ϵ.

The rationale behind this new formulation of the problem is twofold: on the one hand, we are relaxing the request for the exact enumeration of safe points—in fact, as argued in [130] due to the #P-hardness proof (from #3-SAT), even guaranteeing a constant approximation to $|\Gamma(\mathcal{T})|$ by a deterministic polynomial time algorithm is not possible unless $P = NP$; on the other hand, we are requiring that the output is more concisely representable by means of hyperrectangles of *some significant* size.

Note that for $\epsilon \to 0$, *ϵ-RUA-DNN* and AllDNN-Verification become the same problem. More generally, whenever the solution $\Gamma(\mathcal{T})$ to an instance $\mathcal{T}$ of AllDNN-Verification can be partitioned into a collection of rectilinear ϵ-bounded hyperrectangles, $\Gamma(\mathcal{T})$ can be attained by an optimal solution for the *ϵ-RUA-DNN*. This allows us to tackle the AllDNN-Verification problem via an efficient approach with strong probabilistic approximation guarantees to solve the *ϵ-RUA-DNN* problem.

Our method is based on two main concepts: the analysis of an underestimated output reachable set with probabilistic guarantees and the *iterative refinement* approach [274]. In particular, in Fig. 3.27 we report a schematic representation of the approach that can be set up through reachable set analysis. Let us consider a possible domain for the safety property, i.e., the polygon highlighted in light blue in the upper left corner of Fig. 3.27.

Suppose that the undesired output reachable set is the one highlighted in red called $\mathcal{R}^*$ in the bottom left part of the image, i.e., this set describes all the unsafe outcomes the DNN should never output starting from $\mathcal{X}$. Hence, in order to formally verify that the network respects the desired safety property, the output reachable set computed from the domain of the property (i.e., the green $\mathcal{R}$ area of the left side of the image) should have an empty intersection with the undesired reachable set (the

red one). If this condition is not respected, as, e.g., in the left part of the figure, then there exists at least an input configuration for which the property is not respected.

To find all the portions of the property's domain where either the undesired reachable set and the output reachable set are disjoint, i.e., $\mathcal{R}^* \bigcap \mathcal{R} = \emptyset$, or, dually, discover the unsafe areas where the condition $\mathcal{R} \subseteq \mathcal{R}^*$ holds (as shown in the right part of Fig. 3.27) we can exploit the *iterative refinement* approach [274]. However, given the nonlinear nature of DNNs, computing the exact output reachable set is infeasible. To address this issue, the reachable set is typically over-approximated, thereby ensuring the soundness of the result. In this vein, [294] proposed an enumeration approach based on an over-approximation of the reachable set to compute the set of unsafe regions in the property's input domain. Still, the relaxation of the nonlinear activation functions used to compute the over-approximated reachable set can be computationally demanding. In contrast, we propose a computationally efficient solution that uses underestimation of the reachable set and constructs approximate solutions for the ϵ-*RUA-DNN* problem with strong probabilistic guarantees.

Probabilistic Reachable Set

Given the complexity of computing the exact minimum and maximum of the function computed by a DNN, we propose to approximate the output reachable set using a statistical approach known as *Statistical Prediction of Tolerance Limits* [286, 216].

We use a Monte Carlo sampling approach: for an appropriately chosen n, we sample n input points and take the smallest and the greatest value achieved in the output node as the lower and the upper extreme of our probabilistic estimate of the reachable set. The choice of the sample size is based on the results of [286] that allow us to choose n in order to achieve a given desired guarantee on the probability ψ that our estimate of the output reachable set holds for at least a fixed (chosen) fraction R of a further possibly infinitely large sample of inputs. Crucially, this statistical result does not require any knowledge of the probability distribution governing our function of interest and thus also applies to general DNNs. Stated in terms more directly applicable to our setting, the main result of [286] is as follows.

Lemma 3.5: *For any $R \in (0, 1)$ and integer n, given a sample of n values from a (continuous) set X the probability that for at least a fraction R of the values in a further possibly infinite sequence of samples from X are all not smaller (respectively larger) than the minimum value (resp. maximum value) estimated with the first n samples is given by the ψ satisfying the following equation*

$$n \cdot \int_R^1 x^{n-1}\, dx = (1 - R^n) = \psi \tag{3.19}$$

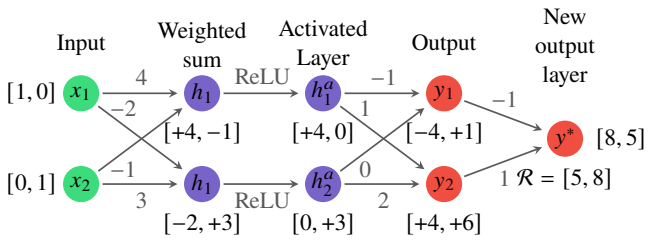


Fig. 3.28: Example of computation single reachable set for a DNN with two outputs.

Computation of Safe Regions

We are now ready to give a detailed account of our algorithm ϵ-`ProVe`.

Our approximation is based on the analysis of an underestimated output reachable set obtained by sampling a set of n points P_A from a domain of interest A. We start by observing that it is possible to assume, without loss of generality, that the network has a single output node on whose reachable set we can verify the desired property [150]. For networks not satisfying this assumption, we can enforce it by adding one layer. For example, consider the network in the example of Fig. 3.28 and suppose we are interested in knowing if, for a given input configuration in a domain $A = [0, 1] \times [0, 1]$, the output y_1 is always less than y_2. By adding a new output layer with a single node y^* connected to y_1 by weight -1 and to y_2 with weight 1 the condition required reduces to check that all the values in the reachable set for y^* are positive.

In general, from the analysis of the underestimated reachable set of the output node computed as $\mathcal{R} = [min_i\ y_i, max_i\ y_i]$, we can obtain one of these three conditions:

$$\begin{cases} A \text{ is unsafe} & \text{upper bound of } \mathcal{R} < 0 \\ A \text{ is safe} & \text{lower bound of } \mathcal{R} \geq 0 \\ unknown & \text{otherwise} \end{cases} \tag{3.20}$$

With reference to the toy example in Fig. 3.28, assuming we sample only $n = 2$ input configurations, $(1, 0)$ and $(0, 1)$ which when propagated through f produce as a result in the new output layer the vector $y^* = [8, 5]$. This results in the estimated reachable set $\mathcal{R} = [5, 8]$. Since the lower bound of this interval is positive, we conclude that the region A, under consideration, is completely safe.

To confirm the correctness of our construction, we can check the partial values of the original output layer and notice that no input generates $y_1 \geq y_2$. Specifically, if all inputs result in $y_1 \geq y_2$ (violating the specification we are trying to verify), then the reachable set must have, by construction, a negative upper bound, leading to the correct conclusion that the area is unsafe. On the other hand, if only some inputs produce $y_1 \geq y_2$, then we obtain a reachable set with a negative lower bound and a positive upper bound, thus we cannot state whether the area is unsafe or not, and we should proceed with an interval refinement process. Hence, this approach allows us to obtain the situations shown to the right of Fig. 3.27, i.e., where the reachable set is either completely positive (A safe) or completely negative (A unsafe).

Algorithm 9: ϵ-ProVe

Input: $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$, n (# of samples to compute $\mathcal{R}$), ϵ-precision desired
Output: set of safe and unsafe regions in $\mathcal{X}$.

$f' \leftarrow$ CreateAugmentedDNN$(f, \mathcal{X}, \mathcal{Y})$
safe_regions $\leftarrow \emptyset$; unsafe_regions $\leftarrow \emptyset$
unknown $\leftarrow$ GetDomain$(\mathcal{X})$
while (unknown $\neq \emptyset$) or (ϵ-precision not reached) **do**
    $A \leftarrow$ GetAreaToVerify(unknown)
    $\mathcal{R}_A \leftarrow$ ComputeReachableSet(f', A, n)
    if lower$(\mathcal{R}_A) \geq 0$ **then**
        safe_regions $\leftarrow$ safe_regions $\cup \{A\}$
    else if upper$(\mathcal{R}_A) < 0$ **then**
        unsafe_regions $\leftarrow$ unsafe_regions $\cup \{A\}$
    else
        unknown $\leftarrow$ unknown $\cup$ IntervalRefinement(A)
return safe_regions, unsafe_regions

We present the complete pipeline of ϵ-ProVe in Algorithm 9. Our approach receives as input a standard tuple for the DNN-Verification and creates the augmented DNN f' following the intuitions provided above. Moreover, we initialize respectively the set of safe regions as an empty set and the unchecked regions as the entire domain of the safety specification encoded in $\mathcal{P}$. Inside the loop, our approximation iteratively considers one area A at the time and begins computing the reachable set, as shown above. We proceed with the analysis of the interval computed, where in case we obtain a positive reachable set, i.e., the lower bound is positive, then the area under consideration is deemed as safe and stored in the set of safe regions we are enumerating. On the other hand, if the interval is negative, that is, the upper bound is negative, we add the area into the unsafe regions and proceed. Finally, if we are not in any of these cases, we cannot assert any conclusions about the nature of the region we are checking, and therefore, we must proceed with splitting the area according to the heuristic we prefer. The loop ends when either we have checked all areas of the domain of interest or we have reached the ϵ-precision on the iterative refinement. In detail, given the continuous nature of the domain, it is always possible to split an interval into two subparts, that is, the process could continue indefinitely in the worst case. For this reason, as is the case of other state-of-the-art FV methods that are based on this approach, we use a parameter to decide when to stop the process. This does not affect the correctness of the output since our goal is to (tightly) underapproximate the safe regions, and thus, in case the ϵ-precision is reached, the area under consideration would not be considered in the set that the algorithm returns, thus preserving the correctness of the result. Although the level of precision can be set arbitrarily, it does have an effect on the performance of the method. In the supplementary material, we discuss the impact that different heuristics and hyperparameter settings have on the resulting approximation.

Theoretical Guarantees

In this section, we analyze the theoretical guarantees that our approach can provide. We assume that the `IntervalRefinement` procedure consists of iteratively choosing one of the dimensions of the input domain and splitting the area into two halves of equal size, as in [274]. The theoretical guarantee easily extends to any other heuristic provided that each split produces two parts, both at least a fixed constant fraction β of the subdivided area. Moreover, we assume that reaching the ϵ precision is implemented as testing that the area has reached size ϵ^d, i.e., it is the cartesian product of d intervals of size ϵ. It follows that, by definition, the areas output by ϵ-`ProVe` are ϵ-bounded and ϵ-aligned.

The following proposition is the basis of the approximation guarantee (in terms of the size of the safe area returned) on the solution output by ϵ-`ProVe` on an instance of the ϵ-*RUA-DNN* problem.

Proposition 3.5: *Fix a real number $\epsilon > 0$, an integer $k \geq 3$, and a real $\gamma > k\epsilon$. Let $\mathcal{T}$ be an instance of the ϵ-RUA-DNN problem. Then for any solution $\mathcal{R} = \{r_1, \ldots, r_m\}$ such that for each $i = 1, \ldots, m$, r_i is γ-bounded, there is a solution $\mathcal{R}^{(\epsilon)} = \{r_1^{(\epsilon)}, \ldots, r_m^{(\epsilon)}\}$ such that each $r_i^{(\epsilon)}$ is ϵ-aligned and $||\mathcal{R}^{(\epsilon)}|| \geq \left(\frac{k-2}{k}\right)^d ||\mathcal{R}||$, where d is the number of dimensions of the input space, and for every solution $\mathcal{R}'$, $||\mathcal{R}'|| = |\cup_i r_i|$ is the total area covered by the hyperrectangles in $\mathcal{R}'$.*

The result is obtained by applying the following lemma to each hyperrectangle of the solution $\mathcal{R}$.

Lemma 3.6: *Fix a real number $\epsilon > 0$ and an integer $k \geq 3$. For any $\gamma > k\epsilon$ and any γ-bounded rectilinear hyperrectangle $r \subseteq \mathbb{R}^d$, there is an ϵ-aligned rectilinear hyperrectangle $r^{(\epsilon)}$ such that: (i) $r^{(\epsilon)} \subseteq r$; and (ii) $|r^{(\epsilon)}| \geq \left(\frac{k-2}{k}\right)^d |r|$.*

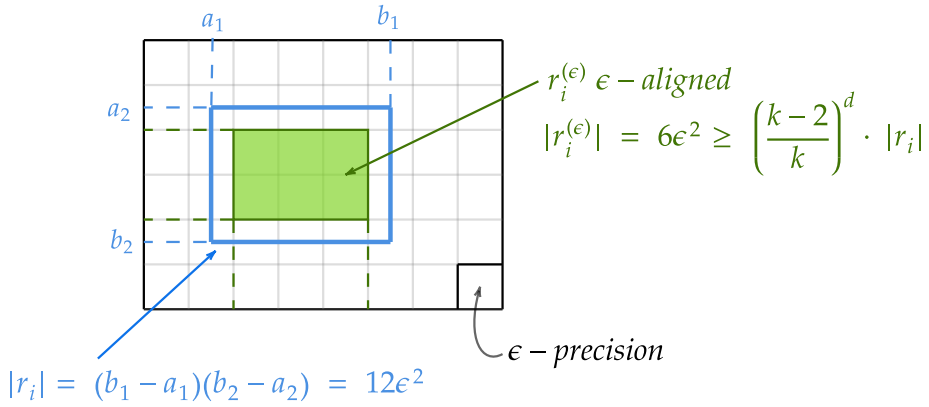


Fig. 3.29: An example of applying Lemma 3.6 with $k = 3$.

Fig. 3.29 gives a pictorial explanation of the lemma. In the example shown, $k = 3$ and the parameter ϵ is the unit of the grid, which we can imagine superimposed to the bidimensional ($d = 2$) space in which the hyperrectangles live. Hence $\gamma = 3\epsilon$. The

non-ϵ-aligned r_i (depicted in blue) is γ-bounded, since its width $w = (b_1 - a_1) = 4\epsilon$ and its height $h = (b_2 - a_2) = 3\epsilon$ are both $\geq 3\epsilon$. Hence, it covers completely at least $w - 2$ columns and $h - 2$ rows of the grid. These rows and columns define the green ϵ-aligned hyperrectangle $r_i^{(\epsilon)}$, of dimension

$$\geq (w-2)\cdot(h-2) \geq \frac{k-2}{k}w \cdot \frac{k-2}{k}h = \left(\frac{k-2}{k}\right)^d |r_i|.$$

The following theorem summarizes the coverage approximation guarantee and the confidence guarantee on the safety nature of the areas returned by ϵ-ProVe.

Theorem 3.8: *Fix a positive integer d and real values $\epsilon, \psi, R \in (0,1)$, with $R > 1 - \epsilon^d$. Let $\mathcal{T}$ be an instance of the AllDNN-Verification with input in*[12] $[0,1]^d$, *and let k be the largest integer such that $\Gamma(\mathcal{T})$ can be partitioned into $k\epsilon$-bounded rectilinear hyperrectangle.*

Let $\mathcal{R}_+^{(\epsilon)}$ and $\mathcal{R}_-^{(\epsilon)}$ be the sets of areas identified as safe and unsafe, respectively, by ϵ-ProVe using n samples at each iteration, with $n \geq \log_R(1 - \psi^{1/m})$ and $m \geq \max\{|\mathcal{R}_+^{(\epsilon)}|, |\mathcal{R}_-^{(\epsilon)}|\}$. Then, with probability $\geq \psi$

1. *(coverage guarantee) the solution $\mathcal{R}_+^{(\epsilon)}$ is a $\left(\frac{k-2}{k}\right)^d$ approximation of $\Gamma(\mathcal{T})$, i.e., $||\mathcal{R}_+^{(\epsilon)}|| \geq \left(\frac{k-2}{k}\right)^d |\Gamma(\mathcal{T})|$;*
2. *(safety guarantee) in each hyperrectangle $r \in \mathcal{R}_+^{(\epsilon)}$ at most $(1-R)\cdot|r|$ points are not safe.*

This theorem gives two types of guarantees on the solution returned by ϵ-ProVe. Specifically, point 2. states that for any $R < 1$ and $\psi < 1$, ϵ-ProVe can guarantee that with probability ψ no more than $(1 - R)$ of the points classified as safe can, in fact, be violations. Moreover, point 1. guarantees that, provided the space of safety points is not too scattered—formalized by the existence of some representation in $k\epsilon$-bounded hyperrectangles— the total area returned by ϵ-ProVe is guaranteed to be close to the actual $\Gamma(\mathcal{T})$. Finally, the theorem shows that the two guarantees are attainable in an efficient way, providing a quantification of the size n of the sample needed at each iteration. Note that the value of m needed in defining n can be either set using the upper limit $2^{d\log(1/\epsilon)}$— which is the maximum number of possible split operations performed before reaching the ϵ-precision limit—or m can be estimated by a standard doubling technique: repeatedly run the algorithm doubling the estimate for m at each new run until the actual number of areas returned is upper bounded by the current guess for m.

Proof. The safety guarantee (item 2.) is a direct consequence of Lemma 3.7. In fact, a hyperrectangle r is returned as safe if all the n sampled points from r are not violations, i.e., their output is ≥ 0 (see (3.20)). By Lemma 3.7, at most $(1 - R)$ of the points in r can give an output < 0, with probability $\hat{\psi} = (1 - R^n)$. Since samples are chosen independently in different hyperrectangles, this bound on the number of violations in a hyperrectangle of $\mathcal{R}^{(\epsilon)}$ holds simultaneously for all of them

[12] This assumption is w.l.o.g. modulo some normalization.

with probability $\geq \hat{\psi}^m$. With $n \geq \log_R(1-\psi^{1/m})$ we have $\psi \leq \hat{\psi}^m$, i.e., the safety guarantee holds with probability $\geq \psi$.

For the coverage guarantee, we start by noticing that under the hypotheses on k, Proposition 3.5 guarantees the existence of a solution $\mathcal{R}_1^\epsilon$ made of ϵ-bounded and ϵ-aligned rectilinear hyperrectangles. Let $\mathcal{R}_2^\epsilon$ be a solution obtained from $\mathcal{R}_1^\epsilon$ by partitioning each hyperrectangle into hyperrectangles of minimum possible size ϵ^d, each one ϵ-aligned. The first observation is that, being a solution made of ϵ-bounded and ϵ-aligned rectilinear hyperrectangles, $\mathcal{R}_2^\epsilon$ is among the solutions possibly returned by ϵ-`ProVe`. We now observe that, with probability $\geq \psi$, each hyperrectangles in $\mathcal{R}_2^\epsilon$ must be contained in some hyperrectangle r in the solution $\mathcal{R}_+^{(\epsilon)}$ returned by ϵ-`ProVe`.

First note that if in each iteration of ϵ-`ProVe`, r' keeps on being contained in an area where both safe and violation points are sampled, then eventually r' will become itself an area to analyze. At such a step, clearly every sample in r' will be safe and r' will be included in $\mathcal{R}_+^{(\epsilon)}$, as desired. Therefore, the only possibility for r' not to be contained in any $r \in \mathcal{R}_+^{(\epsilon)}$ is that at some iteration an area $A \supseteq r'$ is analyzed and all the n points sampled in A turn out to be violation points, so A (including r') is classified unsafe (and added to $\mathcal{R}_-^{(\epsilon)}$) by ϵ-`ProVe`. However, by Lemma 3.7 with probability $(1-R^n)$ this can happen only if $\epsilon^d = |r'| < (1-R)|A|$, which contradicts the hypotheses. Hence, with probability $(1-R^n)^m \geq \psi$, no hyperrectangle of $\mathcal{R}_-^{(\epsilon)}$ contains r', whence it must be contained in a hyperrectangle of $\mathcal{R}_+^{(\epsilon)}$, concluding the argument. □

Fig. 3.30 (left) shows the correlation between the number of points to be sampled based on the number of areas obtained by ϵ-`ProVe` if we want to obtain a total confidence of $\psi = 99.9\%$ and a lower bound $R = 99.5\%$. As we can notice from the plot, if we compute our output reachable set sampling $n = 3250$ points, we are able to obtain the desired confidence and lower bound if the number of regions is in $[1, 10000]$. For this reason, in all our empirical evaluations, we use $n = 3500$ to compute $\mathcal{R}$. An example of the possible result achievable using our approach is depicted in Fig. 3.30 (right) with different shades of green for better visualization.

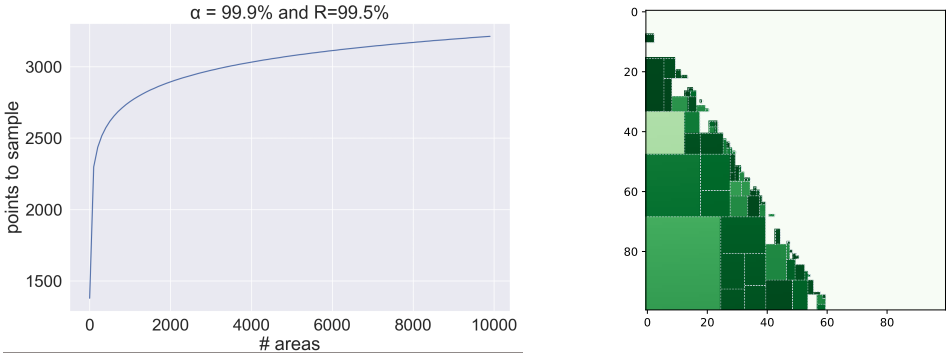


Fig. 3.30: Left: Correlation point to sample and # of (un)safe areas using ϵ-`ProVe` to obtain a confidence $\psi = 99.9\%$ and a lower bound $R = 99.5\%$. Right: example of a set of safe regions (in green) returned by ϵ-`ProVe` (scaled x100).

3.5.2 Empirical Evaluation

In this section, we evaluate the scalability of our approach, and we validate the theoretical guarantees discussed in the previous section. Our analysis considers both simple DNNs to analyze in detail the theoretical guarantees and two real-world scenarios to evaluate scalability. The first scenario is the ACAS xu [125], an airborne collision avoidance system for aircraft, which is a well-known standard benchmark for formal verification of DNNs [150, 131, 273]. The second scenario considers DNN trained and employed for autonomous mapless navigation tasks in a Deep Reinforcement Learning context [174, 168].

All the data are collected on a commercial PC equipped with an M2 Apple silicon. The code used to collect the results and several additional experiments and discussions on the impact of different heuristics for our approximation are available in the supplementary material.

Correctness and Scalability Experiments

These experiments aim to estimate the correctness and scalability of our approach. Specifically for each model tested, we used ϵ-ProVe to return the set of safe regions in the domain of the property under consideration. All data are collected with parameters $\psi_{TOT} = 99.9\%$ and $R = 99.5\%$ and $n = 3500$ points to compute the reachable set used for the analysis. The results are presented in Table 3.25. For all experiments, we report the number of safe regions returned by ϵ-ProVe (for which we also know the hyperrectangles position in the property domain), the percentage of safe areas relative to the total starting area (i.e., the safe rate), and the computation time. Moreover, we include a comparison, measured as percentage distance, of the safe rate computed with alternative methods, such as an exact enumeration method (whenever feasible due to the scalability issue discussed above) and a Monte Carlo (MC) Sampling approach using a large number of samples (i.e., 1 million). It's important to note that the MC sampling only provides a probabilistic estimate of the safe rate, lacking information about the location of safe regions in the input domain.

The first block of Table 3.17 involves two-dimensional models with two hidden layers of 32 nodes activated with ReLU. The safety property consists of all the intervals of $\mathcal{X}$ in the range $[0, 1]$ and a postcondition $\mathcal{Y}$ that encodes a strictly positive output. Notably, ϵ-ProVe is able to return the set of safe regions in a fraction of a second, and the safe rate returned by our approximation deviates at most a 1.75% from the one computed by an exact count, which shows the tightness

Instance	**ϵ-ProVe** ($\psi = 99.9\%$)			**Exact count** or ***MC sampling***		**Und-estimation (% distance)**
	# Safe regions	Safe rate	Time	Safe rate	Time	
Model_2_20	335	78.50%	0.4s	79.1%	234min	0.74%
Model_2_56	251	43.69%	0.3s	44.46%	196min	1.75%
Model_MN_1	545	64.72%	60.6s	*67.59%*	*0.6s*	4.24%
Model_MN_2	1	100%	0.4s	*100%*	*0.4s*	-
ϕ_2 ACAS Xu_2.1	2462	97.47%	26.9s	99.25%	*0.6s*	1.81%
ϕ_2 ACAS Xu_3.3	1	100%	0.4s	*100%*	*0.5s*	-

Table 3.17: Comparison of ϵ-ProVe and Exact count or Monte Carlo (MC) sampling approach on different benchmark setups.

of the bound returned by our approach. In the second block of Tab. 3.17, the Mapless Navigation (MN) DNNs are composed of 22 inputs, two hidden layers of 64 nodes activated with $ReLU$, and finally, an output space composed of five nodes, which encode the possible actions of the robot. We test a behavioral safety property where $\mathcal{X}$ encodes a potentially unsafe situation (e.g., *there is an obstacle in front*), and the postcondition $\mathcal{Y}$ specifies the unsafe action that should not be selected. The table illustrates how increasing input space and complexity affects computation time. Nevertheless, the proposed approximation remains efficient even for ACAS xu tests, returning results within seconds. Crucially, focusing on *Model_MN_2* and ϕ_2 ACAS Xu_3.3, ϵ-`ProVe` states that all the property's domain is safe (i.e., no violation points). The correctness of the results was verified by employing VeriNet [100], a state-of-the-art FV tool.

Impact of Heuristics on ϵ-`ProVe` Performances

Although the correctness of the approximation does not depend on the heuristic chosen to split the input space under consideration (as shown in Theorem 3.8), we performed an additional experiment to understand its impact on the quality of the estimation. In particular, as shown in Alg. 9, the loop ends when either ϵ-`ProVe` explores all the unknown regions, or it reaches a specific ϵ-precision. Regarding the latter, all our results are given for the case of a maximum number of splits equal to $s = 18$, since, in preliminary testing, this value has been shown to give the best trade-off between efficiency and accuracy. Hence, with respect to theoretical analysis in Theorem 3.8), with this choice, we expect the solution returned by ϵ-ProVe to be ϵ-bounded, for some $\epsilon \in [2^{1/s}, 2^{d/s}]$, considering the maximum and the expected number of times that a split happens on the same dimension.

Moreover, another key aspect when using iterative refinement is the choice of the dimension (input coordinate) and the position where to perform the split. To this end, our sampling-based approach to estimating the reachable set also allows us to obtain some useful statistical measures, such as the mean and median of the samples in each dimension, that can be useful for the heuristics. More specifically, we tested the following five heuristics:

- *H1*: split the node along the bigger dimension, i.e., $d = argmax_{i=1}^{m} u_i - \ell_i$, with u_i and ℓ_i the upper and lower bound of each interval, respectively. The split is performed in correspondence with the median value of the samples that result safe after the propagation through the DNN.
- *H2*: as *H1*, but the split is performed along the mean value of the safe samples.
- *H3*: select one dimension randomly, and the split is performed along the median value of the safe samples.
- *H4*: as *H3*, but the split is performed along the mean value of the safe samples.
- *H5*: select the dimension and the position to split based on the distribution of the safe and violation points, i.e., partitioning the input space into parts where we have as many as possible violation points in one part and safety in the other. We report a simplified example of this heuristic in Fig. 3.31.
  More specifically, let us suppose to have only two dimensions and to have sampled n points that result in the situation depicted in Fig. 3.31. Hence, for

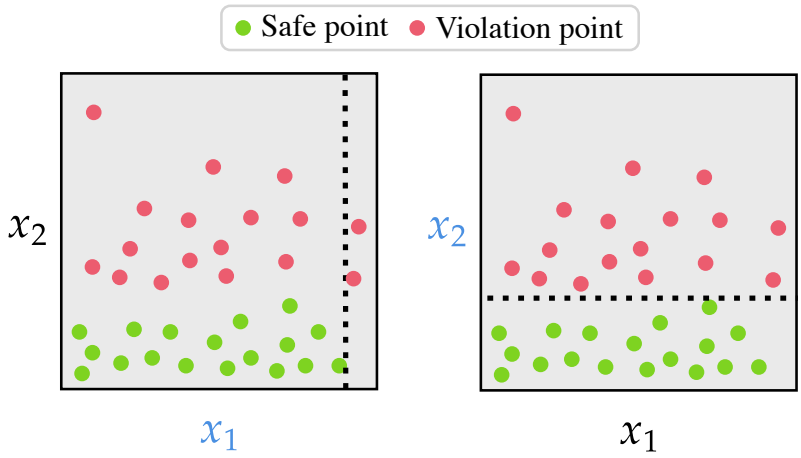


Fig. 3.31: Example of single test split on each dimension x_1, x_2, (highlighted in blue) using $H5$ in a specific situation.

each dimension, the heuristic seeks to find the splitting point that better divides the safe points from the unsafe (violation) ones. In this situation, the dimension chosen will be x_2 as it allows a better partition of the input space, splitting in correspondence with the dotted line, which represents the maximum value of the safe points along x_2.

We have noticed that all these heuristics have roughly the same computational cost on a single split, and thus an improvement in the computation time of the complete execution of ϵ-`ProVe` translates directly into the choice of better heuristics. We tested each heuristic on three different models with $\psi = 99.9\%, R = 99.5\%$, and $n = 3500$ samples. For each heuristic tested, we collected: the number of safe regions, the safe rate, and the computation time, respectively. Ideally, for a similar safe rate, the best heuristic should return the fewest regions (i.e., the best compact solution of the safe regions) in the lowest possible time. In Tab.3.18, we report the results obtained. As we can see, *H5* is the best heuristic as it yields the best trade-off between the efficiency and compactness of the solution for all the models tested. We can notice that for $model\_MN\_3$ the improvement between $H2$ and $H5$ is not significant, we think this is due to the larger cardinality of the input space w.r.t the other models. Further experiments should be performed to devise better heuristics for these types of DNNs.

Instance	H1		
	# Safe regions	Safe rate	Time
Model_2_20	731	78.91%	0.5s
Model_MN_3	1196	53.14%	1m21s
ϕ_2 ACAS Xu_2.1	3540	97.32%	29.1s
	H2		
	# Safe regions	Safe rate	Time
Model_2_20	678	79.01%	1.2s
Model_MN_3	739	53.08%	1m42s
ϕ_2 ACAS Xu_2.1	1613	97.24%	39s
	H3		
	# Safe regions	Safe rate	Time
Model_2_20	2319	78.63%	1.9s
Model_MN_3	-	-	-
ϕ_2 ACAS Xu_2.1	22853	83%	4m20s
	H4		
	# Safe regions	Safe rate	Time
Model_2_20	1757	78.8%	2.9s
Model_MN_3	-	-	-
ϕ_2 ACAS Xu_2.1	31515	78.09%	5m
	H5		
	# Safe regions	Safe rate	Time
Model_2_20	**355**	**78.5%**	**0.4s**
Model_MN_3	**845**	**52.95%**	**1m15s**
ϕ_2 ACAS Xu_2.1	**2462**	**97.47%**	**26.9s**

Table 3.18: Comparison of different heuristic on ϵ-Prove. '-' indicates a heuristic that requires more than 5 minutes. The target safe rates obtained using either an exact enumeration method (when possible) or a Monte Carlo estimation are: for model_2_20 79.1%, for model_MN_3 55.93% and finally for ϕ_2 ACAS Xu_2.1 99.25%.

Full Results ϵ-ProVe on Different Benchmarks

We report in Tab 3.19 the full results comparing ϵ-ProVe on different benchmarks. We consider three different setups: the first one is a set of small DNNs, the second one is a realistic set of Mapless Navigation (MN) DNNs, used in a DRL context, and finally, we used the standard FV benchmark ACAS xu[125]. For the latter, we consider only a subset of the original 45 models. More precisely, based on the result of the second international Verification of Neural Networks (VNN) competition [35], we test part of the models for which the property ϕ_2 does not hold (i.e., the models that present at least one single input configuration that violates the safety property) and one single model (ϕ_2 ACAS Xu_3.3) for which the property holds (i.e., 100% provable safe).

Instance	ϵ-**ProVe** (ψ_{TOT} = 99.9%)			**Exact count** or ***MC sampling***		**Und-estimation (% distance)**
	# Safe regions	Safe rate	Time	Safe rate	Time	
Model_2_20	335	78.50%	0.4s	79.1%	234min	0.74%
Model_2_56	251	43.69%	0.3s	44.46%	196min	1.75%
Model_2_68	252	31.07%	0.3s	31.72%	210min	2.07%
Model_MN_1	545	64.72%	60.6s	*67.59%*	*0.6s*	4.24%
Model_MN_2	1	100%	0.4s	*100%*	*0.4s*	-
Model_MN_3	845	52.95%	75.5s	*55.93%*	*0.4s*	5.33%
ϕ_2 ACAS Xu_2.1	2462	97.47%	26.9s	99.25%	*0.6s*	1.81%
ϕ_2 ACAS Xu_2.2	1990	97.12%	21.7s	*98.66%*	*0.5s*	1.56%
ϕ_2 ACAS Xu_2.3	2059	96.77%	20.0s	*98.22%*	*0.5s*	1.47%
ϕ_2 ACAS Xu_2.4	2451	97.97%	17.5s	*99.09%*	*0.5s*	1.13%
ϕ_2 ACAS Xu_2.5	2227	95.72%	23.6s	*98.19%*	*0.5s*	2.51%
ϕ_2 ACAS Xu_2.6	1745	97.46%	19.2s	*98.79%*	*0.5s*	1.34%
ϕ_2 ACAS Xu_2.7	2165	95.69%	23.6s	*97.35%*	*0.5s*	1.7%
ϕ_2 ACAS Xu_2.8	1101	97.52%	9.0s	*98.06%*	*0.5s*	0.55%
ϕ_2 ACAS Xu_2.9	767	99.24%	6.7s	*99.7%*	*0.5s*	0.47%
ϕ_2 ACAS Xu_3.1	1378	97.37%	13.1s	*98.15%*	*0.5s*	0.79%
ϕ_2 ACAS Xu_3.3	1	100%	0.4s	*100%*	*0.5s*	-
ϕ_2 ACAS Xu_3.4	1230	99.03%	9.3s	*99.53%*	*0.5s*	0.49%
ϕ_2 ACAS Xu_3.5	1166	98.39%	8.0s	*98.9%*	*0.5s*	0.51%
ϕ_2 ACAS Xu_3.6	1823	96.34%	20.8s	*98.17%*	*0.4s*	1.86%
ϕ_2 ACAS Xu_3.7	1079	98.16%	11.1s	*99.82%*	*0.5s*	0.66%
ϕ_2 ACAS Xu_3.8	1854	97.97%	17.0s	*99.09%*	*0.5s*	1.12%
ϕ_2 ACAS Xu_3.9	1374	95.96%	17.6s	*97.34%*	*0.5s*	1.43%
ϕ_2 ACAS Xu_4.1	1205	98.91%	10.1s	*99.6%*	*0.5s*	0.67%
ϕ_2 ACAS Xu_4.3	2395	97.38%	18.9s	*98.56%*	*0.5s*	1.2%
ϕ_2 ACAS Xu_4.4	2693	97.99%	17.7s	*99%*	*0.5s*	1.04%
ϕ_2 ACAS Xu_4.5	1733	97.24%	16.2s	*98.2%*	*0.5s*	1.01%
ϕ_2 ACAS Xu_4.6	1843	96.72%	17.0s	*97.74%*	*0.5s*	1.04%
ϕ_2 ACAS Xu_4.7	1923	96.34%	25.7s	*98.17%*	*0.5s*	1.86%
ϕ_2 ACAS Xu_4.8	1996	95.53%	22.3s	*98.14%*	*0.5s*	1.64%
ϕ_2 ACAS Xu_4.9	601	99.62%	4.3s	*99.85%*	*0.5s*	0.23%
ϕ_2 ACAS Xu_5.1	2530	97.77%	16.5s	*98.72%*	*0.5s*	0.97%
ϕ_2 ACAS Xu_5.2	2496	96.59%	27.7s	*98.92%*	*0.5s*	2.36%
ϕ_2 ACAS Xu_5.4	2875	97.83%	21.6s	*99.13%*	*0.5s*	1.31%
ϕ_2 ACAS Xu_5.5	1660	97.07%	15.1s	*98.03%*	*0.5s*	0.97%
ϕ_2 ACAS Xu_5.6	1909	97.06%	14.7s	*97.94%*	*0.5s*	0.89%
ϕ_2 ACAS Xu_5.7	1452	96.15%	16.2s	*97.2%*	*0.5s*	1.09%
ϕ_2 ACAS Xu_5.8	2357	95.15%	27.5s	*97.74%*	*0.5s*	2.65%
ϕ_2 ACAS Xu_5.9	1494	97.13%	10.9s	*97.83%*	*0.5s*	0.71%
ϕ_3 ACAS Xu_1.3	1	100%	0.7s	*100%*	*0.6s*	-
ϕ_3 ACAS Xu_1.4	1	100%	0.7s	*100%*	*0.6s*	-
ϕ_3 ACAS Xu_1.5	1	100%	0.7s	*100%*	*0.6s*	-
						Mean: 1.27%

Table 3.19: Comparison of ϵ-ProVe and Exact count or Monte Carlo (MC) sampling approach on different benchmark setups.

Moreover, to further validate the correctness of our approximation, we also performed a final experiment on property ϕ_3 of the same benchmark. This property is particularly interesting as it holds for most of the 45 models (as shown in [35]). In particular, in this scenario we tested only models for which the property holds, i.e., we expect a 100% of safe rate for each model tested. For simplicity, in Tab. 3.19, we report only the first three models tested for ϕ_3 since the results were similar for all the DNNs evaluated. As expected, we empirically confirmed that also for this particular situation, ϵ-ProVe returns a correct solution.

From the results of Tab 3.19, we can see that our approximation returns a tight, safe rate (mean of underestimation 1.27%) for each model tested in at most half a minute, confirming the effectiveness and correctness of our approach.

#DNN-Verification on ϵ-ProVe Result

We performed an empirical experiment using an exact count method to confirm the safety probability guarantee of Theorem 3.8. Specifically, theorem 3.8 proves thatϵ-ProVe returns with probability ψ a set of safe areas $\mathcal{R}^{(\epsilon)}$, where for each hyperrectangle $r \in \mathcal{R}^{(\epsilon)}$ at most $(1 - R) \cdot |r|$ points are not safe. This experiment aims to empirically verify this safety guarantee with a formal method. To this end, we rely on ProVe [51], an exact count method that returns the violation rate, i.e., the portion of the area that presents violation points in a given domain of interest.

Due to scalability issues of exact counters, we only perform this evaluation on each of the 252 hyperrectangles returned by ϵ-ProVe for the $model\_2\_68$. The formal results are reported in Tab. 3.20.

Hyperrectangle r	**ProVe** Violation rate
[[0.6999231, 1], [0, 0.2563121]]	0%
[[0.6999231, 1],[0.2563121, 0.40074086]]	0%
[[0.48361894, 0.6999231], [0, 0.18614466]]	0%
[[0.23474114, 0.34573981],[0, 0.09308449]]	0%
[[0.39214072, 0.39503244],[0.245827, 0.25156769]]	**0.031%**
[[0.95905697, 0.95924747], [0.60187447, 0.60229677]]	**0.0488%**

Table 3.20: Partial results of Prove [51] on each single hyperrectangle r returned as solution for the ϵ-*RUA-DNN* for $model\_2\_68$.

The exact count method used on each hyperrectangle r returned by ϵ-ProVe confirms the correctness of our approach. Since we set $R = 99.5\%$ as a provable safe lower bound guaranteed, we expected at most a 0.05% violation rate on each hyperrectangle r_i. Crucially, we notice that for only 2 of 252 hyperrectangles tested, we have a violation rate over the 0%, and this value is strictly less than 0.05%. Hence, this experiment also confirms the strong probabilistic guarantee presented in point 2 of Theorem 3.8.

Summary. In this section, we introduced and studied the AllDNN-Verification, a novel problem in the verification of DNNs, asking for the set of all the safe regions for a given safety property. Due to the #P-hardness of the problem, we presented a novel approximation approach, ϵ-ProVe, which is able to efficiently approximate the safe regions with strong probabilistic guarantees on the tightness of the solution returned. It is important to note, however, that the reliance on a single decision tree to provide the solution often results in the generation of a large number of regions, which in complex scenarios can even exhaust memory resources, producing highly fragmented representations that are difficult to interpret and impractical for downstream tasks such as safe recovery or explanation. To address this issue in the next section, we explore the potential of bootstrap-based and randomized approaches that are capable of capturing complex patterns in high-dimensional spaces, including input regions where a given output property holds.

3.6 On the Probabilistic Learnability of Compact Neural Network Preimages Bounds

The ability of Deep neural networks to learn complex patterns from vast amounts of data has allowed them to tackle challenging tasks in several domains. However, as DNNs become more powerful and pervasive, safety concerns have become increasingly prominent. In particular, DNNs are often considered "black-box" systems, meaning their internal representation is not fully transparent. A crucial weakness of DNNs is the vulnerability to adversarial attacks [246, 8], wherein small, imperceptible modifications to input data can lead to wrong and potentially catastrophic decisions when deployed.

On top of standard DNN-Verification [150, 297, 291, 275], which aims to establish provable guarantees that the network adheres to specific formal specifications, recent works [136, 304], based on seminal results of [55, 186], have formalized the quantitative version of the verification problem, namely identifying the subset of a desired input region where a DNN produces (or not) a desired output. This problem is formally defined as AllDNN-Verification or provable DNNs' preimage bound computation.[13] Computing the preimage bound provides a more informative and fine-grained characterization of the model's behavior, enabling the quantification and localization of the full region of inputs that lead to unsafe outputs, rather than relying on the mere existence of (possibly) isolated counterexamples. This information can be used to guide model debugging, improve training procedures through targeted data augmentation, and inform safe recovery strategies by identifying and avoiding risky regions during deployment. In this context, producing compact representations of such unsafe regions is crucial to enhance explainability and support safer fallback mechanisms, as compact regions are easier to interpret.

However, as for most of the classical enumeration problems (e.g., AllSAT [264]), the exact enumeration of neural network preimage bounds is computationally prohibitive, as the problem has been shown to be #P-hard [173]. To circumvent such a problem, recent efforts [304, 305] have explored the combination of sound under- and over-approximations to approximate the preimage bounds of a neural network with a set of polytopes as compact as possible. Nonetheless, these solutions still face significant scalability issues due to the reliance on a provably sound solution. We argue that the #P-hardness of the problem and its intractability necessitate novel probabilistic solutions that balance computational feasibility with accuracy. Specifically, in this work, we investigate an approximate variant of the AllDNN-Verification problem which is probabilistically solvable, that is, we devise an efficient algorithm that delivers an *approximate and compact* solution with high-confidence guarantees and bounded error. In this vein, in the previous section we propose a probabilistic enumeration of preimage bounds. However, our focus there lies primarily on maximizing coverage, rather than on ensuring compactness of the solution. In fact, as previously stated, the reliance on a single decision tree to provide the solution

[13] We note that our work in [173] (presented in the previous section) and Kotha et al. [136] independently and contemporaneously addressed the same underlying problem under different names. In this section, we use AllDNN-Verification problem or bounding the DNN's preimage interchangeably.

often results in the generation of a large number of polytopes, which in complex scenarios can even exhaust memory resources, producing highly fragmented representations that are difficult to interpret and impractical for downstream tasks such as safe recovery or explanation.

In contrast, in this section, we explore the potential of bootstrap-based and randomized approaches that are capable of capturing complex patterns in high-dimensional spaces, including input regions where a given output property holds. Our probabilistic bounds are, once again, from the realm of *statistical prediction on tolerance limits* [286], which enable high-confidence guarantees on region purity and global coverage.

Specifically, we present **R**andom **F**orest-**Pro**perty **Ve**rifier (`RF-ProVe`), a novel probabilistic approach based on a random forest-inspired classifier. In detail, we exploit an ensemble of randomized decision trees structurally similar to a random forest, but without relying on the traditional majority voting scheme for classification [33].[14] This choice is motivated by the goal of representing the preimage bounds of a neural network as axis-aligned boxes. Alternative representations, such as unions of halfspaces, are computationally more complex and often less interpretable [29]. Although random forests implicitly partition the input space into axis-aligned regions, they are not represented in an explicit way. To address this, we extract axis-aligned boxes directly from the decision paths leading to the leaves of the trees. However, while these leaf regions may appear pure (e.g., according to the Gini index), their reliability could be compromised by limited training data. To mitigate this, we employ a filtering phase based on an *active resampling strategy* that validates the purity of each region. Crucially, our probabilistic guarantees, based on Wilks [286]' results, allow us to formally determine the number of resampling points needed during this filtering phase. This enables us to return a final set of regions for which we can provide high-confidence guarantees on both their individual purity and the overall coverage of the preimage.

Our empirical evaluation on standard verification benchmarks demonstrates that `RF-ProVe` provides a valuable probabilistic framework for challenging instances that are difficult to verify with exact or provable solvers, producing compact solutions with fewer polytopes compared to existing approaches for the (approximate) AllDNN-Verification problem.

In summary, the contributions of this section are:

- We present `RF-ProVe`, a random forest-based method that combines passive learning with an active resampling strategy to efficiently approximate unions of axis-aligned boxes representing compact neural network preimages.
- We develop probabilistic bounds based on [286] statistical tolerance limits, providing high-confidence assurances on the purity and coverage of the extracted input regions, guaranteeing a scalable and practical approximate solution to the (#P-hard) exact verification problem.

[14] Throughout the section, we slightly abuse notation by referring to this ensemble as a random forest, even though it does not employ majority voting.

3.6.1 ALLDNN-Verification or DNN's Preimages Bounds Computation

The AllDNN-Verification problem presented in the previous section, also referred to as exact preimage bounds of a neural network [186, 136, 304], asks for the subset of points in the input space $\mathcal{X}$ that a given function f maps to a given subset $\mathcal{Y}$ of output values, i.e., the pre-image of $\mathcal{Y}$ with respect to f. In detail, we can rewrite the problem as:

Definition 3.11 (AllDNN-Verification Problem):
Input: *A tuple* $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$.
Output: $\Gamma(\mathcal{T}) = \left\{ \boldsymbol{x} \in \mathcal{X} \mid f(\boldsymbol{x}) \in \mathcal{Y} \right\}$.

For the sake of simplifying the presentation, we focus on a binary classification task, and we assume that f is the boolean function obtained by thresholding the single output of a DNN, i.e., such that $f(x) = 1$ iff the output of the DNN is ≥ 0.5, hence we have $\mathcal{Y} = \{1\}$ and $\Gamma(\mathcal{T}) = \{x \in \mathcal{X} \mid f(x) = 1\}$.

As we have already discussed, one possible approach to solve this challenge in an exact fashion, e.g., discovering the set of polytopes that exactly cover the volume of $\Gamma(\mathcal{T})$, $Vol(\Gamma(\mathcal{T}))$, is to leverage the branch-and-bound (BaB) [38, 37] process commonly used in verification and recursively record which regions are (or are not) correctly mapped into $\mathcal{Y}$, as illustrated in Fig. 5.11.

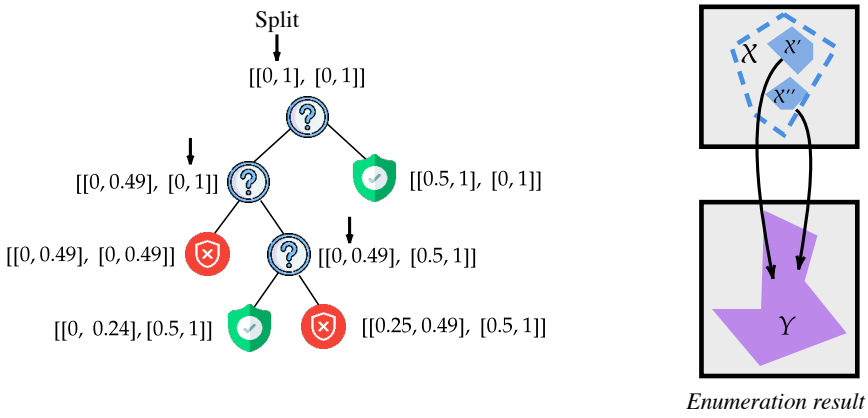


Fig. 3.32: Illustrative overview of AllDNN-Verification problem.

However, as shown in Sec. 3.3, similarly to standard verification, the number of splits either on the input or on the network's non-linearities required in the worst case can grow exponentially, since the problem is #P-hard. Recent progress has been made through *linear relaxation* techniques [297, 291, 275, 290], which over-approximate the network's non-linear behavior and enable backward analysis to compute conservative estimates of the preimage. However, approaches like [136] rely on sound over-approximations and still face scalability limitations, making them unsuitable for quantitative verification. To address such an issue, novel solutions have been proposed in [304, 305, 26] for the approximate version of the problem:

Definition 3.12 (Approximate AllDNN-Verification):
Input: $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle, c \in (0, 1]$.
Output: *a set* $\mathcal{B} = \{b_1, \ldots, b_m\}$ *of disjoint polytopes such that* $\bigcup_i b_i \subseteq \Gamma(\mathcal{T})$ *and* $\frac{Vol(\bigcup_i b_i)}{Vol(\Gamma(\mathcal{T}))} \geq c$.

In this setting, the input includes the tuple $\mathcal{T}$ and a desired coverage ratio of the volume of the preimage set $\Gamma(\mathcal{T})$. Since computing this volume in an exact fashion is computationally prohibitive, typically an estimation is computed, for example, using the Monte Carlo method obtaining $Vol(\Gamma(\mathcal{T})) = Vol(\mathcal{X}) \times \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}_{f(x_i)=1}$ where $x_1, \ldots, x_k$ are sampled from the input domain $\mathcal{X}$, and $\mathbb{1}_{f(x_i)=1}$ indicates whether each sample is mapped to the target set $\mathcal{Y}$ encoded in $\mathcal{T}$. The goal then is to construct a set $\mathcal{B}$ of disjoint polytopes (e.g., axis-aligned hyperrectangles) that under-approximate $\Gamma(\mathcal{T})$ while covering at least a fraction $c \in (0, 1]$ of the estimated volume $Vol(\Gamma(\mathcal{T}))$. Specifically, [304, 26, 305] extend the work of [136] by introducing a novel combination of sound under- and over-approximation strategies based on neural network linearization, effectively guiding the divide-and-conquer procedure for estimating the preimage bounds set. Nonetheless, these approaches are deterministic and sound, but due to the absence of a theoretical bound, to guarantee a desired approximation, the algorithm needs to empirically verify it at run time by estimating the coverage via sampling, which can still lead to scalability issues, as shown also in our experiments.

In this work, we focus on a novel probabilistic relaxation of the problem, where the solution is allowed to (possibly) include some incorrect input points but guaranteeing that with confidence at least $1 - \delta$ the volume of the incorrect points is bounded to at most an ϵ-fraction of the returned solution, and, moreover, this covers at least a desired portion of the exact preimage set.

Definition 3.13 (Probabilistic Approximate AllDNN-Verification):
Input: $\mathcal{T}, c \in (0, 1]$ *and* $\epsilon, \delta \in (0, 1)$.
Output: *A set* $\mathcal{B} = \{b_1, \ldots, b_m\}$ *of polytopes such that, with probability at least* $1 - \delta$,

$$\frac{Vol(\Gamma(\mathcal{T}) \cap \bigcup_i b_i)}{Vol(\Gamma(\mathcal{T}))} \geq c \quad (coverage)$$

and

$$\frac{Vol(\{f(x) \notin \mathcal{Y} \mid x \in \bigcup_i b_i\})}{Vol(\{\bigcup_i b_i\})} \leq \epsilon \quad (error).$$

In this vein, [176] employs a sampling-based approach to generate probabilistically sound reachable sets and designs efficient heuristics to support the BaB verification process, ultimately collecting a set of axis-aligned hyperrectangles. However, as noted earlier, their reliance on a single decision tree often results in highly fragmented representations of the preimage bounds and, in the worst-case scenarios, can lead to memory exhaustion. In this work, we use both approaches, namely the sound under-approximation provided by [305] and the probabilistic one provided by [176], as baselines for our empirical evaluation.

3.6.2 RF-ProVe: a Novel Probabilistic Approach

While recent approximate solutions for AllDNN-Verification have made significant progress in efficiently addressing the problem, they often face trade-offs between scalability and provable coverage guarantees. To address this, we propose `RF-ProVe`, a novel probabilistic random forest learning-inspired method specifically tailored for the probabilistic AllDNN-Verification problem.

Our key idea is to leverage the potential of bootstrap and randomized-based approaches, which are well-suited for capturing complex patterns in high-dimensional spaces. Fig.3.33 illustrates the overall problem and our proposed approach. Given a target output property $\mathcal{Y}$, our objective is to identify the corresponding region(s) in the input space, denoted as $\overline{X}$, that the neural network maps into $\mathcal{Y}$. Since the location of such input regions is not known a priori, we propose to sample labeled examples from the original input space $\mathcal{X}$ and use them to guide the construction of a collection of decision trees. In detail, these trees are used to partition $\mathcal{X}$ into subregions up to a fixed depth D, which inherently defines a user-defined precision parameter $\xi = 2^{-D}$. Consequently, our goal becomes identifying, with high confidence and bounded error, a collection $\mathcal{B}$ of ξ-bounded axis-aligned boxes that approximate, as tightly as possible, the neural network preimage of $\mathcal{Y}$. We highlight that the discretization step does not compromise the soundness of the procedure, as the input space can be assumed to be discretized up to the resolution allowed by machine precision. Moreover, if a region cannot be resolved to the required ξ-precision, it is excluded from the returned set, which preserves the correctness of the final result. In fact, in the worst case, this may lead to a conservative approximation, i.e., a looser under-approximation of the true preimage bounds. Importantly, our method leverages *statistical prediction via tolerance limits* [286] to derive novel theoretical guarantees for the use of randomized ensemble learners such as random forests on both the error within individual regions and the overall coverage of the returned set of boxes.

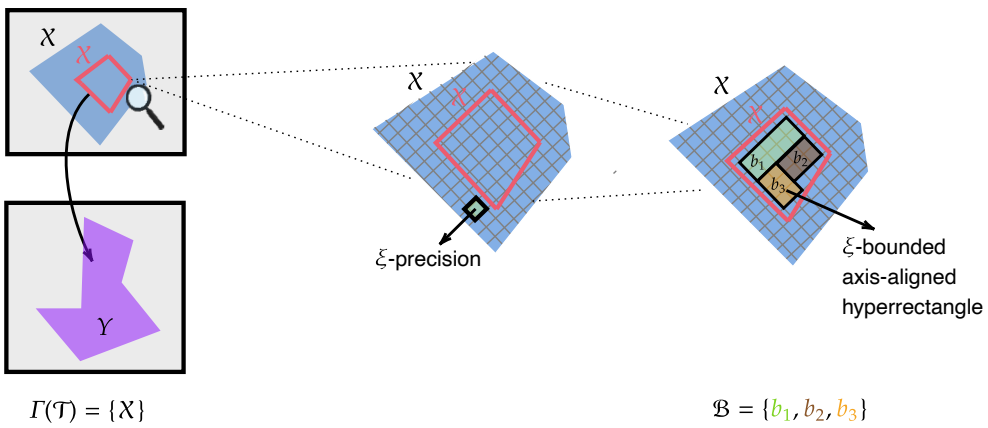


Fig. 3.33: Explanatory image of the solution returned by our `RF-ProVe`.

Algorithm 10: RF-ProVe

Input: $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$, T # decision trees, D maximum depth, R leaf purity desired, δ confidence error, m # training examples, k testing examples, c desired coverage.
Output: $\mathcal{B}$ set of regions (hyperrectangles) satisfying $\mathcal{Y}$, estimated coverage reached.
$\mathcal{B} \leftarrow \emptyset$
S $\leftarrow$ `GetExamples`$(f, m, \mathcal{X}, \mathcal{Y})$
rf $\leftarrow$ `RandomForest`(S, T, D)
for tree in rf.trees **do**
  $B \leftarrow$ `GetPurePositiveLeaves`(tree, $\mathcal{Y}$)
  $n = \frac{\ln(\delta)}{\ln(R)}$
                                                ▷ filtering phase.
  for b in B **do**
    if `SamplesInside`$(b) \geq n$ **then**
      $\mathcal{B} \leftarrow \mathcal{B} \cup b$
    else
      $S' \leftarrow$ `GetExamples`$(f, n, b, \mathcal{Y})$
      if $f(x_i) = 1 \; \forall \; x_i \in S'$ **then**
        $\mathcal{B} \leftarrow \mathcal{B} \cup b$
  $\mathcal{B} \leftarrow$ `RemoveDuplicateBoxes`$(\mathcal{B})$
  coverage, k $\leftarrow$ `EstimateCoverage`$(\mathcal{B}, k)$
  if coverage $\geq$ c **then**
    break
return $\mathcal{B}$, coverage

Random Forest Classifier

The first component of our novel probabilistic approach is a random forest-inspired classifier [33]. Given a labeled dataset $S = \{(x_i, y_i)\}_{i=1}^{m}$, where $x_i \in \mathbb{R}^N$ and $y_i \in \{0, 1\}$, we train a random forest with T (fixed) decision trees. Each tree creates a partition of the input space into axis-aligned boxes, corresponding to its leaf nodes up to a maximum predefined depth D to be reached in each tree. We use the Gini criterion to maximize the purity of leaves (i.e., maximizing the probability of having leaves containing only positive or non-positive examples from S). Hence, after the training of the classifier, we collect all pure positive leaves (boxes containing only positive examples in S) across the T trees and store them in B.

Active Resampling Strategy

Each box in the set B, denoted $b_i \subseteq \mathbb{R}^N$, is an axis-aligned hyperrectangle representing a candidate preimage region in the input space. These boxes are initially extracted from leaves of decision trees in the random forest that appear pure with respect to the target output property $\mathcal{Y}$, based on the Gini. However, this criterion may overestimate the true purity of a region, especially when leaves contain only a few training samples. As a result, a region may appear purely positive due to sampling bias, despite containing unobserved non-positive points. To mitigate this issue and obtain stronger probabilistic guarantees, we introduce an active resampling strategy. Specifically, we compute the number of positive samples $n = \frac{\ln(\delta)}{\ln(R)}$ derived

from our theoretical analysis (detailed in the next paragraph), that each candidate box b_i should contain in order to be stored in the returned solution. Hence, we first verify whether a positive leaf, i.e., a b_i, already contains at least n such samples; if it does, we include b_i in $\mathcal{B}$. Otherwise, we uniformly sample n new inputs from b_i, label them using the neural network f, and collect the results in a set S'. If all inputs $x_i \in S'$ satisfy $f(x_i) = 1$, then b_i is added to $\mathcal{B}$; otherwise, it is discarded. As we will show in the next paragraph, this procedure guarantees that, with probability at least $1 - \delta$, each accepted box $b_i \in \mathcal{B}$ contains at least a fraction R of its volume classified as positive. The boxes in $\mathcal{B}$ may partially overlap, as only full containment is eliminated by the filtering step. Notwithstanding the theoretical guarantee on the achieved coverage (Theorem 3.9), since this, in practice, may speed up the convergence, we also estimate the volume of the coverage of the current solution using a Monte Carlo estimation as in [305].

Specifically, we count how many new examples in a fresh test set of k samples fall within at least one of the collected boxes in $\mathcal{B}$, i.e., satisfying $f(x) = 1$. This empirical estimate serves as a proxy for the true volume of the positive part of the preimage under construction. If the estimated volume reached the desired coverage ratio, we stop the loop and return the solution $\mathcal{B}$ and the corresponding coverage; otherwise, we proceed.

Theoretical Guarantees

In this part, we discuss the theoretical guarantees underlying our `RF-ProVe` approach. To this end, we begin by revisiting the key result on *statistical prediction of tolerance limits* [286], also presented in the previous section 3.5, adapting it to our specific setting.

Lemma 3.7 (Wilks [286]): *Fix a function $g : \mathbb{R}^d \mapsto \mathbb{R}$. For any $R \in (0, 1)$ and integer n, given a sample X_1 of n values from a (continuous) set $X \subseteq \mathbb{R}^d$ the probability that for at least a fraction R of the values in a further possibly infinite sequence of samples x from X the value of $g(x)$ is not smaller (respectively larger) than the minimum value $\min_{x \in X_1} g(x)$ (resp. maximum value $\min_{x \in X_1} g(x)$) of g estimated with the first n samples is at least equal to $1 - \delta$, where δ is the value satisfying the following equation*

$$1 - \delta = n \cdot \int_R^1 x^{n-1} \, dx = (1 - R^n) \tag{3.21}$$

Corollary 1: *Let $g\colon \mathbb{R}^N \to [0,1]$ be a real-valued function and let $\mathcal{X} \subseteq \mathbb{R}^N$ be a region of interest. Let f be the function mapping points from $\mathbb{R}^N$ to $\{0,1\}$ defined by $f(x) = 1$ iff $g(x) \geq 1/2$. Fix $\delta, R \in (0,1)$ and let $n \geq \frac{\ln \delta}{\ln R}$.*

Draw n i.i.d. samples $x_1, \ldots, x_n$ from $\mathcal{X}$. Let $p = \frac{Vol(\{x \in \mathcal{X} | f(x)=1)\})}{Vol(\mathcal{X})}$, be the true fraction of points in $\mathcal{X}$ which are positive for f. If for each $i = 1, \ldots, n$ we have $f(x_i) = 1$ then

$$\Pr\big[p < R\big] < \delta.$$

Equivalently, with probability at least $1 - \delta$ the region $\mathcal{X}$ has at least a fraction $p \geq R$ of positive points for f.

Importantly, Lemma 3.7 and Corollary 5.12 do not require any knowledge of the probability distribution governing the function of interest and thus also apply to general DNNs.

Definition 3.14 (ξ-bounded hyperrectangle): *A rectilinear ξ-bounded hyperrectangle is defined as the cartesian product of intervals of size at least ξ. Moreover, for $\xi > 0$, we say that a rectilinear hyperrectangle $r = \times_i [\ell_i, u_i]$ is* ξ-aligned *if for each i, both extremes ℓ_i and u_i are multiples of ξ.*

Lemma 3.8 (Positive Samples in $b^{(\xi)}$): *Let $\mathcal{X} \in \mathbb{R}^N$ be a region of interest. Fix $\xi, R, \delta \in (0,1)$ and let $n = \frac{\ln \delta}{\ln R}$ be the sample size sufficient to guarantee the bound in Lemma 5.12. Let $Vol_\xi = \xi^N$ be the volume of a hyperrectangle where each side is of size ξ. Fix $\alpha > 1$ and let $m > \frac{n\alpha}{Vol_\xi}$, $\mu = m \cdot Vol_\xi$. and $P_\neg = exp(-\frac{(1-\frac{1}{\alpha})^2 \mu}{2})$. Consider a hyperrectangle $b^{(\xi)} \subseteq \mathcal{X}^N$ of volume Vol_ξ. Then, the probability that among m points independently and uniformly sampled from the input space $\mathcal{X}$ less than n points are from b^ξ is $\leq P_\neg$.*

Proof. For $i = 1, \ldots, m$, let X_i be the indicator random variable of the event that the ith point is from $b^{(\xi)}$. Then, we have $\mathbb{E}[X_i] = Vol_\xi$ and $\mu = m\mathbb{E}[X_i] = \mathbb{E}[\sum_i X_i]$. Then, the desired result is a direct consequence of the Chernoff bound [194]. □

Theorem 3.9 (Coverage Guarantees of `RF-ProVe`): *Let $\mathcal{X} \in \mathbb{R}^N$ be a region of interest. Let $\mathcal{B} = \{b_1, \ldots, b_k\}$ be the collection of disjoint hyperrectangles containing all and only the input positive points of the neural network for $\mathcal{X}$, i.e., $\mathcal{B} = \cup_j b_j = f^{-1}(1)$, where f is the function computed by the neural network. Assume that for each $j = 1, \ldots, k$, it holds that b_j is $k\xi$-bounded, for some $k \geq 3$, hence, in particular, we have $Vol(b_j) \geq k^N Vol_\xi$. Let $\mathcal{B}^* = \bigcup_j b_j$ be the total exact preimage bound.*

Consider a random forest with T random trees trained on m samples, with m, satisfying the bound of Lemma 3.8. Let $\mathcal{B}^A = \{b_1^A, b_2^A, \ldots, b_s^A\}$, be the set of (possibly overlapping) hyperrectangles that estimate the preimage output bounds computed by `RF-ProVe`. Then, we have that $(\frac{k-2}{k})^N Vol(\mathcal{B}^*) \leq Vol(\mathcal{B}^A \cap \mathcal{B}^*)$ and $Vol(\mathcal{B}^A \cap \mathcal{B}^*) \geq R\ Vol(\mathcal{B}^A)$. In particular, the fraction of incorrect points (false positives) among the output boxes satisfies: $Vol(\mathcal{B}^A \setminus \mathcal{B}^*) \leq (1 - R)\ Vol(\mathcal{B}^A)$.

Proof. Recall the definitions and the notation of Lemma 3.8. For the sake of simplifying the argument, We will use the following lemma from [176], rephrased in the context of our present setting.

Lemma 3.9 (Marzari et al. [176]): *Fix a real number $\xi > 0$ and an integer $k \geq 3$. For any $\gamma > k\xi$ and any γ-bounded rectilinear hyperrectangle $r \subseteq \mathbb{R}^N$, there is an ξ-aligned rectilinear hyperrectangle $r^{(\xi)}$ such that: (i) $r^{(\xi)} \subseteq r$; and (ii) $Vol(r^{(\xi)}) \geq \left(\frac{k-2}{k}\right)^N Vol(r)$.*

By applying this lemma to each hyperrectangle b_j we obtain a collection of rectilinear ξ-bounded and ξ-aligned hyperrectangles $\hat{b_1}, \ldots, \hat{b_k}$, such that for each $j = 1, \ldots, k$, we have $\hat{b}_j \subseteq b_j$, and $Vol(\hat{b}_j) \geq \frac{k-2}{k} Vol(b_j)$. Let $\hat{\mathcal{B}} = \bigcup_j \hat{b}_j$. For each j and each ξ-aligned hyperrectangle $b^{(\xi)}$ of volume ξ^N contained in $\hat{b}_j$ we have that the probability that for each tree the training set used for building the forest T contains less than n points sampled from $b^{(\xi)}$ is at most $P_{\neg}^T$. Let $P_{\neg,\mathcal{B}}$ be the probability that for some $j \in [k]$ there is an ξ-aligned hyperrectangle of volume ξ^N included in $\hat{b}_j$ such that in the training set of each tree, less than n samples are from $b^{(\xi)}$. Then, by the union bound, we have $P_{\neg,\hat{\mathcal{B}}} \leq \frac{Vol(\hat{\mathcal{B}})}{\xi^N} P_{\neg}^T$. Hence, with probability $\geq 1 - P_{\neg,\hat{\mathcal{B}}}$, for every ξ-aligned hyperrectangle $b^{(\xi)}$ of volume ξ^N contained in $\hat{\mathcal{B}}$ there is at least one tree t whose training set contains at least n points from $b^{(\xi)}$. Since we are assuming that our algorithm uses ξ-aligned splits, in each tree, the points from $b^{(\xi)}$ will all be assigned the same leaf ℓ. Let b_ℓ be the hyperrectangle associated to ℓ. Since the tree is built so that leaves are pure, the leaf ℓ and hence all the points in b_ℓ are classified as positive. Moreover, since b_ℓ contains $\geq n$ samples, in the output of the algorithm, there is a hyperrectangle containing b_ℓ, i.e., either b_ℓ itself or some hyperrectangle that completely contains it. Since this holds simultaneously for every ξ-aligned hyperrectangle $b^{(\xi)}$ of volume ξ^N contained in $\hat{\mathcal{B}}$ it follows that $\mathcal{B}^A \cap \mathcal{B}^* \supseteq \hat{\mathcal{B}}$, whence $Vol(\mathcal{B}^A \cap \mathcal{B}^*) \geq Vol(\hat{\mathcal{B}}) \geq (\frac{k-2}{k})^N Vol(\mathcal{B}^*)$, which proves the first inequality in the statement of the theorem.

For the right inequality, we note that from b_ℓ we have sampled $\geq n$ points all testing positive. Hence, by Corollary 5.12 with probability at least $1-\delta$, at least a fraction R of b_ℓ contains only positive points, i.e, it is part of the positive preimage. Considering all the boxes returned, we get $Vol(\mathcal{B}^A \cap \mathcal{B}^*) \geq R \sum_i Vol(b_i^A) = RVol(\mathcal{B}^A)$ from which directly follows $Vol(\mathcal{B}^A \setminus \mathcal{B}^*) = Vol(\mathcal{B}^A) - Vol(\mathcal{B}^A \cap \mathcal{B}^*) \leq (1-R)Vol(\mathcal{B}^A)$, concluding the proof. □

These theoretical results show that the ensemble of positive leaves produced by `RF-ProVe` has strong probabilistic guarantees on both purity and coverage. Importantly, since `RF-ProVe` aggregates the positively classified regions from all T trees, the total covered region $\mathcal{B}$ can only grow larger than in the single-tree case. In practice, it is often significantly higher, thanks to the complementary contributions from multiple trees, a phenomenon clearly confirmed by our empirical evaluation.

3.6.3 Empirical Evaluation

In this section, we investigate whether our new random-forest-inspired method, `RF-ProVe`, can generate more compact solutions and better scale with both the input dimensionality and the encoding constraints of the problem. We begin our empirical evaluation by analyzing how to set the hyperparameters of `RF-ProVe` to ensure probabilistic guarantees on both the confidence and the purity of the collected regions.

How to select the hyperparameters? In `RF-ProVe`, two main hyperparameters guide performance and guarantees: the training set size m, and the total number of resampling points n used to validate leaf purity. While there is no closed-form rule for selecting m, as it depends on input dimension, and desired property to verify, we empirically found that using $m = 20000$ uniformly sampled examples provides a sufficiently dense coverage of the input space to populate the leaf regions of the decision trees across various depths. It also ensures that each tree receives a diverse subset of examples via bootstrapping, preserving both region purity and ensemble diversity. Larger values of m yield diminishing returns while increasing training costs. The number of total resampling points n is derived from Theorem 3.9 and depends on the confidence level $1 - \delta$, the minimum required purity R, and the maximum number of candidate regions $|\mathcal{B}|_{max}$, which is dictated by the forest structure. For trees of depth D, each can produce up to 2^{D-1} pure positive leaves, so a forest with T trees yields $|\mathcal{B}|_{max} = T \cdot 2^{D-1}$. Fig. 3.34 (top) shows that even for $D = 12$, achieving up to 1024 boxes, the total needed resamples stay under $1.5M$ for $\delta = 0.001$ and $R = 0.995$, ensuring a very efficient solution. Crucially, rather than relying on deep trees that risk overfitting, we favor many shallow ones to enhance generalization via randomized partitions. Fig. 3.34 (bottom) shows that even for a fixed extreme maximum number of boxes (e.g., 32000), using depths $D \in [5, 7]$ allows for forests with 500–2000 trees. We adopt $D = 5$ in all experiments, offering a scalable and expressive partitioning of the input space.

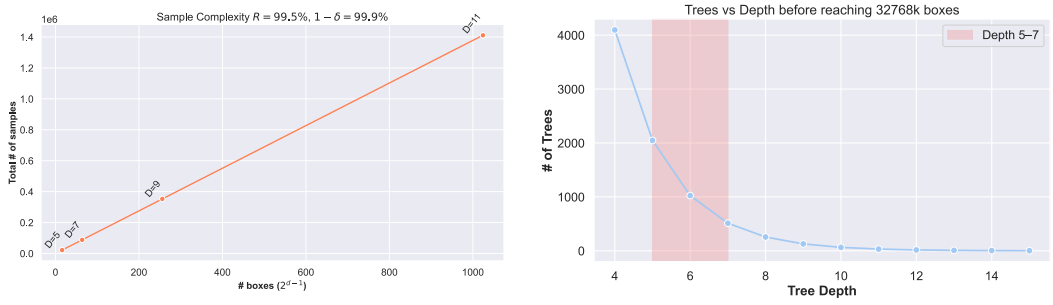


Fig. 3.34: Correlation samples complexity, number of trees, and depth decision trees.

Verification experiments We compare `RF-ProVe` against the Exact [186] solution, provable sound PREMAP [305], as well as the probabilistic approach ε-ProVe [176]. All these approaches compute the preimage using unions of axis-aligned hyperrectangles, making them directly comparable in both representation and output format. In our evaluation, we consider standard verification benchmarks used in [305], such as the aircraft collision avoidance system (*VCAS*) from [124], and reinforcement learning tasks, such as *Cartpole*, *Lunarlander*, and *Dubinsrejoin*.[15] Notably, we focus on structured, verification-relevant domains (e.g., *Dubinsrejoin*) where compact preimage bounds are interpretable and actionable. Image datasets like MNIST or CIFAR lack such semantics and are less meaningful for safety analysis. Since methods like PREMAP and ε-ProVe already struggle with *Dubinsrejoin*, higher-dimensional image inputs would add stress without offering additional insight. To evaluate the quality of the solutions produced by the tested methods, we follow the approach proposed in [305], using for all approximate methods the same number of samples ($10k$) to estimate the coverage, and define a target *coverage ratio* for each task. Given the stochastic nature of the `RF-ProVe`, results Tab. 4.9 including the number of polytopes (# Poly), the achieved coverage, the percentage of impurity (for probabilistic methods), and the runtime across the tested models, report the average result over 3 random initializations. Moreover, we set a desired confidence in the result of $1 - \delta \geq 99.9\%$ (i.e, $\delta = 0.001$) and a maximum error in the final solution of $1 - R \leq 0.005$ (i.e., $R = 0.995$). Our goal is to compute the most compact representation of the preimage region, i.e., using the fewest number of polytopes—while achieving a target level of coverage and ensuring zero, or statistically bounded, impurity. All data are collected on an RTX 2070, and an i7-9700k.

Method	Task	Property	Config	#Poly	Coverage	%error	Time
Exact	VCAS	$\{y \in \mathbb{R}^9 \mid \wedge_{i\in[1,8]}\, y_0 \geq y_i\}$	as in [186]	131	100%	0%	6352.21s
PREMAP	VCAS	$\{y \in \mathbb{R}^9 \mid \wedge_{i\in[1,8]}\, y_0 \geq y_i\}$	as in [186]	**15**	**90.8%**	0%	12.8s
ε-ProVe	VCAS	$\{y \in \mathbb{R}^9 \mid \wedge_{i\in[1,8]}\, y_0 \geq y_i\}$	as in [186]	122	90.48%	0.02%	0.65s
`RF-ProVe`	VCAS	$\{y \in \mathbb{R}^9 \mid \wedge_{i\in[1,8]}\, y_0 \geq y_i\}$	as in [186]	**15**	90.5%	0.06%	0.3s
PREMAP	Cartpole	$\{y \in \mathbb{R}^2 \mid y_0 \geq y_1\}$	$\dot{\theta} \in [-2, 0]$	66	75.5%	0%	32.37s
ε-ProVe	Cartpole	$\{y \in \mathbb{R}^2 \mid y_0 \geq y_1\}$	$\dot{\theta} \in [-2, 0]$	72	76.47%	0.27%	2s
`RF-ProVe`	Cartpole	$\{y \in \mathbb{R}^2 \mid y_0 \geq y_1\}$	$\dot{\theta} \in [-2, 0]$	**22**	**76.8%**	0.3%	4.5s
PREMAP	Lunarlander	$\{y \in \mathbb{R}^4 \mid \wedge_{i\in\{0,2,3\}}\, y_1 \geq y_i\}$	$\dot{v} \in [-4, 0]$	97	75.1%	0%	85.42s
ε-ProVe	Lunarlander	$\{y \in \mathbb{R}^4 \mid \wedge_{i\in\{0,2,3\}}\, y_1 \geq y_i\}$	$\dot{v} \in [-4, 0]$	440	76.51%	0.5%	12.2s
`RF-ProVe`	Lunarlander	$\{y \in \mathbb{R}^4 \mid \wedge_{i\in\{0,2,3\}}\, y_1 \geq y_i\}$	$\dot{v} \in [-4, 0]$	**42**	**75.63%**	0.3%	59s
PREMAP	Dubinsrejoin	$\{y \in \mathbb{R}^8 \mid (\wedge_{i\in[1,3]} y_0 \geq y_i) \wedge (\wedge_{i\in[5,7]} y_4 \geq y_i)\}$	$x_v \in [-0.3, 0.3]$	1002	78.7%	0%	656.47s
ε-ProVe	Dubinsrejoin	$\{y \in \mathbb{R}^8 \mid (\wedge_{i\in[1,3]} y_0 \geq y_i) \wedge (\wedge_{i\in[5,7]} y_4 \geq y_i)\}$	$x_v \in [-0.3, 0.3]$	4929	85.02%	0.3%	260.23s
`RF-ProVe`	Dubinsrejoin	$\{y \in \mathbb{R}^8 \mid (\wedge_{i\in[1,3]} y_0 \geq y_i) \wedge (\wedge_{i\in[5,7]} y_4 \geq y_i)\}$	$x_v \in [-0.3, 0.3]$	**136**	**90.08%**	0.3%	66s

Table 3.21: Empirical evaluation results of preimage approximation for reinforcement learning tasks, with Exact [186], PREMAP [305], ε-ProVe [176] and `RF-ProVe` in gray proposed in this work.

VCAS task results. For the first task, we consider the entire set of VCAS models of the benchmark and we set a desired coverage ratio of at least 90% as in [305]. Tab. 4.9 reports the mean across all the tested models. As we can notice, the Exact method [186] achieves full coverage but at a prohibitive cost, as it requires over

[15] We refer the interested readers to [305] for a comprehensive overview of the selected tasks.

130 polytopes and takes more than 6300 seconds on average to complete. This highlights the scalability bottleneck of exact methods , which even on simpler instances struggle to scale. Importantly, our `RF-ProVe` achieves the same number of polytopes as PREMAP [305] (15) while maintaining extremely low impurity (less than 0.1%) but with an increase of 20× faster runtime, showcasing the power of bootstrapped, data-driven strategies over fixed symbolic solvers.

RL task results. In this experiment, we evaluate preimage approximation methods on neural network controllers across several reinforcement learning tasks. Specifically, we target a coverage of 75% for *Cartpole* and *Lunarlander*, and 90% for the more challenging *DubinsRejoin* task. The Exact method [186] is omitted from this evaluation, as it cannot scale to networks of this size. The results demonstrate the effectiveness of our proposed method. Across all tasks, `RF-ProVe` consistently matches or exceeds the coverage achieved by existing methods, while requiring significantly fewer polytopes and less computation time. The benefit of our approach is particularly evident in the *DubinsRejoin* task, where PREMAP fails to meet the 90% coverage target, achieving only 78.7% coverage despite generating over 1000 polytopes and requiring more than 650 seconds. Similarly, ε-ProVe fails to meet the desired coverage, reaching just 85% while producing a large number of polytopes before encountering memory issues. In contrast, `RF-ProVe` attains 90.08% coverage using just 136 polytopes and 66 seconds, with an impurity of only 0.3%, crucially below the $1 - R = 0.5\%$ desired. This highlights a key strength of our approach: by allowing an infinitesimal error, we can efficiently approximate high-coverage preimages with high confidence, even for complex tasks where exact or provable methods are no longer practical. These results demonstrate the scalability and practical relevance of `RF-ProVe`, offering a valuable alternative for real-world safety-critical applications where soundness can be slightly relaxed in favor of crucial safety information gains.

Ablation study. To assess the contribution of our active resampling strategy, we evaluate the performance of `RF-ProVe` with and without this phase. Specifically, we consider the solution of the method that skips the filtering step and directly returns the pure positive leaves selected by the Gini index from each decision tree, even if the number of positive samples in the leaf is fewer than the one derived theoretically. This isolates the effect of resampling on compactness (number of polytopes), correctness (error rate), and runtime. Table 3.22 summarizes the results on the RL benchmarks.

Method	Task	#Poly	Coverage	%error	Time
`RF-ProVe`	Cartpole	19	75.48%	0.39%	2.6s
`RF-ProVe`	Cartpole	**22**	**76.8%**	**0.3%**	4.5s
`RF-ProVe`	Lunarlander	190	76.33%	3.54%	20s
`RF-ProVe`	Lunarlander	**42**	**75.63%**	**0.3%**	59s
`RF-ProVe`	Dubinsrejoin	308	90.26%	3.43%	39s
`RF-ProVe`	Dubinsrejoin	**136**	**90.08%**	**0.3%**	66s

Table 3.22: Ablation study of preimage approximation for reinforcement learning tasks, with `RF-ProVe` without filtering phase (in white) and original (in gray).

Across all tasks, active resampling consistently reduces impurity by over an order of magnitude, from > 3% down to less 0.5%, while also producing significantly more compact solutions. For instance, in the *LunarLander* task, the number of polytopes drops from 190 to 42 with nearly identical coverage. While the resampling step introduces a moderate runtime overhead (roughly 2×), the added cost is negligible compared to the error reduction and interpretability gain. These results highlight that active resampling is crucial to achieving the desired statistical guarantees of `RF-ProVe`. Without it, the method tends to overfit sparse training data, returning leaf regions that appear pure but actually include a substantial number of non-positive inputs. Hence, we can conclude that the filtering phase effectively corrects this bias by validating each candidate box using a statistically derived number of additional samples, ensuring high-confidence guarantees on region purity.

Summary. In this section, we addressed the computational intractability of exact neural network preimage bound computation by proposing a novel probabilistic framework, `RF-ProVe`. Our approach exploits the strength of bootstrap-based and randomized methods to capture complex structures in high-dimensional input spaces, introducing a random forest-inspired method that combines passive learning with active resampling to approximate preimage regions with high-confidence guarantees. Our novel theoretical results provide strong probabilistic guarantees on region purity and global coverage of the returned solution. Empirically, `RF-ProVe` significantly produces compact solutions, while maintaining low impurity and high coverage, even on complex verification tasks where existing exact, provable, and probabilistic methods fail to scale. Overall, `RF-ProVe` represents a promising shift toward scalable, data-driven verification tools that retain strong probabilistic guarantees. Future work may explore its integration with hybrid verification pipelines and extensions to richer geometric representations.

In the next section, we conclude the chapter by presenting `ModelVerification.jl`, a comprehensive toolbox for verifying deep neural networks that incorporates several of the solutions discussed in the previous sections.

3.7 ModelVerification.jl: a Comprehensive Toolbox for Formally Verifying Deep Neural Networks

The use of deep neural networks is becoming increasingly prominent in several applications, including image classification [78, 138, 146, 95], autonomous navigation [44, 172, 249], robotics [8, 98, 171, 281], and control [151, 279, 280, 288]. The main characteristic of these functions is their ability to approximate complex nonlinear functions often employed in solving these tasks. Nonetheless, while these functions are very efficient, their opaque nature can result in unpredictable and potentially unsafe behavior when small changes in the input, often imperceptible to the human, are performed. Broadly speaking, these functions are subject to so-called “adversarial inputs" [246] that can make them behave unsafely both for the system itself and especially for those around them. Hence, given the applications of DNNs in safety-critical contexts where human life is potentially at risk, the need to obtain formal guarantees about the safety of these systems arises and is of paramount importance.

To address this issue, the research field of formal verificatio of DNNs [150], has emerged as a valuable solution to provide formal assurances on the safety aspect of these functions before the actual deployment in real scenarios. The main goal of FV is to prove (or falsify) a desired input-output relationship (safety property) for a given DNN. More specifically, many methods have been developed to formally verify collision avoidance tasks with standard Feed Forward Neural Networks (FFNNs) or robustness in image classification with Convolutional Neural Networks (CNNs) using reachability analysis [79, 154, 289, 61, 254], optimization techniques [19, 253, 131], or combining the two approaches [100, 31, 297, 275, 298]. Recently, there have also been techniques to find not only an individual violation point in the property’s input domain but also to enumerate entire regions that may lead to unsafe behaviors to repair the network in those specific areas [173, 176, 294].

Despite the considerable advancements made by FV over the years, given the NP-complete nature of the problem [131], there are still several remaining issues, such as scalability, that limit the application of these systems in very large and complex real-world scenarios. Moreover, another limitation of applying FV in realistic scenarios is that existing toolboxes tailored themselves to different assumptions of tasks or properties. Hence, the complexity of the verification landscape in literature implies that users may need to switch between toolboxes or solvers when they intend to employ diverse verification approaches. This necessity poses a significant challenge, as such transitions are often neither convenient nor user-friendly. As a result, using an in-depth and comprehensive pre-deployment formal verification process is hard to achieve, and often, the only guarantees of safety rely on pure empirical evaluations.

To this end, in this work, we present `ModelVerification.jl` (Fig. 4.39), the first comprehensive cutting-edge toolbox that contains a suite of state-of-the-art methods for verifying different types of DNNs and safety specifications.

Our toolbox targets two distinct user categories within the formal verification domain. The first target audience comprises individuals who are relatively new or considered “outsiders" to the FV world. For this latter, our toolbox is designed to be *user-friendly*, accompanied by complete and comprehensive documentation of all the methods developed. Hence, we provide an accessible and educational re-

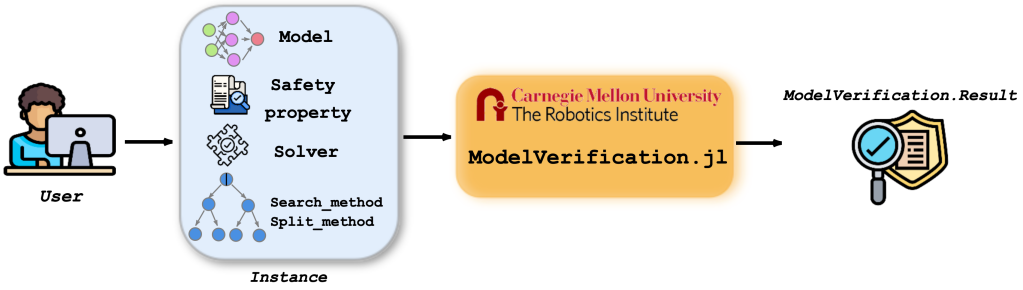


Fig. 3.35: The user specifies the network, the safety property to check, and the solver. `ModelVerification.jl` provides an assertion of whether the safety property holds.

source for those looking to learn the intricacies of the field. Concurrently, the second audience consists of researchers already well-versed in FV practices. Our toolbox offers valuable resources to even the sophisticated requirements of experienced practitioners. More specifically, our toolbox is written in Julia [22] language, ideal for specifying algorithms in human-readable form, and with the key *"multiple dispatch"* feature, that enhances the development of an elegant and highly modularized design for `ModelVerification.jl` . From this design, expert users can access the combination of great performing solvers on par with state-of-the-art ones (i.e., we focus not only on user-friendliness but also on toolbox efficiency) and even the possibility of implementing novel strategies of different natures (e.g., combining optimization and reachability) all in a single comprehensive toolbox.

Yet another FV toolbox? The Formal Verification of DNNs is increasingly becoming essential for providing provable guarantees of deep learning models. We refer the interested reader to the following article for a complete taxonomy of the various state-of-the-art methods [150]. In addition to these works mentioned in [150], it is important to mention recent methods such as Verinet [100], MN-BaB[71], α-β-CROWN [275, 291, 297], which provide the ability to test more complex properties, such as semantic perturbation, in addition to the classic methods based on regression and classification tasks. However, although α-β-Crown, for instance, was the top performer in the last three years of the NN verification competition [35], it lacks support for novel types of DNNs such as Neural Ordinary Differential Equations (NeuralODEs) [161]. To this end, a recent toolbox, NNV 2.0 [155], has arisen to overcome this limitation. Still, the latter presents a lack of support for different deep learning models such as Residual Neural Networks (ResNets) [97], confirming the non-existence of a single, self-contained framework that allows a complete range of verification types. We then have a range of toolboxes such as Juliareach [230], Sherlock [61], jax_verify [56], ReachNN [69], and RINO [87] that mainly focus on specific verification of DNNs (e.g., for Control Systems); and as also pointed out in [155], either they present lack of support for different types of DNNs or are no longer maintained. In contrast, our toolbox covers major state-of-the-art verifiers, including α,β-CROWN [275, 291, 297], Image-Star [254], DeepZ, Zonotope [79], and different layer types as mentioned before, enabling the user to pick the most appropriate solver for the given problem. Hence, `ModelVerification.jl` is the

first self-contained toolbox that supports different verification and safety specification types designed to empower developers and machine learning practitioners with robust tools for verifying and ensuring the trustworthiness of their DNN models.

3.7.1 Toolbox Features

To overcome the limitations presented in the previous section, we now discuss the main features and the improvement of `ModelVerification.jl` over the state-of-the-art in four macro categories:

1) ***Comprehensiveness*****.** As previously discussed, a notable constraint of using pre-deployment FV arises from the lack of a unified framework for verifying a broad spectrum of safety models and properties. Notably, existing solvers employ distinct representations for property verification, or they exclusively address particular categories of DNNs, thereby complicating the transition between tools. Consider a scenario where we have a collection of models encompassing both ResNets [97] and NeuralODEs [161] alongside a set of safety properties to be verified. If, for instance, we opt to use the state-of-the-art α-β-CROWN method [291, 275, 297], it exclusively supports the verification of the former type of networks. Meanwhile, for the latter, an alternative solver such as NNV 2.0 [155] becomes necessary. A critical constraint lies in the fact that these distinct solvers may be implemented following different design architecture strategies or even in different programming languages, as exemplified in this case, where the first solver is coded in Python while the second one is in Matlab. Consequently, accomplishing the verification process, in this case, entails the user's proficiency in both languages and a comprehensive understanding of how safety properties are encoded within the respective toolboxes.

To address such an issue, our primary objective is to provide the community with a tool of maximal comprehensiveness. We report in Tab. 3.23 the main features supported by `ModelVerification.jl` .

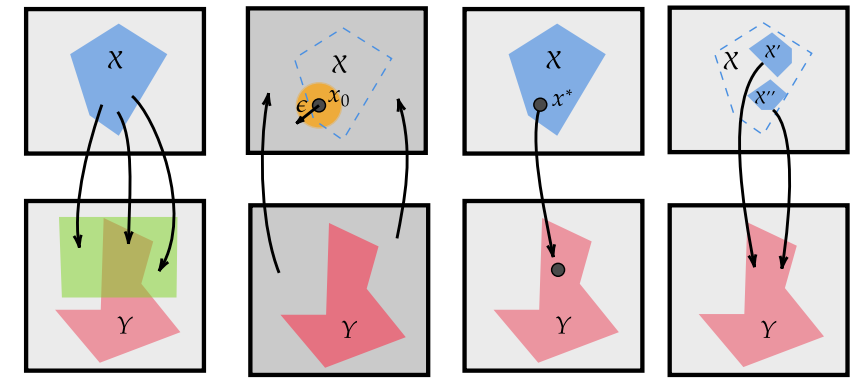
Feature	**`ModelVerification.jl` support**
Neural Network	FFNN, CNN, ResNet, and NeuralODE
Activation functions	ReLU, Sigmoid, Tanh
Layers type	FC, Linear, ReLU, MaxPool, AvgPool, Conv, Identity, BatchNorm, Skip, Parallel
Geometries Representation	Hyperrectangle, Polytope, Zonotope, Star, ImageStar, ImageZono, Image Convex Hull, Taylor Model Reachable Set
Verification	Safety, Robustness, Adversarial attack, VNNLIB, Enumeration of (un)safe regions
Reachable set visualization	Layer-by-layer, Exact and Over-approximation visualization

Table 3.23: Features supported by `ModelVerification.jl` .

Hence, the main purpose of our toolbox is to provide the possibility to verify all different types of neural networks, starting from the classical FFNNs and CNNs up to the more complicated ResNets and NeuralODEs. Also of primary importance is the support of general squashing activation functions, such as Tanh and Sigmoid, in addition to the standard ReLU. Moreover, we decide to write `ModelVerification.jl` in Julia for the following reasons:

- Julia is a language specifically designed for scientific computation, which combines the efficiency of C and the flexibility of Python.
- We have an ample range of libraries available for operations with various complex geometric figures (e.g., *LazySets* [73]). While this gives us prominent performance, it also allows us to encode a wide range of safety properties with consistent geometric representations, resulting in a unified framework as desired.
- Julia's *"multiple dispatch"* feature allows us to adopt a uniform abstract pipeline such that different solvers can share the same function interface. The pipeline is both efficient and easy to follow.

Addressing the comprehensiveness, our toolbox provides the ability to perform several types of verification (Fig. 3.36), not only safety and robustness, using reachability analysis (Fig. 3.36a-b), but also the possibility to perform adversarial attack (Fig. 3.36c) –by exploiting one of the main methods, such as Fast Gradient Sign Method (FGSM), Projected Gradient Descent (PGD) attack, and Auto-PGD. Moreover, our toolbox includes recent exact and approximation methods [294, 173, 176] even to enumerate the set of (un)safe regions of a given safety property (Fig. 3.36d). Finally, `ModelVerification.jl` provides the user with visual representations of the intermediate results of the verification process (i.e., the reachable sets) as depicted in Fig. 3.37. We also introduce a new type of input set, ImageConvexHull, which contains all possible interpolations of the given seed images. ImageConvexHull is particularly useful for semantic perturbations such as occlusion, rain, fog, and shadow.



(a) Reachability analysis **(b)** Robustness result **(c)** Adversarial attack **(d)** Enumeration result

Fig. 3.36: Different types of verification supported in `ModelVerification.jl` . $\mathcal{X}$ represents the safety property's domain, while $\mathcal{Y}$ the undesired reachable set.

All of these features enable `ModelVerification.jl` to verify different types of networks and properties in a single framework. We report in Tab. 3.24 the improvements of our toolbox with respect to α-β-CROWN [291, 275, 297], NNV 2.0 [155], and MN-BaB [71] methods, considered state-of-the-art for formal verification of neural networks.

Features	**Toolbox - Solver**			
	α-β-CROWN	NNV 2.0	MN-BaB	`MV.jl`
Standard Layers	✓	✓	✓	✓
General Comp. Graph	✓	✓	✓	✓
General non-linearities	✓	✓	✓	✓
GPU support	✓		✓	✓
Reachable set vis.		✓		✓
Input sets	L_p-ball, VNNLIB format	L_∞-ball, Zonotope, Star, Polyhedron, VNNLIB format	L_∞-ball, Zonotope, VNNLIB format	L_p-ball, Polytope, Zonotope, Star, ImageConvexHull, VNNLIB format
Solvers	α-β-CROWN, IBP, CROWN, MIP	Zonotope, Star, NeuralODE	IBP, Zonotope, MN-BaB	α-β-CROWN, IBP, CROWN, Zonotope, Star, MN-BaB, NeuralODE

Table 3.24: Comparison between `ModelVerification.jl` and existing state-of-the-art toolboxes.

2) *User-friendliness*. Another major aspect of our toolbox is the ease of use. Our toolbox only requires several lines of code to formulate a verification problem in most cases. We provide comprehensive documentation, facilitating the use of the toolbox through detailed explanations and tutorials and, for Python users, a compiled library of this package such that they can directly call the package from Python itself.

To provide the reader with an intuition of the user-friendliness of our toolbox, let us consider a verification task that considers verifying a ResNet-based NeuralODE. Due to the modularized design chosen for `ModelVerification.jl` and the possibility of combining different solver strategies, we obtain a toolbox that encapsulates a vast range of verification scenarios, avoiding any model architecture modification potentially required in other solvers to meet their specific design.

```
using ModelVerification as MV
model = MV.get_resnet_model("path_to_model")
input_set  = Hyperrectangle(low=[0.9, -0.1], high=[1.1, 0.1])
output_safe_set = Hyperrectangle(low = [2.2, 2.2], high = [2.8, 3.2])
search_method = BFS(max_iter=10, batch_size=1)
split_method = Bisect(1)
prop_method = ODETaylor(t_span=1.0)
verify(search_method, split_method, prop_method, ODEProblem(model, input_set, output_safe_set))
```

More specifically, in `ModelVerification.jl` with a few simple lines of code, reported in the listing above, we can load the desired model and perform the required type of verification, regardless of the dataset we want to use. In particular, our toolbox provides a set of model converters commonly used in the literature, such

as ONNX, TensorFlow (Keras), and PyTorch, to name a few, to Flux models, a Julia library for machine learning that contains an intuitive way to define models, just like mathematical notation. In addition, Flux allows differentiable programming of cutting-edge models such as neuralODEs, typically not supported by state-of-the-art (e.g., α-β-CROWN) methods, as previously discussed. As we can notice in the code above, the high-level language exploited in Julia allows for an easy understanding of what is being performed in the verification phase. Moreover, to increase even further the level of clarity, `ModelVerification.jl` provides the possibility to obtain a set of intra-layer representations of reachable sets obtained during the verification process, as shown in Fig. 3.37. This visualization shows how the perturbations “diffuse" during the reachability analysis, and how does it affects the final prediction, providing a human conceivable robustness.

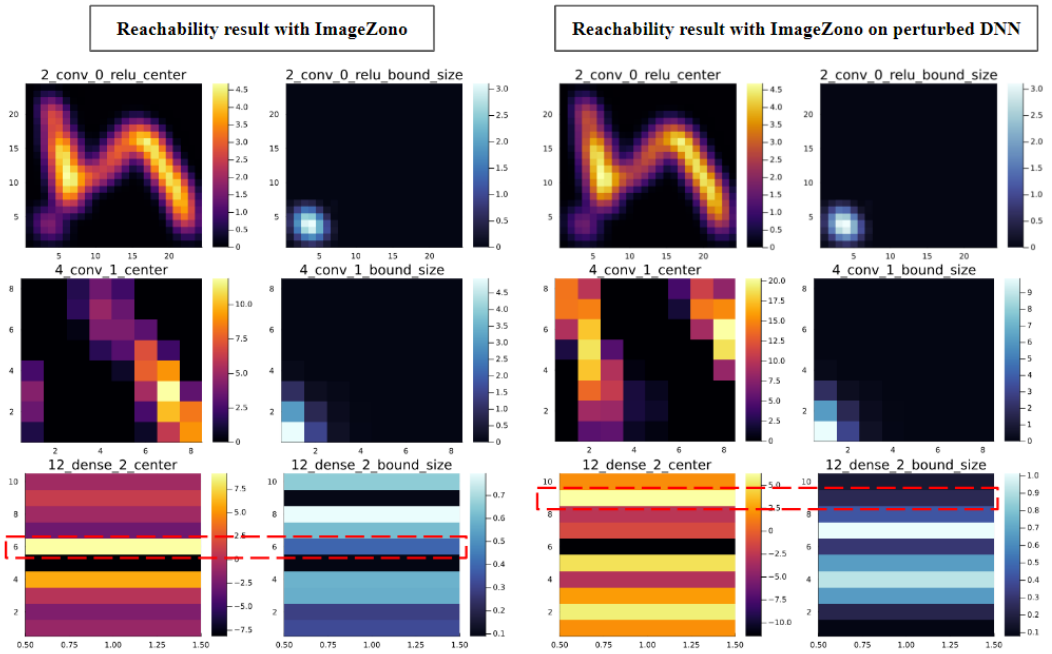


Fig. 3.37: Explanatory example of visualization of the reachable set layer-by-layer using `ModelVerification.jl` for a specific robustness verification instance of MNIST dataset. In this example, a single image representing the “five" handwritten digit and a local perturbation in the bottom left corner of the figure is considered. On the left part of the image, we report layers 2, 4, and 12’s reachable sets computed using ImageZono, where each reachable set is visualized using its center and the bound size using a heatmap. On the right, the reachable sets computed using ImageZono for a perturbed DNN are visualized. A convolutional layer and the last dense layers of the DNN are perturbed to visualize their effect on the final prediction. In the last row, we highlighted the predicted class in red. Crucially, we can notice a large scale in correspondence to the lighter row, meaning that the noise is larger.

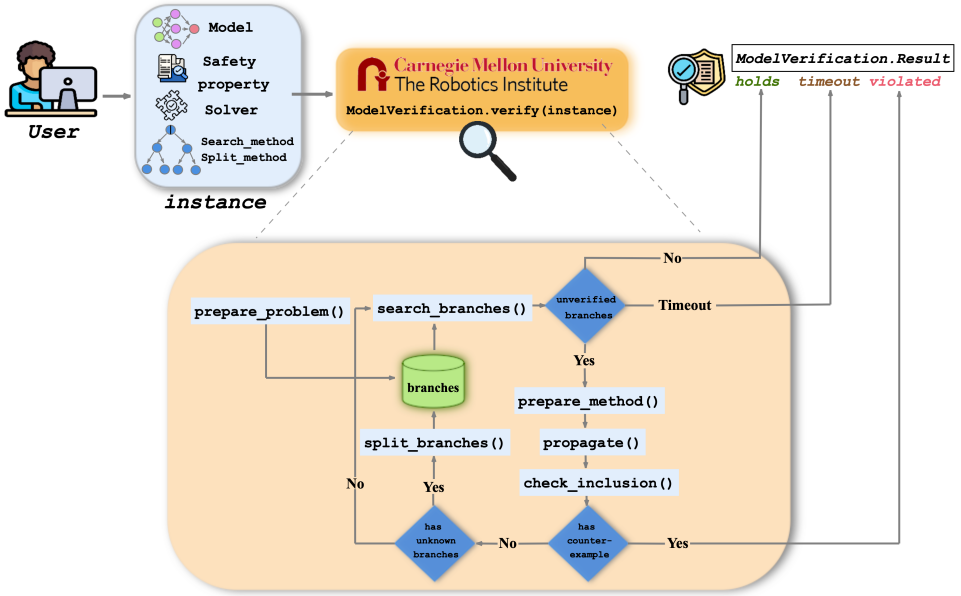


Fig. 3.38: Computational flow of `ModelVerification.jl` . The user provides the verification problem, including the model, the input set, and the desired output property. Our toolbox follows a branch and bound scheme to divide and conquer the problem. A result will be returned to verify or falsify the property if not timed out.

3) *Extensibility*. Our toolbox follows a highly modularized design, making it easy to understand and customize. Specifically, based on the *"multiple dispatch"* feature previously mentioned, we developed straightforward and easy-to-follow implementations. We abstract out a general pipeline and modularize `MV.jl` following the standard Branch-and-Bound [150] paradigm. In detail, all the verification algorithms implemented in our toolbox divide the hard-to-verify problem into easier problems and proceed to verify the single easier subparts. This results in the possibility of choosing and combining different existing solvers provided in the toolbox to solve each part of the verification process optimally.

In the literature, it is worth noting that some solvers work best exploiting GPU computation, while others heavily rely on the CPU. Crucially, our toolbox supports both GPU and CPU-based methods. This dual support, in combination with an elegant, highly modularized, and well-documented BaB design, enables a key feature of our toolbox with respect to other state-of-the-art methods, as such, the possibility to combine different solvers for any verification purpose. We report in Fig. 3.38 a high-level overview of the computational flow of `ModelVerification.jl`. As discussed, each base submethod that composes the *verify* function is highly customizable based on the user's necessity. Based on `MV.jl`, neural control barrier functions [107] and neural Hamilton-Jacobi Reachability value functions [295] can be verified.

4) *Efficiency*. Besides prioritizing user-friendliness, the last main feature of `ModelVerification.jl` is concerned with efficiency. Our toolbox provides significant improvements over the first Julia toolbox ever for FV of DNNs called NeuralVerification.jl (NV.jl) [150]. In detail, NV.jl is written in Julia to provide the community with pedagogical and immediate-to-understand implementations, similar to our goal. However, given the pedagogical nature adopted, performance is suboptimal. In contrast, based on the architectural design choices for our toolbox, we are able to achieve user-friendly implementations and, at the same time, efficient results –in terms of verification time and scalability– comparable to, or in certain cases, surpassing those achieved by state-of-the-art solvers, as shown in Sec. 3.7.2. Recently, [157] efficiently verifies semantic perturbations on images using zonotope-based reachability analysis, while [45] verifies robust model predictive control leveraging the efficient CROWN [297] implementation in `ModelVerification.jl`.

3.7.2 Evaluation

This section demonstrates how versatile `ModelVerification.jl` is in encoding various input-output specifications for various tasks, as well as testing the performance of our toolbox in standard benchmarks from the VNN competition [35] to showcase the efficiency.

Empirical evaluation on VNN benchmarks

The first part of our evaluation concerns the robustness of trained ResNets, in particular, ResNet2b and ResNet4b. This verification is a valuable benchmark of scalability, particularly difficult to verify due to the large number of parameters contained in these architectures. In detail, ResNet2b comprises two residual blocks composed of five convolutional layers plus two linear layers, while ResNet4b has four residual blocks with nine convolutional layers and two linear layers. For this evaluation, we use images from the CIFAR-10 dataset [137] (composed by image size 32×32) with different occlusion perturbations. The occlusion adopted is a 6×6 black block and is randomly placed on the original image. We report in Fig. 3.39 a set of explanatory images of the type of robustness tested. In this study, a total of 100 images were subjected to verification using *ImageZono* as the solver, and the outcomes are presented in Tab. 3.25. All instances yielded deterministic

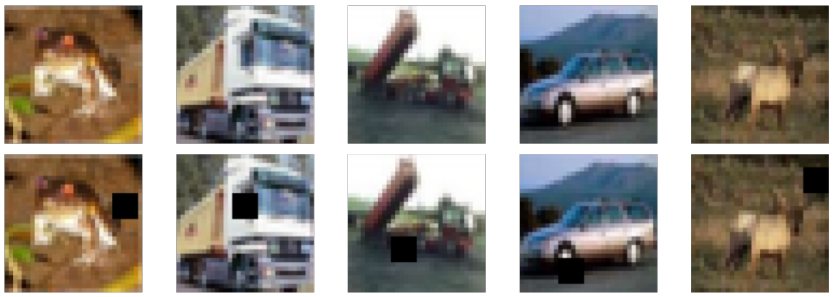


Fig. 3.39: Examples of the original and the occluded images used for the ResNet verification process.

	Holds instance	Violated instance	Unknown instance	#Parameters	Time
ResNet2b	54	46	0	112K	93.51s
ResNet4b	72	28	0	123K	1086.40s

Table 3.25: Verifying ResNet with occlusion perturbation.

results. Notably, the ResNet4b model, characterized by a greater number of layers and enhanced robustness, exhibited a higher number of *holds* instances and thus a longer verification time compared to ResNet2b.

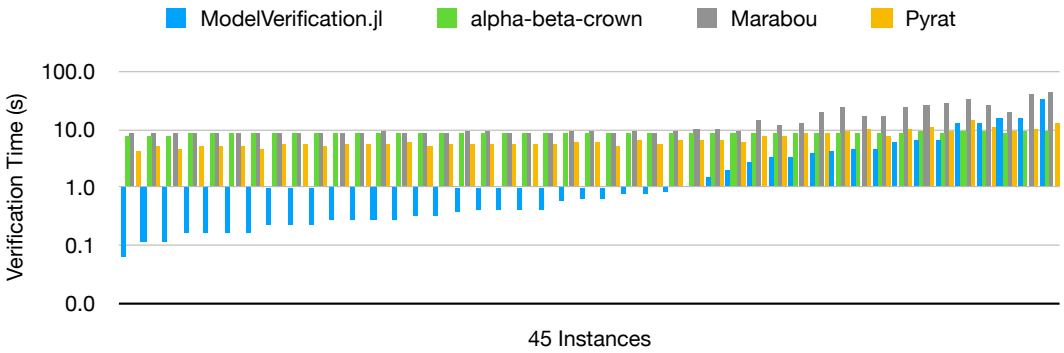


Fig. 3.40: Verification time of 45 instances in ACAS Xu ϕ_1. `ModelVerification.jl` is the fastest for most of the instances. The average verification time is `ModelVerification.jl` : 3.34s, α-β-`CROWN`: 8.37s, `Marabou`: 13.60s, and `PyRat`: 9.12s.

We then evaluate `ModelVerification.jl` on a subset of benchmarks from VNN-COMP'23 [35], ACAS Xu property ϕ_1 for 45 different networks that have 13K parameters. We compare with the toolboxes that won the first three places: α-β-`CROWN`, `Marabou`, and `PyRat`. We run our toolbox on an AWS m5 instance following the same VNN-COMP setup [35] as other toolboxes. The results of other toolboxes are directly from the competition. Detailed setting can be found in our toolbox repository. As shown in Fig. 3.40, our toolbox is faster than other toolboxes for most instances of this property, showcasing its efficiency.

Summary. In this last section, we introduced `ModelVerification.jl`, a comprehensive toolbox for verifying deep learning models. Our tool is the first cutting-edge toolbox containing a suite of state-of-the-art methods for verifying DNNs, including the verification of feedforward, convolutional, ResNet [97], and NeuralODEs. We believe the easy-to-follow implementation, combined with detailed documentation, provides a valuable and unique resource for using formal verification, even for people new to the subject. Moreover, the wide range of geometries that can be employed to describe both safety properties and different types of verification problems allows even the most experienced users to be able to take full advantage of this tool. For the future development of this toolbox, we want to further optimize the performance of the implemented solvers, including making the code more GPU-friendly, optimizing the general structure to reduce redundant computation, supporting more branching algorithms and solvers, and optimizing memory cost as well as performing a more comprehensive comparison with other state-of-the-art toolboxes.

In the next chapter, we will demonstrate how the verification techniques introduced in this part of the thesis can be applied to complex DRL tasks, improving both the safety of the trained models and their explainability.

CHAPTER 4

Formal and Probabilistic Verification for Safe DRL Applications

"Not everything that can be counted counts, and not everything that counts can be counted."

– Albert Einstein

Chapter Contributions[1]

In this chapter, we address **RQ.2**, namely how the formal and probabilistic verification methods introduced in the previous chapter can be effectively applied to enhance and ensure the safety of deep reinforcement learning systems. Due to the NP-hard nature of formal verification, these techniques are typically applied post-training. Within this set of contributions, we investigate two complementary directions: (i) Post-training verification for model selection: we demonstrate how verification can be used to guide model selection in safety-critical applications, such as autonomous navigation in medical robotic colonoscopy. Here, verification helps identify the most reliable policy among several candidates, based on formal safety guarantees. (ii) Verification-informed training: we integrate our novel probabilistic verification methods into the DRL training loop, enabling safety-aware learning. Crucially, we study how safety-relevant information collected during training can be leveraged to automatically infer task-level safety properties, particularly in scenarios where such properties are difficult to specify manually. Finally, we conclude by presenting a novel framework that integrates probabilistic verification to design a control barrier function-based layer that ensures safety for given trained DRL policies for autonomous mapless navigation tasks.

[1] The content of this chapter is based on the following articles:
[C.9] Amir G., Corsi D., Yerushalmi R., Marzari L., Harel D., Farinelli A., and Katz G. (2023) *"Verifying Learning-Based Robotic Navigation Systems"*, International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS).
[C.10] Corsi D.*, Marzari L.*, Pore A.*, et al. (2023). *"Constrained Reinforcement Learning and Formal Verification for Safe Colonoscopy Navigation"*, IEEE International Conference on Intelligent Robots and Systems (IROS).
[C.11] Marzari L. et al. (2023). *"Online Safety Properties Collection and Refinement for Deep Reinforcement Learning Mapless Navigation"*, IEEE International Conference on Robotics and Automation (ICRA)
[C.12] Marchesini E.*, Marzari L.*, Farinelli A., and Amato C. (2023). *"Safe Deep Reinforcement Learning by Verifying Task-Level Properties"*, International Conference on Autonomous Agents and Multiagent Systems (AAMAS).
[C.13] Marzari L., et al. (2025), *"Verifying Online Safety Properties for Safe Deep Reinforcement Learning"*, To appear in ACM Transactions on Intelligent Systems and Technology (TIST).
[C.14] Marzari L., et al. (2025). *"Improving Policy Optimization via ε-Retrain"*, International Conference on Autonomous Agents and Multiagent Systems (AAMAS).
[C.15] Marzari L., et al. (2025). *" ε-Retrainin Reinforcement Learning Algorithms"*, Under review JAAMAS.
[C.16] Marzari L., et al. (2025), *"Designing Control Barrier Function via Probabilistic Enumeration for Safe Reinforcement Learning Navigation"*, IEEE Robotics and Automation Letters (RA-L).

4.1 Neural Network Verification for Safe Colonoscopy Navigation

In 2020, there were 1.9 million new cases of ColoRectal Cancer (CRC) detected globally, resulting in a mortality of 935 thousand people [245]. The World Health Organization (WHO) predicts an average annual increase of 3% worldwide for the next two decades [202]. Early detection is crucial for improving the survival rate, which decreases to below 5% at Stage IV and is close to 100% at Stage 0 [267]. Early detection of ColoRectal Cancer (CRC) is a key element for achieving optimal disease treatment and improving the survival rate. Colonoscopy is a widely adopted diagnostic and therapeutic procedure for CRC, where a Flexible Endoscope (FE) is manually operated by an expert interventionist [159]. However, one of the main drawbacks to this procedure is that patients without sedation experience significant discomfort and pain due to tissue stretching associated with FE manipulation. To this end, Robotic Endoscopes (RE) provide a less painful and more ergonomic approach to colonoscopy. Manual control of RE, however, is prone to human error and requires extensive operator training. These limitations have motivated the development of autonomous navigation systems since navigation is one of the principal phases of colonoscopy [170].

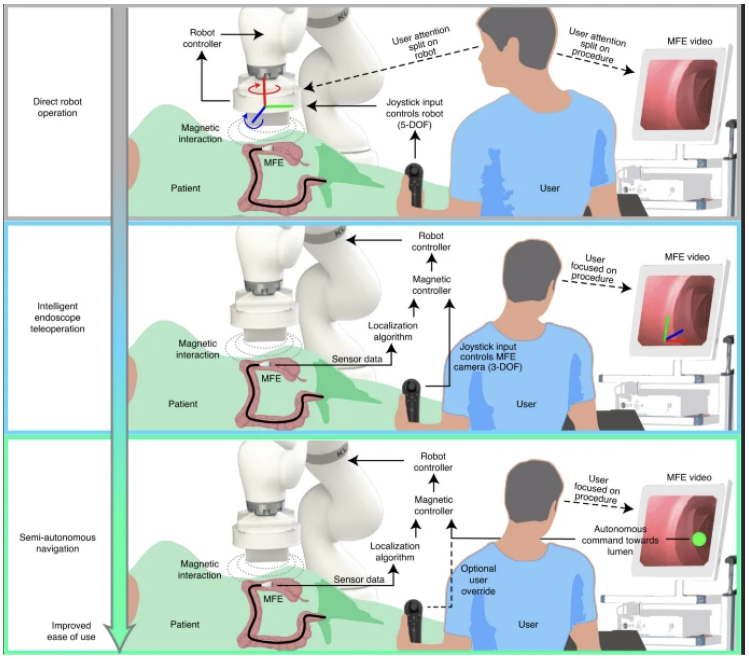


Fig. 4.1: Schematic overview of the control layers associated with autonomy levels. (Martin et al. 2020)

In particular, the navigation system is composed of several elementary blocks organized in three main layers (as shown in the Fig.4.1). Each layer provides a set of features characterized by increasing autonomy, relying on functionalities offered by the underlying layers. Autonomous navigation systems based on visual information use different processing techniques, which rely on the assumption that the region of maximum depth within an image represents a valuable target for immediate heading

adjustment. This region typically corresponds to the darkest area in the endoscopic image. Hence, different segmentation methods for the estimation of darkest region have been proposed based on contour estimation [220, 219], optical flow [224], image intensity [170] and convolutional neural network (CNN) [140].

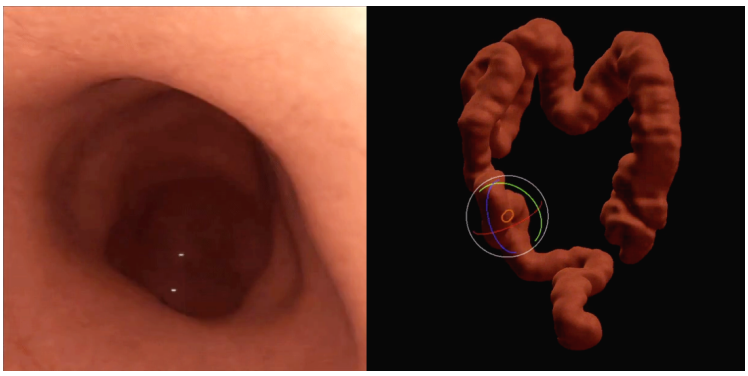


Fig. 4.2: A realistic Unity-based simulator of the colon environment, developed and used in this section.

Regardless of the estimation method used, once the deeper or darker region is detected, a rule-based controller, commonly based on Proportional-Integral-Derivative approach [140, 170] or finite state machines [219], is used to minimize the error with respect to the center of the endoscopic image, called lumen distance. However, these techniques employed to minimize the error with respect to the center of the endoscopic image, called lumen distance, share 2 main limitations: (i) reaching the darkest part of the image does not always correspond to safe navigation in the patient's colon, (ii) these controllers are not robust to rapid changes in the estimates provided, often due to errors in the segmentation method or dynamic deformations of the anatomy.

As an alternative, deep reinforcement learning has been proposed for generating adaptive control signals. In particular, this method uses a deep neural network providing an end-to-end mapping between the endoscopic images and the endoscope's control signal [215]. The capsule agent learns to make decisions through trial and error in the realistic simulated scenario, as the one depicted in Fig. 4.2. However, implementing DRL in real-world robotic systems raises concerns over safety, which is of utmost priority in surgical settings. As discussed earlier, since DRL methods rely on DNNs, they are vulnerable to unexpected behaviors in situations not encountered during training, which can lead to potentially harmful consequences [246]. To this end, in this section, we propose the use of a constrained reinforcement learning (CRL) techniques which provide a way to tackle safety by restricting the agents from taking potentially unsafe actions through the incorporation of an additional so-called *"cost function"* that should be minimized. While the reward function incentivizes specific behavior, the cost function is designed to penalize undesired actions. However, in practice, achieving a perfect zero-cost result through numerical optimization in DRL is often impractical. Thus, a threshold is set as a maximum acceptable value

for the cumulative cost. Examples of algorithms to face this challenging problem include CPO [2], based on the concept of safe policy improvement. In this section, we employ the Lagrangian Proximal Policy Optimization (L-PPO) algorithm, which leverages the Lagrangian dual relaxation of a constrained optimization problem [243]. L-PPO inherits all the strengths of the PPO algorithm [232] (e.g., trust-region policy improvement and first-order optimization) while offering a simple and efficient method for updating constraints.

In CRL methods, safety specifications estimate the expected risk over the entire trajectory and do not guarantee safety at a particular state [150, 2]. Hence, ensuring that the DRL agent never causes safety violations is crucial. Formal Verification has been recently applied to robot-assisted surgical tasks [214]. However, previous FV approaches applied to robot-assisted surgical setups only identify states that can potentially lead to safety violations without utilizing the findings for other scalable objectives, such as model selection.

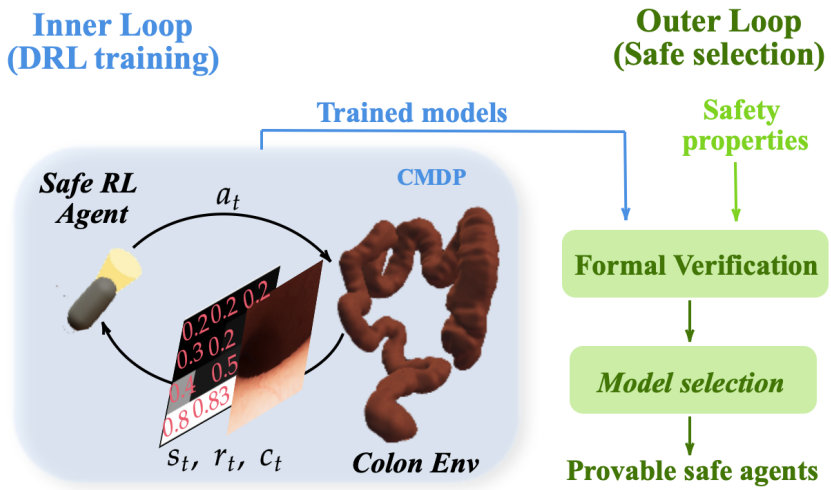


Fig. 4.3: Safe reinforcement learning framework proposed in this work. Agents are trained in a CMDP setting with soft constraints. The trained policies are examined with the FV tool, which identifies the safety violations. Policies without safety violations are selected for final deployment to ensure a completely safe behavior.

In this study, we develop a model selection strategy (see Fig.4.3) that chooses policies without any safety violation, by formally verifying each of the policies over a set of safety properties.

Therefore, we propose a novel framework that integrates the following features:

- An end-to-end DRL method for colon navigation that eliminates the need for a separate lumen detection system.
- A CRL approach that constrains the policy in a pre-defined safe state-space to minimize potentially dangerous actions.
- A model selection strategy that selects policies satisfying all safety constraints, with each policy formally verified to check for its safety violations.

We evaluate the proposed framework in a virtually simulated colonoscopy setup that accurately emulates the dynamics of colon tissue. The colon navigation per-

formance and the safety of the proposed CRL approach are evaluated against the standard safe DRL approaches. Our results demonstrate that the combination of CRL and FV can improve safety in autonomous colonoscopy navigation.

4.1.1 Problem Statement

In this section, we give an overview of the colonoscopy environment used and briefly introduce the safety objectives for autonomous navigation.

Colonoscopy Environment

The 3D models of the colon are derived from publicly available CT colonography datasets and are refined to generate volumetric and superficial meshes with realistic textures [110, 215]. In order to simulate the deformable nature of the colon, a biomechanical model based on the Simulation Open Framework Architecture (SOFA) is integrated in Unity3D to obtain a high-quality, realistic anatomical environment.

To simulate the navigation of a RE tip actuation, we consider a rigid capsule with a camera, weighing 20g and with a length and diameter of 36mm and 14mm, respectively, as the endoscope tip. This modeling approach is consistent with previous works [110, 215].

Overview of the safety framework

The conventional approach of using the region of greatest depth as the immediate heading adjustment goal suffers from an intrinsic weakness, where lighting conditions, focal length, and surrounding tissue geometry can considerably impact the actual distance to the deepest point, making it an unreliable target for precise navigation [140, 170]. Relying solely on the perfect alignment of the camera towards the deepest point ignores the 3D structure of the surrounding anatomy and will inherently limit the ability of the endoscope to navigate through tight bends (as depicted in Fig. 4.4), where a large proportion of images will not be well-centered within the lumen (shown in Fig. 4.5a). This results in close-up views of the lumen wall, which can be highly illuminated due to reflection. Thus, defining safety objectives to prevent the endoscope from moving in the orthogonal direction of the colon wall, which could potentially lead to perforation, can result in a safer trajectory.

Henceforth, we establish two safety indices for our study: (1) Soft constraints that provide guidance for avoiding colon wall collisions based on safety probability analysis and optimization. The incorporation of soft constraints during the training of standard DRL methods facilitates the agent to learn actions that conform to safe configurations, leading to CRL. (2) Hard constraints that impose strict restrictions on the system to prevent it from entering specified unsafe regions, such as perforation. We observe that movement of the scope towards the illuminated region of the image can lead to a trajectory orthogonal to the wall. Consequently, we propose a set of four hard constraints, referred to as safety properties (Θ), based on user-defined brightness thresholds in different image regions. If the values surpass the threshold, the robot must restrict its actions in that direction. It is often difficult for CRL methods to enforce hard constraints by setting indirect constraints on the

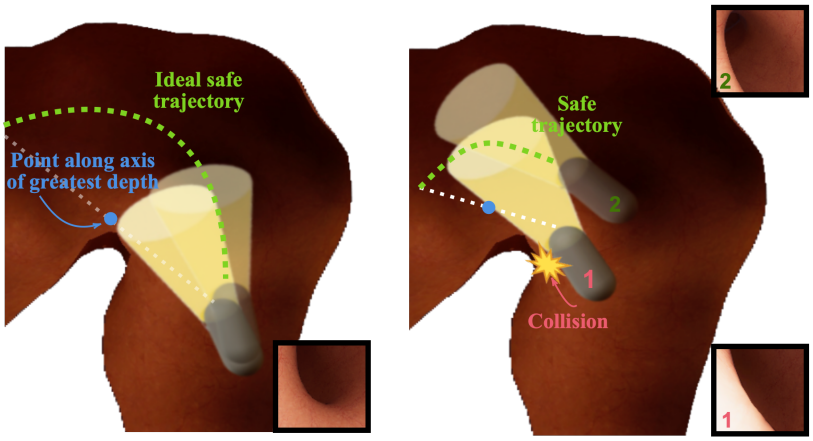


Fig. 4.4: Capsule endoscope positioned inside the lumen, facing an upcoming turn, with both the point of greatest depth (darkest point) and the lumen center visible. If the capsule endoscope is guided towards the deepest point, it will inevitably cause biased motion towards the inner wall of the turn, as shown in (a). This biased motion could potentially lead to occlusion of the camera and collision with the wall, as shown in (b). The right-hand side square boxes display the endoscopic view, with (a) showing the two endoscope tips at a similar position, providing the same view, and (b) illustrating that endoscope 1 approaches the wall more closely than endoscope 2 as it follows the line of greatest depth.

expectation of cumulative cost [150, 2]. Therefore, a key objective of our work is to leverage FV techniques to analyze the policy and assess its adherence to the specified hard constraints.

Observation Space

The observation space is characterized by a low-dimensional discretization of the endoscopic image, represented by a 4x4 matrix (as depicted in Fig. 4.5b). The image is first discretized by dividing each dimension into four regions, and each square region is assigned a value that represents the normalized average of the underlying pixels. We demonstrate that achieving state-of-the-art autonomous navigation performance is feasible even with a discretized low-dimensional input space, without relying on CNN or other complex architectures. This discretization step further simplifies the FV process, which can be computationally expensive and face scalability challenges when dealing with high-dimensional inputs like images, as shown in the previous chapter. The resulting 2-D down-scaled image is then flattened into a list of 16 values, which form the input to the DRL algorithm.

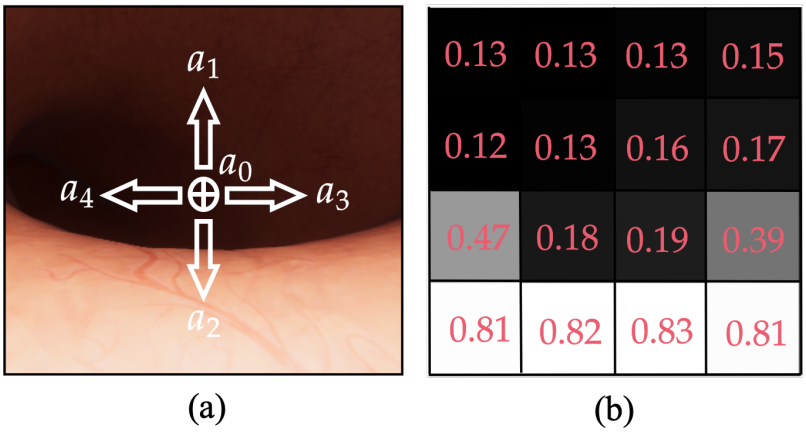


Fig. 4.5: (a) Endoscopic view with the allowed actions. (b) Discrete representation of the input space used for the agent.

Action space

The action space is comprised of five discrete actions, each of which corresponds to a movement in one of the four cardinal directions, namely a_1:`up`, a_2:`down`, a_3:`right`, and a_4:`left`; plus an additional action, a_0:`center`, to set the angular velocity to zero. The agent moves at a constant linear velocity of 3 mm/s. The angular velocity, which determines the rotation of the endoscope tip, is dependent on the specific action selected and corresponds to a fixed angle of 0.017 rad/s in the two degrees of freedom. To facilitate the input-output mapping, the neural network controller has been designed with 5 output neurons (one for each action) and 16 input nodes, which is consistent with the discretized image representation discussed in the previous subsection.

Reward function

We design a reward function that incentivizes the agent to reduce the distance from the end of the colon while minimizing the interactions with the colon wall. In light of this, we have formulated a reward function that provides a high positive reward to the agent upon successful completion of the task, a small penalty upon touching the colon wall, and an additional penalty that scales with the distance between the agent and the end of the colon. The mathematical expression of the reward function is as follows:

$$R_t = \begin{cases} 10 & \text{reaches the end} \\ -\beta & \text{touches the wall} \\ (-dist_t) \cdot \eta & \text{otherwise} \end{cases} \tag{4.1}$$

where $dist_t$ is the centerline distance from the end of the colon at time t. The centerline distance for each colon model is estimated prior to training using checkpoints. η is a normalization factor, and β is a fixed penalty for each collision. The values of η and β are empirically set to 0.001 and 0.01, respectively, in our experiments.

For the sake of clarity in the following, we recall the notation and the concepts presented in Chapter 2 regarding constrained reinforcement learning and verification of neural networks.

Constrained DRL and Lagrangian-PPO

In the previous sections, we have discussed the concept of optimal policy for an MDP. However, in safety-critical scenarios, it is necessary for an agent to guarantee additional behaviors of paramount importance, even more than achieving the primary task [243]. For instance, in colonoscopy, preventing lumen wall perforation takes precedence over reaching the destination, despite the latter being the primary objective [219].

This issue is typically addressed by modeling the problem using a constrained Markov decision process, which is an extension of a standard MDP that includes an additional signal, namely the *cost function*, defined as $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a threshold value $c \in \mathbb{R}$ that the expected value of the cost must remain below. For the sake of simplicity, we consider the case of only one cost function and its corresponding threshold, but the framework can be easily extended to handle multiple constraints. We formally define the set of feasible policies for a CMDP as follows:

$$\Pi_C := \{\pi_\theta \in \Pi : \ \forall k, \ J_C(\pi_\theta) \leq c_k\} \tag{4.2}$$

where $J_C(\pi_\theta)$ is the expected cost function over the trajectory and c_k is the corresponding threshold.

A constrained DRL algorithm should find a policy $\theta \in \Pi_C$ that maximizes the reward. A natural way to encode this problem is through a constrained optimization problem in the form of:

$$\max_{\pi_\theta} \quad J_r(\pi_\theta), \quad \text{s.t.} \quad J_C(\pi_\theta) \leq c \tag{4.3}$$

One viable method to incorporate the constraints in an optimization problem involves the utilization of *Lagrange multipliers*. In the context of DRL, a possible technique is to transform the constrained problem into its dual unconstrained counterpart. The objective function for optimization can be expressed as follows:

$$J(\theta) = \min_{\pi_\theta} \max_{\lambda \geq 0} \mathcal{L}(\pi_\theta, \lambda) \tag{4.4}$$

where $\mathcal{L}(\pi_\theta, \lambda) = J_r(\pi_\theta) - \lambda(J_C(\pi_\theta) - c)$. Among the various DRL algorithms that can be used to maximize this function, a typical choice is to exploit PPO, which has shown promising results when applied together with the Lagrangian dual optimization [243]. In our setup, we consider the same reward function of PPO, with the addition of a cost function that assumes a value of 1 only when the capsule interacts with the wall; otherwise is always 0. A cost threshold values of 500 was selected, balancing the safety of the capsule without compromising its reward performance (discussed in Sec.4.1.2).

Formal Verification

Formal Verification of DNNs [150] is mathematically defined by the tuple $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$, where f is a trained DNN, is a precondition on the input, and $\mathcal{Y}$ is a postcondition on the output. The precondition $\mathcal{X}$ specifies the admissible input configurations that are of interest, while the postcondition $\mathcal{Y}$ represents the desired

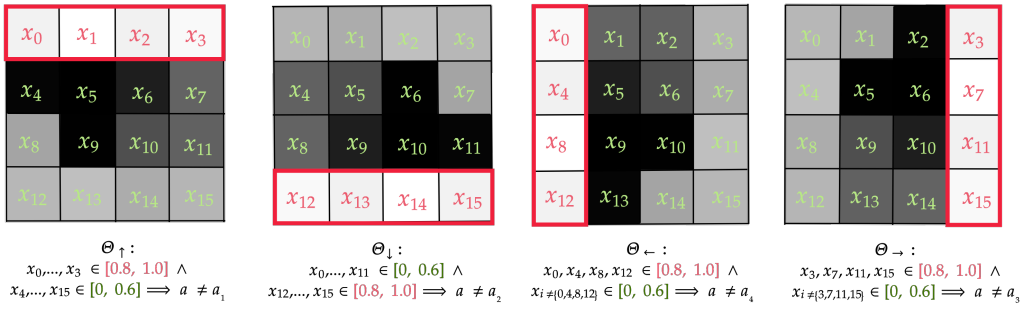


Fig. 4.6: Illustration of four safety properties designed, namely $\Theta_\downarrow, \Theta_\uparrow, \Theta_\rightarrow$, and $\Theta_\leftarrow$. When the scope is close to the upper, lower, left, or right lumen wall, the respective row squares in the input space have high illumination with values in $[0.8, 1]$, hence the agent should not move in that direction.

output results that must be verified. Solving the verification problem requires demonstrating the existence of at least one concrete input (vector) $\boldsymbol{x}$ that satisfies the given constraint, as formulated by the following assertion:

$$\exists\, \boldsymbol{x} \mid \mathcal{X}(\boldsymbol{x}) \wedge \mathcal{Y}(f(\boldsymbol{x})) \tag{4.5}$$

As previously discussed, the verification process uses a search procedure to determine whether there exists an input vector $\boldsymbol{x}$ that satisfies both the precondition $\mathcal{X}$ and the postcondition $\mathcal{Y}$, returning `SAT` if such a vector is found [131]. To solve the verification problem, we employ VeriNet, a state-of-the-art FV tool [100].

In this section, we define a set of safety properties, denoted as $\Theta_\downarrow, \Theta_\uparrow, \Theta_\leftarrow$ and $\Theta_\rightarrow$, to ensure safe operation of the agent during colonoscopy, shown in Fig. 4.6. The safety properties are expressed using the possible input values $\boldsymbol{x} = [x_0, \ldots, x_{15}]^T$ for f, where x_0 and x_{15} represent the values of the upper-left and bottom-right squares, respectively and the five possible actions, denoted as $a_0, \ldots, a_4$ (highlighted in Fig. 4.5a), that the agent can take.

The precondition $\mathcal{X}$ is encoded using a set of hyper-rectangles, represented by intervals, one for each possible input value. In more detail, we consider two types of intervals to encode $\mathcal{X}$: $[0, 0.6]$ represents a safe image area, free of obstacles, while the interval $[0.8, 1]$ represents a bright image area, i.e., the agent is close to the colon wall (illustrated in Fig.4.6). The postcondition $\mathcal{Y}$ requires the agent to choose any action other than a_i, which corresponds to the unsafe action of scope motion in the direction of illumination. Hence, to verify these properties, FV searches for a single input $\boldsymbol{x}$ that satisfies $\mathcal{X}$ and for which f satisfies the negation of the postcondition, i.e., a configuration in which the agent selects the unsafe action a_i. If no such configuration is found, the original property holds.

It is important to emphasize that the safety properties outlined in our study specify the action the agent should not take in an unsafe situation. However, they do not specify which action should be taken instead. This is a crucial concept because we do not want to force the agent to select a specific action, limiting its capability of finding novel and optimal strategies, but instead only avoiding the most harmful actions.

4.1.2 Experimental Validation

In this part, we present the results of the empirical evaluation of the proposed framework. The experiments aim to address the following research questions: (Q1) *What is the effect of a constrained approach on the training of the agent and its performance in the task of autonomous colon navigation?* (Q2) *Can reducing the violation of soft constraints lead to the elimination of violations of hard constraints?*

Experimental setup

The evaluation of the proposed framework is based on four colon models of varying complexity, characterized by their length and the number of acute bending angles (exceeding 90°), as shown in Fig. 4.7 and detailed in [215].

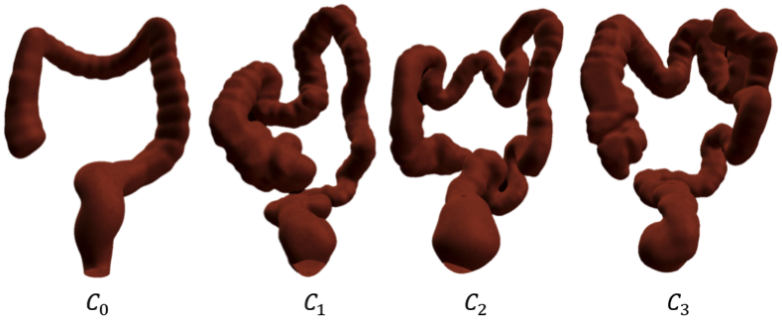


Fig. 4.7: Colon models used in the experimental phase. (From left to right) ranked in increasing complexity order, C_0, C_1, C_2, C_3 models. The model complexity is characterised by the centerline from rectum to caecum, and the number of acute bends, i.e.,>90 degrees, which is estimated through visual inspection.

The primary objective is to evaluate the differences in training the proposed CRL (L-PPO) and standard DRL (PPO) approaches. The following steps were used for evaluation.

- *(S1)*: 5000 policies were trained on the hardest colon model with different random initialization.
- *(S2)*: The best 300 policies were selected based on success rate during training, which is the number of times the agent successfully reaches the colon end in 100 consecutive trials while minimizing the number of collisions with the walls.
- *(S3)*: These 300 policies were evaluated on other colon models, and the navigation performance based on the average distance traveled by the scope on each colon model was recorded.
- *(S4)*: FV was performed on the 300 policies to obtain a policy that shows no safety violation for final deployment.

All data were obtained using an RTX 2070 and an i7-9700k.

While carrying out S3, in addition to the considered methods (i.e. L-PPO and PPO), we also include the results of the PPO_{lum} method in Table 4.1. PPO_{lum} represents the PPO baseline trained using the lumen centralization reward function

proposed in prior research by [215]. The average distance travelled is a crucial factor in evaluating trajectories since multiple backward motions or reversing the direction of motion may lead to suboptimal trajectories. The measurement of distance travelled utilizes position values of the endoscope tip normalized by the centerline distance of the colon model.

	Colon C_0	**Colon C_1**	**Colon C_2**	**Colon C_3**
PPO$_{lum}$ [215]	**0.84**	0.85	0.97	0.92
PPO	0.86	0.92	0.99	0.91
L-PPO	0.88	**0.81**	**0.92**	**0.84**

Table 4.1: Average distance traveled results.

Training results

The learning curves of PPO and L-PPO are presented in Fig. 4.8 left, demonstrating a comparable performance between the two algorithms, with both reaching higher reward values at approximately 400 episodes. Our analysis reveals that L-PPO effectively enforces constraints by maintaining a constraint cost below the limit value at 300 episodes, while PPO's constraint cost remains above the limit.

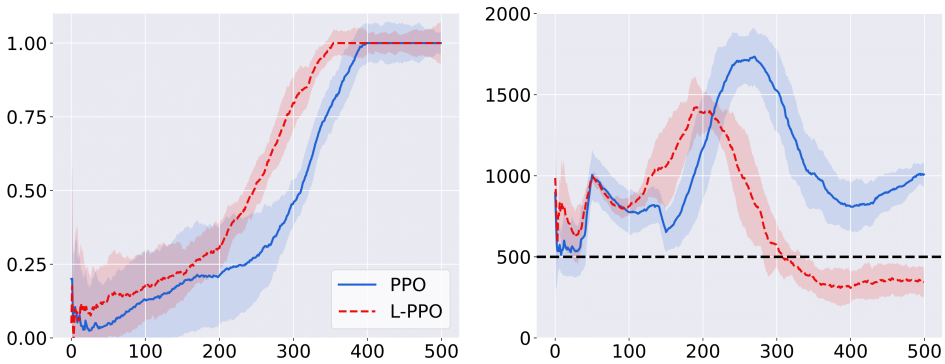


Fig. 4.8: Average performance vs the number of episodes of PPO and L-PPO over ten seeds. On the left, we report the expected returns while, on the right, the cumulative cost. Solid blue and red dashed lines are the empirical mean, while shaded regions represent the standard deviation. The black dashed line indicates the cost threshold.

Note that a single collision can produce a large number of interactions, depending on the number of timesteps the agent stays in contact with the wall. These results suggest that L-PPO can achieve better constraint satisfaction on average than PPO. Our examination further demonstrates that both PPO and L-PPO achieve a 100% success rate in navigating all colon models by reaching the end of the colon. We observe an equivalent performance among all three algorithms, indicating that DRL can be trained without a lumen centralization reward, over a global objective of reaching the colon end. The results in Table 4.1 indicate that all three algorithms follow a

path shorter than the centerline. L-PPO shows the shortest path for Colon 1, 2, and 3. As for Colon 0, which represents a simple scenario, all three algorithms perform well, making it difficult to determine the cause of L-PPO's lower performance on Colon 0.

Formal verification results

To address Q2, FV is conducted on the 300 policies trained using each methodology. Table 4.2 provides the violations for PPO and L-PPO across all four safety properties. Specifically, for each safety property, we report the SAT values indicating the number of models that violate that particular property. Notably, we observe that for the first safety property $\Theta_{\uparrow}$, which pertains to the situation where the upper part of the image is very bright and does not require an upward action from the agent, all 300 PPO policies violate the safety property. The observed violation is not straightforward to interpret, and it may be attributed to the infrequent exposure of the agent to such setups during the training process. It is plausible to suggest that the lack of sufficient training data for these specific scenarios may have hindered the agent's ability to learn the corresponding actions that adhere to the prescribed safety property.

	Safety Properties				
	$\Theta_{\uparrow}$	$\Theta_{\downarrow}$	$\Theta_{\leftarrow}$	$\Theta_{\rightarrow}$	**Model Selection**
Method	SAT	SAT	SAT	SAT	*Completely safe model*
PPO	300	246	80	167	0
L-PPO	221	198	53	161	**3**

Table 4.2: Results of model selection. SAT indicates property violation.

According to the results reported in Table 4.2, L-PPO has fewer violations than PPO, confirming that incorporating soft constraints in training has a direct impact on decreasing violations of hard constraints. We show the positions of the hard constraints violation of PPO on one of the colon models in Fig. 4.9c. As expected, large proportions of violations take place at sharp bends, which are the critical points for the correct execution of the colonoscopic procedure.

This analysis sought to ascertain if it is feasible to identify a policy that adheres to all the hard constraints. As Table 4.2 attests, three models satisfying all the hard constraints were identified in the case of L-PPO, while no policies conforming to the same standards were observed in the case of PPO, demonstrating the efficacy of the framework proposed herein. It is noteworthy that the L-PPO utilized in prior experiments (for example, in Table 4.1) is one of the three safe policies.

In order to emphasize the vulnerability of DNNs and the necessity of using FV in these safety-critical scenarios, we can consider Fig. 4.9. This figure displays the results of the analysis on a policy trained with PPO. Fig. 4.9b shows an input on which the tested model acts safely, without violating the property, while with the same property $\Theta_{\uparrow}$ in Fig. 4.9a, an adversarial input is discovered by the formal verifier. It is clear that the input only differs by 0.18 in the 6th value, yet this insignificant alteration causes the network to output a secure action in one case and a potentially dangerous action in the other.

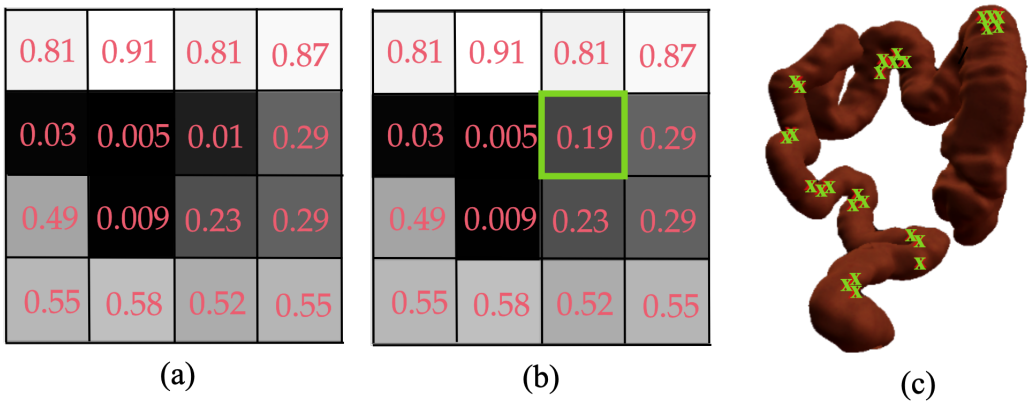


Fig. 4.9: (a) Adversarial example discovered with FV for the safety property $\Theta_{\uparrow}$. (b) A small perturbation in the square marked green, the agent shows safe behavior. (c) Hard constraint violation positions for one of the PPO policies are marked with green crosses.

Summary. In this section, we investigated the challenges associated with the deployment of DRL approaches in safety-critical scenarios such as autonomous colonoscopy navigation in a virtual simulation. DRL-based methods have demonstrated the ability to successfully traverse patient-specific colon models with comparable performance to that of expert clinicians [215]. Nevertheless, these methods are susceptible to adversarial attacks, which could result in safety violations with potentially fatal consequences. Consequently, we exploit a constrained reinforcement learning approach that ensures soft safety constraints through a cost function of safety violations. However, enforcing hard constraints through this methodology is not feasible, and even imposing stringent cost thresholds for soft constraints in CRL-based approaches often leads to suboptimal behaviors that ultimately fail to satisfy the intended behavioral preferences. As a first solution to this challenging problem, we proposed a model selection strategy that exploits formal verification of deep neural networks to evaluate the safety of a vast pool of policies trained using CRL. From the 300 policies trained using CRL, we identified three policies that adhered to all safety constraints, compared to no policies that met the same criterion for standard DRL. In the next section, we explore an alternative unconstrained approach that integrates probabilistic quantitative verification (introduced in the previous chapter) directly into the DRL training loop to further enhance the safety of intelligent systems.

4.2 Verifying Online Safety Properties for Safe Deep Reinforcement Learning

As shown in the previous section, deep reinforcement learning algorithms have demonstrated promising results in various applications, to medical robotic, manipulation tasks [98, 171], video games [276] and mobile navigation [167, 249, 166]. However, using these deep neural networks-based systems on real-world problems requires incorporating safety into DRL methods.

Safe DRL approaches have been explored to enhance safety by modeling a problem as a constrained Markov decision process [6]. As previously stated, in a CMDP, an agent aims to maximize a reward signal while keeping an accumulated cost, received from visiting unsafe states, below a specified threshold. However, defining informative cost functions is as challenging as designing rewards [104] due to the difficulty of quantifying the risk of policy decisions around unsafe states. Consequently, the proposed solutions rely on indicator cost functions where a positive value marks a state-action pair (s, a) as unsafe [6]. These cost values are then backed up to propagate safety information and estimate a cost-value function, which guides the learning process toward safety via penalties [251] and constraints [243, 152]. However, learning cost-value functions with DNNs introduces brittle convergence properties and local optima [109, 163], limiting sample efficiency and the efficacy of cost functions in promoting safety. The sparse nature of the indicator cost also poses a significant issue, requiring numerous unsafe state visits to learn reasonable estimates from sparse feedback. Moreover, these sparse values also fail to provide information about the likelihood of incurring an unsafe behavior for a specific action a around s. Consider robotic navigation as an explanatory domain. In this task, the RL policy determines the agent's velocity based on sensor data, and prior works trigger a positive cost when an action results in a collision (Fig. 4.10 on the left), deeming the state-action pair as unsafe. These methods thus necessitate many similar unsafe interactions around s to approximate the cost-value function effectively [152, 243, 47].

On top of these issues, DNNs are also vulnerable to small input variations that can fool a network to output an undesired value (or action) [246]. Although these inputs are most commonly observed in high-dimensional visual domains, [8] shows that they can also occur in low-dimensional state configurations like DRL navigation tasks, leading to unsafe behaviors and potential hazards. Formal verification for

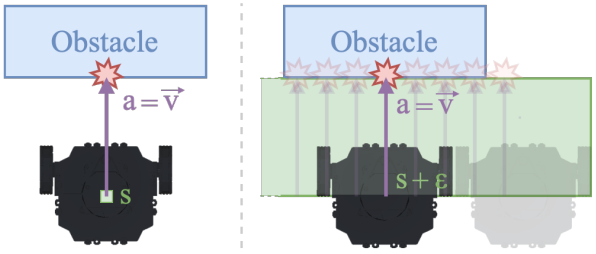


Fig. 4.10: Indicator cost function (left). Unsafe interactions are caused by the same action around the unsafe state (right).

DNNs [150] has arisen to tackle this issue, leveraging state-action relationships (called *safety properties*) to provably detect these configurations. However, applying FV during training presents many challenges:

1. the *safety properties* are hand-designed by a system designer, which may be unfeasible in complex tasks;
2. FV is NP-complete and thus computationally demanding [131]

These challenges raise the question:

How can we enhance safety during training without relying on cost functions?

We address this question by proposing a novel approach that leverages online safety properties, i.e., collected during training, to (i) get information on which part of the state space is potentially unsafe, and (ii) compute the probability of incurring in one of these configurations in the surrounding of that state. We then use this probability as a safety metric, called *violation*, as a penalty to the agent for discouraging unsafe decisions [1, 149].

We hypothesize that this procedure will significantly improve safety, reducing the number of hazardous states visited and addressing all the issues related to cost functions. Recalling the navigation example, let us consider a state s where performing an action a led to a collision. The proposed approach exploits state-action information (e.g., robot size, max velocity) to define an area of a specified size around s where performing a would likely result in unsafe interactions (Fig. 4.10 on the right). Our goal is to quantify the states in the unsafe area where the policy chooses a and use this information as a penalty to bias the policy towards safer regions.

To this end, in this section we present an integrated and extended approach, building on two of our previous works, to bridge the gap between formal verification and safe DRL:

- Collection and refinement of online properties (CROP) framework [174] for collecting safety properties during training without relying on a system designer. CROP also refines *similar* properties to limit their number.
- Monte Carlo sample-based verification [168] that samples and propagates a set of states from the ω area of interest to approximate the violation value by counting the instances where the agent selects an unsafe action.

In detail, we significantly extend these contributions by providing probabilistic guarantees on the approximate violation value. We also analyze the non-Markovian nature of the violation penalty, presenting a state-augmented method to address such an issue. Additionally, we expand the empirical analysis by evaluating our framework on standard safe DRL benchmarks (i.e., SafeMuJoCo and SafetyGym [222]), and additional baselines. Finally, we transfer our trained policies to real robots in a navigation task [300, 166].

In all scenarios, we apply the violation penalty to the well-known Proximal Policy Optimization (PPO) algorithm [232], comparing to a cost-penalty version [276, 232], a constrained implementation based on the Lagrangian method (LPPO) [243], a Lagrangian-based penalty function (RCPO) [251], and the more recent penalty-based algorithm Penalized PPO (P3O) [301]. Our extensive evaluation shows that

cost penalties result in high costs and violations at convergence, confirming the limitations of indicator cost functions. Lagrangian and penalty-based baselines exhibit similar issues due to using additional cost-value functions. In contrast, our violation-based penalty significantly reduces unsafe behaviors (i.e., lower cost and violation) while preserving high returns. To further assess the improvement in terms of safety provided by our proposed approach over different methods tested in this work, we compare the violation value for converged policies using a formal verification tool. Our results confirm the safety benefit, resulting in the lowest violation value percentage, confirming the impact of incorporating an approximate violation value as a penalty for the trained agents.

4.2.1 Safety Properties and Formal Verification

We now revisit the definition of the safety property used in the context of robotic navigation tasks, and specifically in this section. This definition is consistent with the one presented earlier in the thesis, with only minor modifications.

A safety property for a DNN (f) is typically expressed as a state-action relationship [150]. Specifically, as presented early, the property is encoded using a precondition and a postcondition. The precondition is a predicate on the state features modeling desired unsafe situations (i.e., the domain of the property). It is used to check if a specific state falls within a set of intervals $\mathcal{X}$ representing the unsafe area of interest. Thus, $\mathcal{X}$ is a Cartesian product of intervals, one for each feature in the state space. The postcondition is a predicate $\mathcal{Y}$ on the output of f specifying the action to avoid in $\mathcal{X}$. Broadly speaking, the postcondition $\mathcal{Y}$ encodes a requirement such as *never select the action corresponding to the output* y_i, with i being the index of the action to avoid.

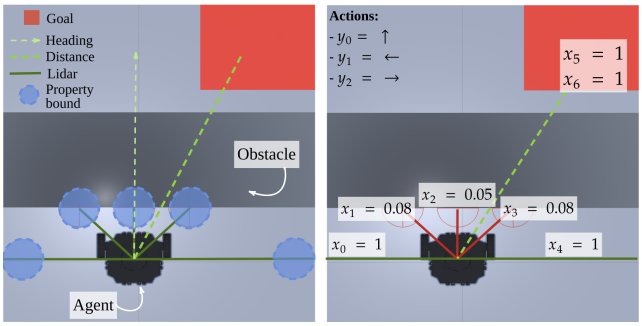


Fig. 4.11: Left: components of a safety property (legend in the upper left corner). Right: Explanatory image of a safety property for a navigation context.

Using the navigation task, Fig. 4.11 shows an explanatory encoding of a safety property in a value-based DRL setup, where the policy selects the action with the highest value.[2] To encode a property, we start by considering the current state of the agent as a vector of observed features $x_0, \ldots, x_6$, and its possible actions $\{y_0, y_1, y_2\}$

[2] Similar considerations apply to the continuous policy-gradient case by using deterministic policies and checking the action values, and not their max.

corresponding to a forward, left, and right movement. Given there is an obstacle in front of the robot, the safety property uses an $\boldsymbol{\omega} = 0.05$ surrounding of the agent's state (the blue bounds on the left figure) to model the set of intervals $\mathcal{X}$ representing the unsafe area (i.e., where a forward action would lead to a collision). In this situation, we want to check that a forward movement is not selected, so a safety property is formally encoded as follows:

$$\underbrace{\forall\, x_0, x_4 \in [0.95, 1], x_1, x_3 \in [0.03, 0.08], x_2 \in (0, 0.05], x_5 \in (0, 1], x_6 \in [-1, 1],}_{\mathcal{X}}$$

$$\underbrace{\max(y_1, y_2) > y_0,}_{\mathcal{Y}}$$

where $x_0, \ldots, x_4$ are the 5 lidar values, x_5, x_6 is the relative position of the agent with respect to the goal (i.e., distance and heading), and action y_0 corresponds to the forward movement. In the example, we set $\mathcal{X}_5 = (0, 1], \mathcal{X}_6 = [-1, 1]$ to indicate a generic goal position in the environment. Hence, if the agent is in any state that satisfies $\mathcal{X}$, i.e., $\forall\, \mathbf{x} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6]^{\mathrm{T}} \in \mathcal{X}$ (with $\mathcal{X} = \{[0.95, 1], [0.03, 0.08], (0, 0.05], [0.03, 0.08], [0.95, 1], (0, 1], [-1, 1]\}$), if $x_0 \in [0.95, 1], x_1 \in [0.03, 0.08], x_2 \in (0, 0.05], x_3 \in [0.03, 0.08], x_4 \in [0.95, 1], x_5 \in (0, 1], x_6 \in [-1, 1]$, it should not select the forward action y_0.

#DNN-Verification and the Violation Value

Since our goal is to quantify the probability of unsafe policy decisions within a specific region of the state space, the #DNN-Verification problem, introduced in Sec. 3.3, is well suited for this purpose. For clarity, we restate the problem definition here.

Definition 4.1 (*#DNN-Verification problem*):
Input: *A tuple* $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$.
Output: $|\Gamma(\mathcal{T})|$

with $\Gamma(\mathcal{T}) = \left\{ \boldsymbol{x} \;\middle|\; \mathcal{X}(\boldsymbol{x}) \wedge \mathcal{Y}(f(\boldsymbol{x})) \right\}$ representing the set of all the input configurations for f satisfying the property defined by $\mathcal{X}$ and $\mathcal{Y}$ (i.e., all the unsafe state-action pairs). Since our setting is in the continuum, the number of points in any non-empty set is infinite. Hence, we consider the cardinality as a proxy for the volume of the corresponding set. Nonetheless, if we discretize the space to the machine precision, $\Gamma(\mathcal{T})$ becomes a finite countable set. In practice, rather than the cardinality of $\Gamma(\mathcal{T})$, it is more useful to define the problem in terms of the ratio between the cardinality of Γ and the one of the set of inputs satisfying $\mathcal{X}$. This quantity is called *violation* and it is formally defined as follows:

Definition 4.2 (*Violation value*): *Given an instance of #DNN-Verification* $\mathcal{T} = \langle f, \mathcal{X}, \mathcal{Y} \rangle$,

$$violation = \frac{|\Gamma(\mathcal{T})|}{|\{\boldsymbol{x} \mid \mathcal{X}(\boldsymbol{x})\}|}.$$

Here, we remark that DNN-Verification is an NP-complete problem [131]. Moreover, formally counting (or enumerating) all violation points in an area of interest is even more challenging, since, as we demonstrate in Sec. 3.3 it is a #P-hard problem. Hence, these methods are unfeasible to be applied during training and motivate the investigation of approximate solutions.

4.2.2 Safe Deep Reinforcement Learning via Probabilistic Verification

In this section, we present our two main contributions addressing the limitations of FV methods discussed above, namely: (i) the complexity of hand-design safety properties and (ii) the computational overhead of computing the violation value. To tackle the first issue, we propose the collection and refinement of online properties framework. To address the latter, we introduce a Monte Carlo sampling approach that approximates the violation value with probabilistic guarantees, providing an efficient alternative to the "formal" violation value computed using modern FV tools.

Collection and Refinement of Online Properties

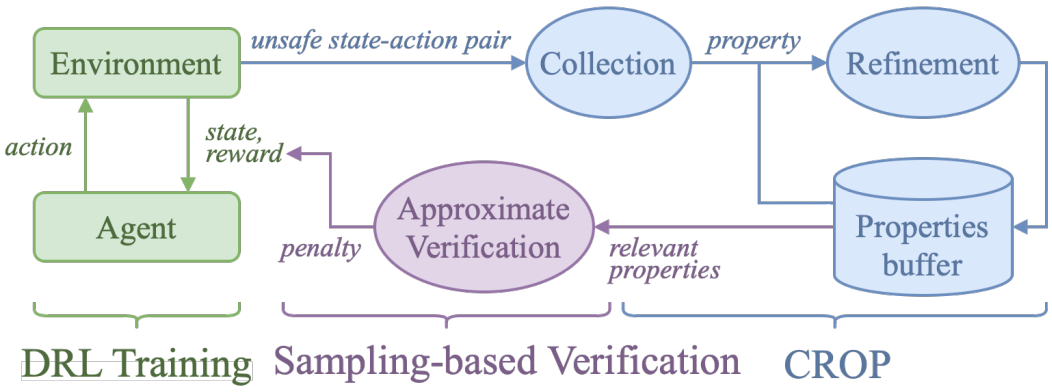


Fig. 4.12: Overview of the CROP framework.

Fig. 4.12 shows an overview of the proposed CROP framework (in blue), which is designed to collect and refine safety properties during training. Similar to existing safe DRL methods [152, 243, 226], CROP uses a cost indicator function to identify unsafe state-action interactions. When such an interaction is detected, we generate a safety property using the state and the action that led to the hazardous situation. CROP then performs a refinement procedure merging properties deemed similar according to a similarity rule, which is described in the following section. Finally, the violation value is computed using the sample-based approximation described in the next section, applied to the properties generated by CROP.

The general flow of CROP is presented in Algorithm 11.

We augment the training of a DRL algorithm with a buffer $\mathbf{P} = \{properties\ \mathcal{P}\}$ that stores the properties generated by CROP. Given a state s_t deemed unsafe by the cost signal, CROP generates a new safety property $\mathcal{P}'$ considering an initial $\boldsymbol{\omega}$ surrounding (passed as a parameter) of the state s_{t-1} and the action a_{t-1} that triggered the positive cost. In particular, we model the precondition $\mathcal{X}$ of the safety

Algorithm 11: Collection and Refinement of Online Properties

Given:
- $f \leftarrow$ a DRL agent parametrized by a DNN
- $\boldsymbol{\omega} \leftarrow$ initial size for the area surrounding an unsafe state
- $\beta \leftarrow$ similarity threshold to merge properties.
- $d_{\text{unsafe}} \leftarrow$ minimum interval that encodes a potentially unsafe situation.

During each episode of the training of DRL agent:
$\mathbf{P} \leftarrow \emptyset$
if s_t is unsafe (i.e., cost > 0) **then**
  $\mathcal{P}' \leftarrow$ `GenerateProperty`$(\boldsymbol{\omega},\ s_{t-1}, a_{t-1})$
  for $\mathcal{P} \in \mathbf{P}$ where $s_{t-1} \subseteq \mathcal{X} \wedge a_{t-1}$ is unsafe in $\mathcal{Q}$ of $\mathcal{P}$ **do**
    if $\nexists$ `Similar`$(\mathcal{P}, \mathcal{P}', \beta, d_{\text{unsafe}})\ \forall \mathcal{P} \in \mathbf{P}$ **then**
      $\mathbf{P} \leftarrow \mathbf{P} \cup \mathcal{P}'$
    else
      $\mathbf{P} \leftarrow$ `PropertyRefinement`$(\mathcal{P}, \mathcal{P}', \mathbf{P})$
  violation $\leftarrow$ `ApproximateVerification`$(f, \mathbf{P})$ ▷ as in Sec. 4.2.2

property with a set $\mathcal{X}$ of intervals encoded as $[x_i - \boldsymbol{\omega},\ x_i] \quad \forall\ x_i \in \mathbf{x}$, where x_i is the i-th input feature of the state s at timestep $t-1$. Notably, we encode the interval $[x_i - \boldsymbol{\omega},\ x_i]$ and not $[x_i - \boldsymbol{\omega},\ x_i + \boldsymbol{\omega}]$ to exclude potentially unfeasible configurations The postcondition $\mathcal{Q}$ checks that the agent does not select the action a_{t-1} that led to the unsafe state s_t. CROP then samples the properties whose set $\mathcal{X}$ contains the state s_{t-1}, and whose postcondition $\mathcal{Q}$ considers the same unsafe action a_{t-1} that triggered the positive cost. For each selected property, we check if the new property $\mathcal{P}'$ is similar to a sampled one $\mathcal{P}$ that will be refined as described in the following section. Notably, the $\boldsymbol{\omega}$ employed by CROP differs from the ω value typically required to design a safety property (as discussed in Sec. 1), which requires some knowledge of the task or agent's hardware. CROP's $\boldsymbol{\omega}$ is initialized as an arbitrarily small value that is reshaped during refinement. Moreover, we note in domains where many unsafe interactions occur, the buffer $\mathbf{P}$ grows unbounded. Hence, we reset $\mathbf{P}$ at the beginning of each episode. This approach allows us to compute the violation value only on unsafe situations encountered over the last epoch.

Similarity Rule and Refinement.

Different safety properties could model a similar unsafe interaction (or not). We define a similarity rule to identify when two safety properties are *not similar*. To clarify the process, let us recall our navigation example. Consider two sets $\mathcal{X} = \{[0.95, 1], [0.95, 1], [0.01, 0.06], [0.03, 0.08], [0.95, 1]\}$, $\mathcal{X}' = \{[0.04, 0.09], (0, 0.05], [0.95, 1],\ [0.95, 1], [0.95, 1]\}$ encoding the preconditions of two safety properties $\mathcal{P}$ and $\mathcal{P}'$, respectively.[3] These scenarios are shown in Fig. 4.13, where the agent would collide upon moving forward (i.e., the two safety properties have the same postcondition). The two preconditions significantly differ. Let us compare the first two intervals of $\mathcal{X}$ and $\mathcal{X}'$— $\mathcal{X}_0 = [0.95, 1]$ and $\mathcal{X}'_0 = [0.04, 0.09]$—as

[3] For simplicity, we omit the values for the goal's heading and distance.

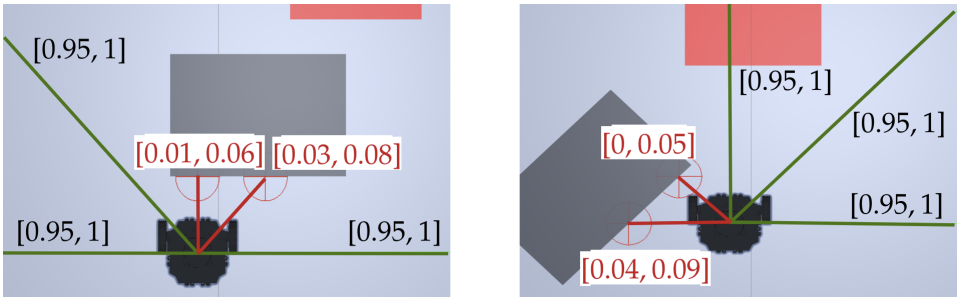
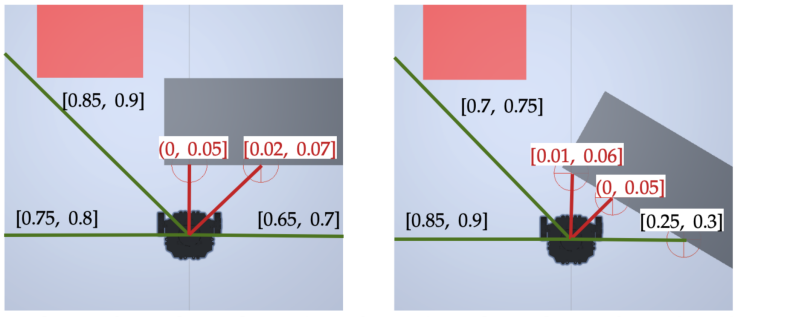


Fig. 4.13: Explanatory example of two different safety properties for the same forward action.

a practical example. We formally define the two properties to be *not similar* by leveraging Moore's interval algebra [196] and defining the following similarity rule:

$$\mathcal{P} \nsim \mathcal{P}' \iff \exists\, i \in \{0, |\mathcal{X}|\}\ s.t.\ \left(\mathcal{X}_i \subseteq d_{\text{unsafe}} \vee \mathcal{X}'_i \subseteq d_{\text{unsafe}}\right) \wedge |\mathcal{X}_i - \mathcal{X}'_i| > \beta.$$

Specifically, given the two safety properties $\mathcal{P}$ and $\mathcal{P}'$, we check: (i) whether there is at least one interval in the sets $\mathcal{X}, \mathcal{X}'$ contained in d_{unsafe}. d_{unsafe} is the smallest interval encoding a potentially unsafe situation (e.g., in Fig. 4.13, $d_{\text{unsafe}} = [0, 0.09]$ and is highlighted in red); (ii) whether the lower bound of the absolute value difference between the two intervals $|\mathcal{X}_i - \mathcal{X}'_i|$ is greater than a similarity threshold value β.[4] If the two conditions hold for at least one interval, then $\mathcal{P}$ and $\mathcal{P}'$ are deemed not similar. On the other hand, if two properties are *similar*, we perform a refinement process merging them by considering the minimum lower and maximum upper bound for each interval of $\mathcal{X}$ and $\mathcal{X}'$. The refinement enables us to cluster together similar unsafe situations. Fig. 4.14 shows an explanatory refinement process in a scenario considering the forward action y_0.



$\mathcal{P}_{\uparrow} : \forall x_0 \in [0.75, 0.9], x_1 \in [0.7, 0.9], x_2 \in (0, 0.06], x_3 \in (0, 0.07], x_4 \in [0.25, 0.7],\ \max(y_1, y_2) > y_0$

Fig. 4.14: Explanatory example of refinement process for two *similar* safety properties.

[4] β is a hyperparameter that can be tuned to be more or less restrictive in considering two properties as similar.

Monte Carlo Sampling-based Approximate Verification

The violation value computed on a safety property quantifies the probability of incurring unsafe policy decisions within an area of the state space, addressing the sparsity of indicator cost functions (Fig. 4.12 in purple). However, due to the NP-completeness and #P-hardness of the verification problem [131, 173], computing a formal violation (i.e., using a FV tool) during training is intractable. Therefore, we propose a computationally efficient sample-based method to approximate the violation value, allowing us to use it as a penalty during training. Similar to the violation computed with formal tools, our approximation can be interpreted as a locally-aware (dense) cost value. It quantifies the risk of performing a certain action a in a state s, based on the hazardous effects that a could have in a local region around s. More formally, consider a deep neural network f with $\{y_1, \ldots, y_n\}$ outputs representing the possible actions the agent can perform in the environment. Let y_k be the unsafe action to avoid in a particular region of interest $\mathcal{X}$ specified by the precondition $\mathcal{X}$. Our goal is to count the number of times $y_k > \max(y_i)\ \forall i \in \{1, \ldots, n\} \setminus \{k\}$ over the total number of states tested.

This approximation offers several advantages over indicator cost functions: it (i) provides safety information for areas of interest as it is computed over state-action mappings (i.e., the safety properties); (ii) approximates how often a property violation might occur, having a similar role to Lagrangian multipliers without requiring additional gradient steps or value estimators; (iii) does not require additional environment interactions, significantly reducing the number of visited unsafe states.

Algorithm 12 shows the general flow of our sample-based method. Our approach requires f, an unsafe state s, a property buffer $\mathbf{P}$, and the number of samples to use for computing the approximate violation value. The first step is to consider the safety properties that contain s in their preconditions, and store these properties in a new buffer $\mathbf{P'}$.

We initialize the violation value to zero, and for each $\mathcal{P} \in \mathbf{P'}$ we randomly sample a set of m input vectors for the DNN $\mathbf{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_m}\}$ from $\mathcal{X}$. After propagating $\mathbf{X}$ through f, we count how many outputs do not satisfy the postconditions. To this end, for each $\mathbf{x_i}$, we check if the index of the corresponding output (i.e., the one with the highest value) is equal to the index k of the unsafe action to avoid. Finally, we compute our approximate violation value as the ratio between the number of such outputs over the total sampled states. This process closely resembles how the formal

Algorithm 12: Computing the Approximate Violation Value

Given f with outputs $y_1, \ldots, y_n$, unsafe state s, properties buffer $\mathbf{P}$, and m number of states to sample.

$\mathbf{P'} \leftarrow \langle \mathcal{X}, \mathcal{Y} \rangle$ if $s \cap \mathcal{X} \neq \emptyset,\ \forall \langle \mathcal{X}, \mathcal{Y} \rangle \in \mathbf{P}$
violation $\leftarrow 0$
for $\mathcal{P} \in \mathbf{P'}$ **do**
    $\mathbf{X} \leftarrow$ `Sample`$(m, \mathcal{X})$
    violation $\leftarrow$ *violation* + `count`$(\text{argmax}(f(\mathbf{X})) = k)$
return *violation* / $(m\ |\mathbf{P'}|)$

violation is computed in Sec.3.3 but employs random sampling and feed-forward steps through the f, avoiding the computational demands associated with FV tools.

Visual Example.

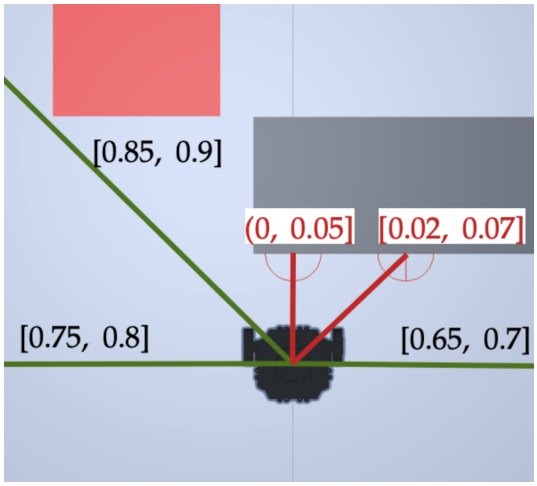


Fig. 4.15: Illustrative example of a potential unsafe scenario for a DRL mapless navigation task.

We further clarify the approximate violation value computation by relying on our navigation example in Fig. 4.15. The agent receives 5 lidar scan values as input and outputs three values $\{y_1, y_2, y_3\}$ corresponding to a forward, left, and right movement, respectively. We thus have $\mathcal{X} = \{[0.75, 0.8], [0.85, 0.9], (0, 0.05], [0.02, 0.07], [0.65, 0.7]\}$ encoding the unsafe area where we want to avoid the action y_1 that has index 0 (i.e., a forward movement at velocity $\vec{v}$). Hence, given any state sampled in $\mathcal{X}$, the action y_1 should not be selected. This safety property is formally defined as $\forall \mathbf{x} \in \mathcal{X}$, $\max(y_2, y_3) > y_1$. Considering a sample of $m = 2$ states from the $\mathcal{X}$ of the property precondition, we obtain two states $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2\} \in \mathcal{X}$ which are then forward propagated through f. If this propagation outputs $f(\mathbf{x}_1) = [1, 3, 2]$, then the greedy agent selects $\text{argmax}(f(\mathbf{x}_1)) = 1$. Conversely, for $\mathbf{x}_2$ we obtain $f(\mathbf{x}_2) = [4, 2, 3]$, and the agent selects $\text{argmax}(f(\mathbf{x}_2)) = 0$. To approximate the violation value, we count the states $\mathbf{x} \in \mathcal{X}$ where the argmax of $f(\mathbf{x})$ is equal to the index of the unsafe action y_k, in this case 0. In this example, we have one violation since only the second propagation of $\mathbf{x}_2$ leads to $\text{argmax}[f(\mathbf{x}_2)] = 0$. By normalizing the number of violations over the size of the sampled configurations, we obtain the violation value of 0.5, meaning that half of all the states sampled led to a property violation.

A natural question that arises is what number of states $\mathbf{x}_m$ is needed to get an accurate estimate of the violation and what kind of guarantees we obtain about this estimate.

Probabilistic Guarantees on the Violation Value

We aim to estimate the number of states where a binary property holds (i.e., whether the agent commits a violation or not) from a potentially infinite set of states. To

this end, we can sample a smaller subset of the states and show that this particular subset is representative of the larger one we are interested in. To this end, we use the following Chernoff bound.

Theorem 4.1 (Chernoff bound): *Let $X_1, \cdots, X_m$ be independent indicator random variables, such that $Pr[X_i = 1] = p_i$. Let $X = \sum_{i=1}^{m} X_i$ and $\mu = \mathbb{E}[X] = \sum_{i=1}^{m} p_i$. For $0 < \epsilon < 1$ the following inequality holds:*

$$Pr[|X - \mu| \geq \epsilon\mu] \leq 2e^{-\frac{\mu\epsilon^2}{3}}.$$

A direct application of Theorem 4.1 implies the following result (see, e.g., [194, section 4.2.1]), that we recall here to keep the section self-contained.

Let p be the actual violation value in the area $\mathcal{X}$ that encodes the property's precondition. Assume we would like to estimate p up to an additive error-tolerance θ and with confidence at least $1 - \delta$, for fixed (desired) $\delta, \theta \in (0, 1)$. To this end, we sample m points $\mathbf{x_1}, \ldots, \mathbf{x_m}$ independently and uniformly from the property input space $\mathcal{X}$. For each $i = 1, \ldots, m$, let X_i be the indicator random variable of the event that $\mathbf{x}_i$ is a violation point, hence $Pr[X_i = 1] = p$. Define $\overline{X} = \frac{\sum_{i=1}^{m} X_i}{m}$. We are interested in a bound on m that can guarantee $Pr[|\overline{X} - p| \leq \theta] \geq 1 - \delta$.

Let $X = \sum_{i=1}^{m} X_i$, hence $\mu = \mathbb{E}[X] = mp$. Considering that $p \leq 1$, by Theorem 4.1 we get

$$\begin{aligned} Pr[|\overline{X} - p| \geq \theta] &= Pr[|X - \mu| \geq m\theta] \\ &= Pr[|X - \mu| \geq \mu\frac{\theta}{p}] \leq 2e^{-\frac{\mu\theta^2}{3p^2}} = 2e^{-\frac{m\theta^2}{3p}} \leq 2e^{-\frac{m\theta^2}{3}}. \end{aligned}$$

Therefore, for

$$m = \frac{3}{\theta^2} \ln\left(\frac{2}{\delta}\right), \tag{4.6}$$

we have

$$Pr[|\overline{X} - p| \geq \theta] \leq \delta.$$

Recalling that we are interested in the complementary event of the term on the left-hand side of the latter inequality, the above derivation leads to the following result.

Proposition 4.1: *Fix $\theta, \delta \in (0, 1)$, and let m be an integer satisfying $m \geq \frac{3}{\theta^2} \ln(\frac{2}{\delta})$. Let $X_1, \cdots, X_m$ be independent and identically distributed indicator random variables for the event that a uniformly sampled input point from the set $\mathcal{X}$ violates the property, and let $X = \sum_{i=1}^{m} X_i$, whence $Pr[X_i = 1] = p$, and $\mu = \mathbb{E}[X] = mp$. Then, $Pr[|\frac{X}{m} - p| \leq \theta] \geq 1 - \delta$.*

The proposition gives a lower bound on the number of m points in $S = \mathbf{x}_1, \ldots, \mathbf{x}_m$ such that if V are violation points, $|V|/m$ provides an estimate of p with the desired precision θ and confidence δ. For example, if we want a maximum error tolerance of $\theta = 0.05$ with confidence at least $1 - \delta = 0.99$ for the approximate violation

we are computing, sampling $m \geq \frac{3}{\theta^2}\ln(\frac{2}{\delta}) = \frac{3}{0.05^2}\ln(\frac{2}{0.01}) = 6357$ states would be sufficient.

It is important to note that our approach is not efficient when the formal violation value we are trying to estimate is arbitrarily small. To clarify this, let M be the size of the instance (the network's encoding and the property where we want to compute the violation value). If $p \geq \frac{1}{poly(M)}$ (i.e., bounded by some polynomial function of the instance size), the approach is clearly a polynomial time algorithm in M as it requires a number of samples polynomial in the size of the instance. On the other hand, if, for instance, $p = o(2^{-M})$, then our sampling method would require an exponential number of samples. However, in view of the hardness of both the FV problem and its counting version, under the standard complexity assumptions, we believe this intractability issue is strictly related to the inherent complexity of the problem rather than a weakness of our approach. From (4.6), we see that for any fixed m, there is a direct proportionality between the error tolerance θ and the values of the confidence $c = 1 - \delta$, that is, $\theta = \sqrt{\frac{3}{m}\ln\left(\frac{2}{1-c}\right)}$. Hence, for any fixed m, the better (higher) the confidence $c = 1 - \delta$, the looser (higher) the error tolerance bound θ. Fig. 4.16 visualizes this relationship. From the curves, we note that using $m = 10000$ samples is sufficient to guarantee confidence ≥ 0.99 and an error tolerance below 0.05.

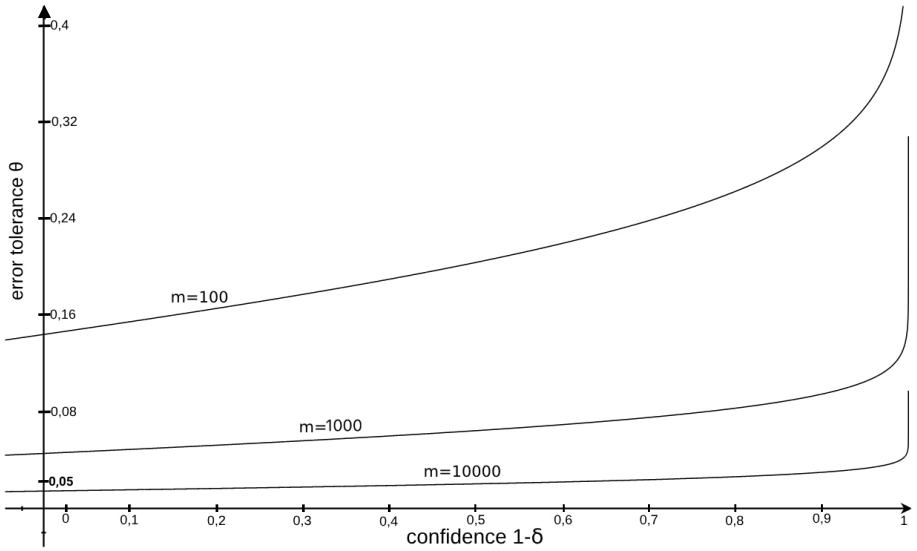


Fig. 4.16: The relation between confidence (x-axis) and error tolerance (y-axis) for different values of $m = 100, 1000, 10000$.

Comparing the Approximate Violation with the Formal One

To empirically assess how good our approximate violation is, we compare it over the exact violation computed by ProVe, a FV tool for the #DNN-Verification.[5]

To this end, we consider the same navigation task used in our previous examples, the average results of ten models collected at random steps during the training, and the following set of hand-designed properties[6]

- $\mathcal{P}_{\uparrow}$: *There is an obstacle close in front* $\Rightarrow$ *Do not go forward*
- $\mathcal{P}_{\leftarrow}$: *There is an obstacle close to the left* $\Rightarrow$ *Do not turn left*
- $\mathcal{P}_{\rightarrow}$: *There is an obstacle close to the right* $\Rightarrow$ *Do not turn right*

Table 4.3 shows a comparison between the violation values for $\mathcal{P}_{\uparrow,\leftarrow,\rightarrow}$ computed by ProVe, and with our approximation with samples of size 100, 1000, and 10000.

In practice, we measure the difference between the formal violation value (i.e., first column) and our best approximation achieved with $m = 10000$ states (i.e., last column). These violation values are computed using the three safety properties, and result in a negligible 0.47% difference. Hence, our sample-based method achieves a very accurate estimation of the real mean violation value.

Moreover, by exploiting parallelism and batch computation of modern deep learning frameworks, the increase in computation time for the approximate violation with 100 or 10000 samples is comparable. Conversely, ProVe's average computation time is orders of magnitude higher compared to our approach (i.e., 0.06 over 157 seconds).

Property	**ProVe**	**Estimation 100**	**Estimation 1k**	**Estimation 10k**
$\mathcal{P}_{\uparrow}$	81.48 ± 1.2	81.0 ± 0.6	81.1 ± 0.8	81.17 ± 0.5
$\mathcal{P}_{\leftarrow}$	73.9 ± 0.8	73.5 ± 0.2	73.63 ± 0.2	73.65 ± 0.3
$\mathcal{P}_{\rightarrow}$	74.2 ± 0.3	73.6 ± 0.1	73.67 ± 0.1	73.68 ± 0.1
Mean violation:	**76.53**	76.00	76.13	**76.17**
Mean computation time:	**≈2m37s**	**≈0.053s**	**≈ 0.056s**	**≈0.060s**

Table 4.3: Average *violation* (%), and computation time for properties $\mathcal{P}_{\uparrow,\leftarrow,\rightarrow}$ calculated using ProVe, and our approximation with 100, 1000, and 10000 samples.

[5] ProVe is available at `https://github.com/Isla-lab/Neural-Network-Verifier`

[6] We use the maximum agent velocity to determine the size of the area around the (unsafe) states of interest (i.e., the $\boldsymbol{\omega}$-area). However, for simplicity, we describe these properties in natural language.

Employing the Violation Value to Foster Safety

Having an efficient way for collecting properties and computing a good approximation of the violation, we now investigate the benefits of using the violation as a penalty during training.

Following prior penalty-based approaches [77, 251], we maximize the following objective:

$$\max_{\pi \in \Pi} J_r^\pi := \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) - Z(\cdot) \right] \tag{4.7}$$

where $Z(\cdot)$ is a generic penalty function. For instance, in the simplest case, $Z(\cdot)$ could be expressed as $Z(s, a) = -1$ upon experiencing a collision, i.e., as a cost function. In our case, we employ a violation-penalty $Z_\pi(s_t \pm \boldsymbol{\omega}) = \mathbb{E}_{s \sim s_t \pm \boldsymbol{\omega}} \mathbb{E}_{a_t \sim \pi(s)} \mathbb{1}[a_t = a_k]$, where a_k is the unsafe action that led to a collision. Such a value indicates that the violation depends on the policy decisions in a proximity $\boldsymbol{\omega}$ of the state (i.e., the $\boldsymbol{\omega}$-area, or R). Equation 4.7 has two main benefits over other constrained DRL algorithms:

- Penalty-based objectives potentially maintain the same optimal policy of the underlying MDP as they do not constrain exploration nor reduce the space of feasible policies as constrained approaches [199].[7]
- Constrained DRL typically estimates an additional value function to propagate cost information, hindering their application with values that strictly depend on the current policy. This also requires visiting unsafe states to learn effective estimates for the sparse cost values. In contrast, penalty-based methods do not require learning additional value functions. Additionally, many DRL algorithms, such as PPO and RCPO, provide significant empirical evidence on the benefits of employing penalties over constraints [251].

4.2.3 Limitations

The proposed approach only considers vectorized inputs for computing the violation. While it is conceptually feasible to include different modalities (e.g., images, temporal sequences), doing so would require further research and experiments. In more detail, while related verification works [190, 235] have been previously employed in vision-based problems [198], they typically consider robustness analysis rather than verifying image-based control policies. This stems from the fact that it is not trivial to apply a perturbation that would change the state of the agent (i.e., what it sees) coherently without additional strong assumptions, such as having access to vision models of the world. For example, applying a perturbation to an Atari game taken as input by a policy would result in a noisy image, while verifying the policy behaviors requires encoding images of related state configurations. Hence, despite being conceptually feasible considering different modalities for control policies, applying the proposed approach to these settings is seldom straightforward and goes beyond the scope of this section. Another limitation of our approach is related to the fact that,

[7] This is not the case for navigation tasks as safe behaviors bias the policy to avoid obstacles.

while our framework addresses the issues of using indicator cost functions and constraints, it remains unclear how to provide provable safety guarantees in model-free, safe DRL training approaches. Such a lack of formal guarantees on the behavior of the policy during training includes our work, constrained DRL [2, 152, 293], and several other approaches summarized in [77]. As discussed in [109], using DNNs to approximate policies and values causes the method to diverge from the underlying theoretical framework. Nonetheless, we begin to address these key issues by employing approximate violation with probabilistic guarantees during training, and formal verification approaches at convergence to check the trained policy decisions over the properties of interest.

4.2.4 Empirical Evaluation

In this section, we propose an empirical evaluation to assess the benefits of employing our violation penalty for training DRL agents. To this end, we used a realistic set of navigation tasks for a Turtlebot3 mobile robot as well as standard safety benchmarks inspired to [222]. Our experiments aim to show the following: (i) The benefits of combining the reward signal with our provable safety metric to enhance the learning of safer behaviors at convergence. (ii) The advantages of using the proposed violation metric over indicator cost functions. (iii) The effectiveness of our additional investigations: using CROP over commonly employed hand-designed properties; relying on a state-augmentation method to deal with the issues related to the Markov property.
To assess our claims, the following plots consider a variety of metrics such as *success* as the number of goals reached, *cost* as the number of the agent collides with an obstacle, and finally, the violation at different stages of the training. All such metrics are averaged over the last 1000 steps. Each methodology uses CROP to collect online properties during training to compare the violation metric. The average violation is thus calculated on the properties collected by each methodology (but clearly, we used the violation value as a penalty only in our approach).

The data are collected on an i7-9700k and consider the mean and standard deviation of ten independent runs [50]. We consider the cost and violation penalty objective (Equation 4.7) in a policy-based (PPO [232]) baselines, referring to the resultant algorithms as PPO_{cost, violation}. Hence, PPO_cost receives a fixed penalty $Z = 1$ upon experiencing an unsafe interaction (i.e., the cost). This penalty value is based on the results obtained in the additional experiment of [168], where we investigate the effects of different penalty regularization terms, Z. In particular, we found a good trade-off between performance and safety with $Z = 1$, while lower values give more importance to the reward, resulting in more unsafe behaviors. We compare with Lagrangian PPO, LPPO [243], as it is a widely adopted constrained DRL baseline and achieves state-of-the-art performance in similar SafetyGym navigation-based tasks[8] In addition, we compare our approach with RCPO, which employs a reward penalty based on Lagrangian multipliers [251] and P3O that relies on a scaled penalty based on the gap between the estimated cost and a

[8] For a fair comparison, we set the cost threshold of LPPO to the average cost obtained by PPO_cost

desired safety threshold. As such, these works are the most closely related methods to ours.

According to relevant literature, the value-based and policy-gradient baselines should achieve the highest rewards and costs, having no penalty information [222]. Conversely, LPPO should show a significant trade-off between average cost and success or present a failure at maintaining the cost threshold when set to low values [152, 222]. In contrast, we expect the penalty-based methods to achieve promising returns while significantly reducing the cost and the number of violations during the training.

Robotic Navigation Tasks

We first introduce our safety navigation tasks that enable rapid testing of policies in realistic contexts. Our environments rely on Unity [126] as it offers rapid prototyping, Gym compatibility, and interface with the Robotic Operating System (ROS), which allows the transfer of the trained policies to actual robots.

In more detail, we employed navigation since it represents a well-known problem in model-free DRL [300, 164], prior safe DRL works [222], and multi-agent DRL [167, 166] literature. While standard navigation planners use a map of the environment and dense sensor information, DRL setups consider significantly more challenging conditions, such as not having a map and relying only on sparse local sensing. We use a similar encoding to prior work [300, 164, 172]. Such encoding in this contest consists of 22 sparse laser scans with a limited range and two values for the target relative position (i.e., distance and heading), resulting in a vector $\mathbf{x} \in \mathbb{R}^{24}$ as input for the agent. Discrete actions encode angular and linear velocities to reduce training times while maintaining good navigation skills [164].[9] The agent thus obtains a dense reward given by the distance (d) difference (Δ) from the goal in two consecutive steps, with a per-step penalty ($-\eta$) that provides an incentive for shorter paths. Formally, at step t, the agent receives a reward:

$$r_t = \begin{cases} 1 & \text{if goal reached} \\ \Delta(d_{t-1}, d_t) - \eta & \text{otherwise} \end{cases} \tag{4.8}$$

Each collision returns a positive cost signal that can be used to compute the desired penalty (e.g., violation), enabling a straightforward application for different penalty-based objectives (as in Equation 4.7) or constraints. We introduce two training and one testing environment with different obstacles, namely *Fixed_obs_NT*, *Dynamic_obs_NT*, and *Evaluation_NT* depicted in Fig. 4.17. Such a variety of conditions provides different settings for properly evaluating safe DRL approaches. The environments inherit several characteristics from known benchmarks such as Safety-Gym [222]. The proposed scenarios share a 4×4 squared meters size (6×6 squared meters for the testing one), randomly generated obstacle-free goals, and a timeout at 500 steps. The main features of the environments are the following:

[9] Our environments also support continuous actions and different domain randomization of the tasks and physical properties through the Unity editor.

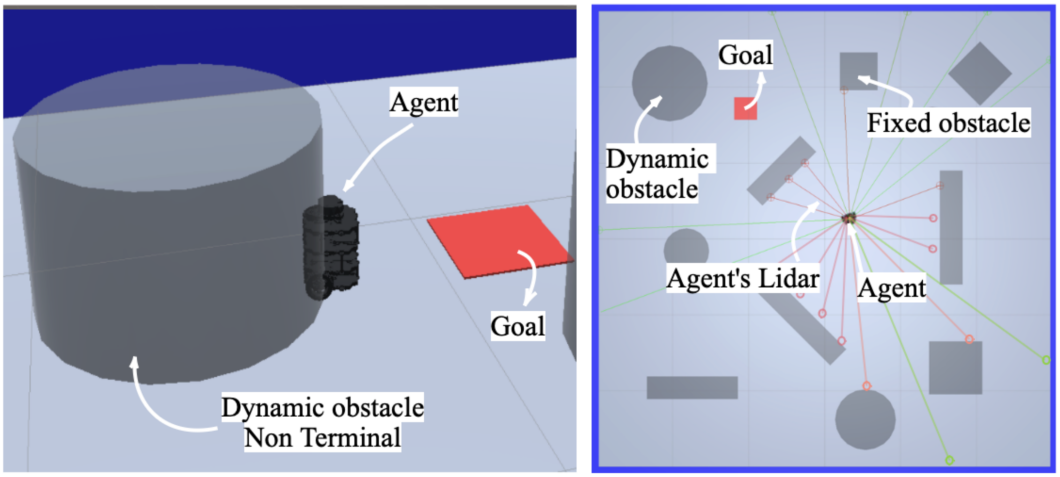


Fig. 4.17: Overview of the Dynamic and Evaluation tasks with different obstacles. Non-terminal obstacles allow the robot to cross them, experiencing more unsafe states. The evaluation environment has both fixed and dynamic non-terminal obstacles.

- *Fixed_obs_NT* has fixed non-terminal (NT) obstacles. The environment returns a signal upon each collision that can be used to model cost functions or other penalties. Non-terminal obstacles are visible to the lidars, but non-tangible, i.e., the Turtlebot3 can pass through them. This class of obstacles represents the main challenge to designing safe DRL solutions, as the robot could obtain positive rewards by crossing an obstacle towards the goal at the expense of unsafe behaviors.
- *Dynamic_obs_NT* has cylindrical-shaped dynamic NT obstacles. Such obstacles move toward random positions at a constant velocity, representing a harder challenge. The obstacles can travel on the robot's goal, so the agent must learn a wider variety of behaviors (e.g., react to an approaching obstacle, stand still to wait for the goal to clear).
- *Evaluation_NT*: we use this evaluation environment to test the generalization abilities of trained policies to new situations. This scenario is wider and contains both fixed and dynamic non-terminal obstacles of different shapes.

Comparison Between hand-designed Properties and CROP

We perform a preliminary experiment showing that creating an exhaustive set of properties encompassing all unsafe behaviors in difficult environments is unfeasible. Hence, we show the benefits of our CROP method to collect and refine safety properties at training time. In our evaluation, we consider the *Fixed_obs_NT* scenario using PPO_violation as a baseline algorithm and hand-design a set of 15 safety properties. The definitions of these specifications follow the rational, safe behaviors for navigation described in Sec. 1.

Fig. 4.18 shows the average success (left) and cost (right) obtained by PPO_violation when using only: (i) properties generated during the training with CROP, (ii) hand-

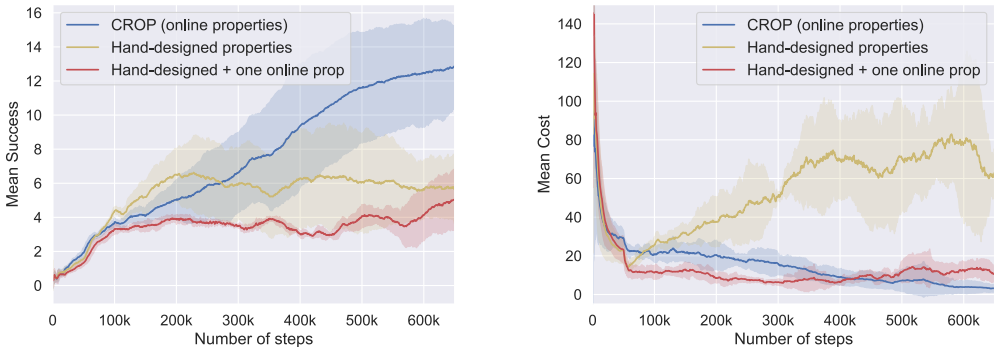


Fig. 4.18: Comparison between a PPO_violation method that uses: CROP (blue), hand-designed (yellow), and hand-designed plus one generated property upon collision (red) properties.

designed properties as described in Sec. 1, (iii) the setup proposed in [168] paper considering hand-designed properties plus one property generated online upon each collision. Note that the latter does not consider the property buffer and the refinement method proposed in CROP [174]. Clearly, computing the violation on CROP's properties provides more information to the learning agent on unsafe behaviors that are challenging to hand-design by a system designer. As such, PPO_violation using CROP achieves the highest performance in terms of average success while also converging to the lowest amount of collisions. For this reason, the following experiments only consider the PPO_violation approach that uses CROP. We will refer to this method as PPO_CROP.

Coverage Online Properties.

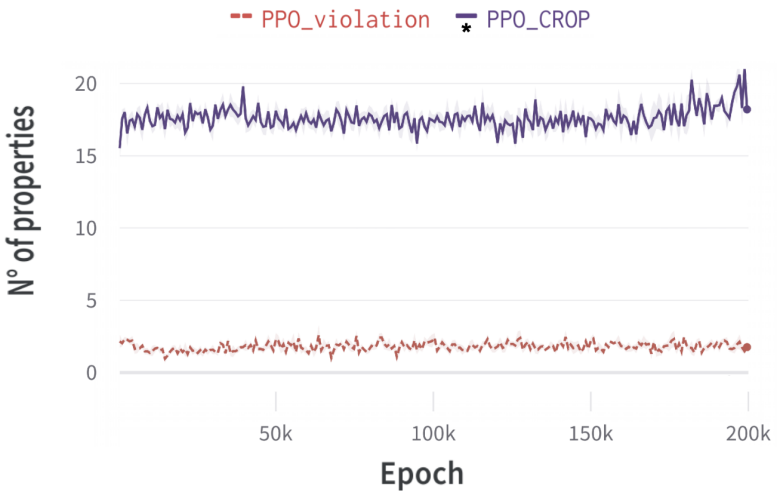


Fig. 4.19: Average number of proprieties' used for the *violation* computation during the training of PPO_violation and our PPO_CROP.

Although is unfeasible to ensure comprehensive coverage of all safety requirements, to investigate the scope of safety requirements collected by CROP, we compare and report in Fig.4.19 the mean number of properties employed to compute the violation value (i.e., properties that contain the unsafe interaction in their domain-codomain) in the case of PPO_CROP (ours) and PPO_violation (which considers the set of hand-designed properties for navigation).

As expected, CROP collects a high number of properties on the task, while the hand-designed approach cannot cope with the complexity of the environment and employs a reduced number of properties for computing the *violation*. Importantly, we note that the number of properties used in CROP tends to slightly increase during the training, while it remains constant for the other methodology. Such a result implies that collecting properties during the training by the agent allows it to find borderline cases of unsafe situations that are difficult to explore using hand-designed properties.

Ensuring the Markov Property

This section aims to tackle one possible drawback of the proposed approach. In particular, by employing the violation value as a penalty, the next state and reward could not be independent of past experienced states and actions, thus potentially violating the Markov property of the problem formulation. Nonetheless, by leveraging the intuitions of [72], we can address such an issue by augmenting the state information with the parameters of the policy at the previous step. While it is unfeasible to include all the policies' weights due to their high dimensionality, we concatenate the output policy parameters, which fully describe the action distribution and the current training step. Previous work [72] refers to such augmentation as *fingerprint* and discusses how this information disambiguates trajectories in more detail. Hence, we adopt the same strategy to solve the issues the violation value could present with respect to the Markov property.

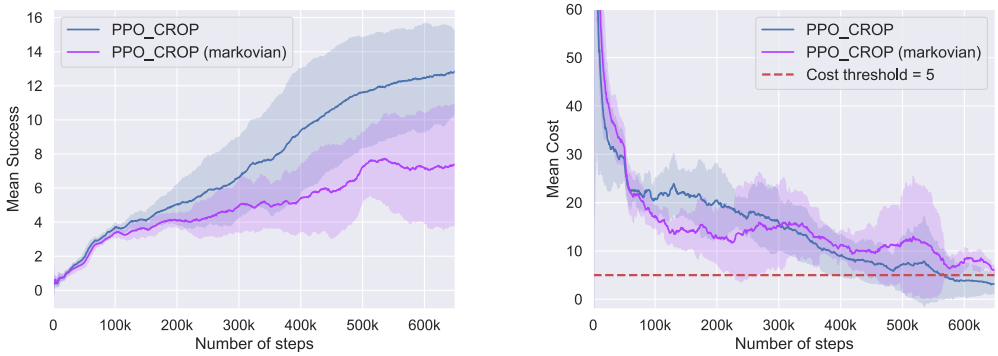


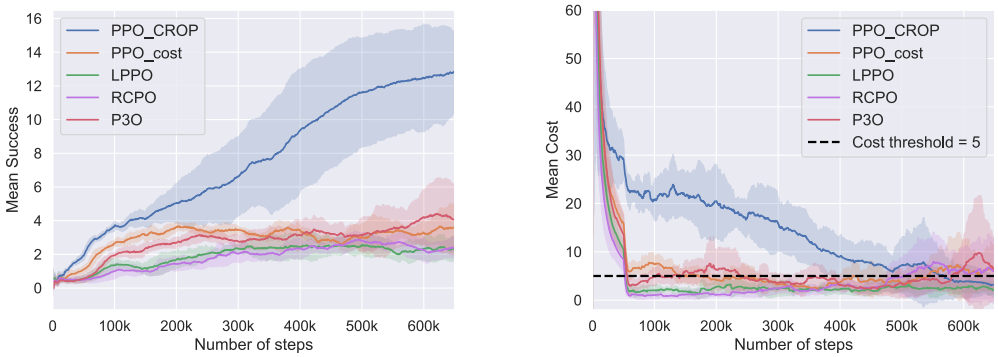
Fig. 4.20: Comparison between PPO_CROP and PPO_CROP (markovian) with fingerprint.

However, while having theoretically grounded drawbacks, practical implementations can often benefit from lower-dimensional state representations. For this reason, we compare the theoretically justified version of PPO_CROP with the fingerprint, called PPO_CROP (markovian) and PPO_CROP that does not use it, in the

Fixed_obs_NT task. Fig. 4.20 confirms the practical advantages of maintaining a lower-dimensional input representation, as PPO_CROP (markovian) leads to lower performance than PPO_CROP in the training regime considered in our experiments. While PPO_CROP (markovian) may eventually converge to the same success and cost values as PPO_CROP, it remains much less sample efficient. Hence, the following sections will consider the PPO_CROP implementation.

Empirical Evaluation on Navigation Tasks

We now compare the performance of our best-performing approach, PPO_CROP (i.e., without employing hand-designed properties and state-augmentation), over the considered safe DRL baselines. Fig.s 4.21, 4.22 show the results of our evaluation with *Fixed* and *Dynamic* obstacles for our PPO_CROP, PPO_cost (that uses the cost as a fixed penalty), and the baselines LPPO, RCPO, and P3O.



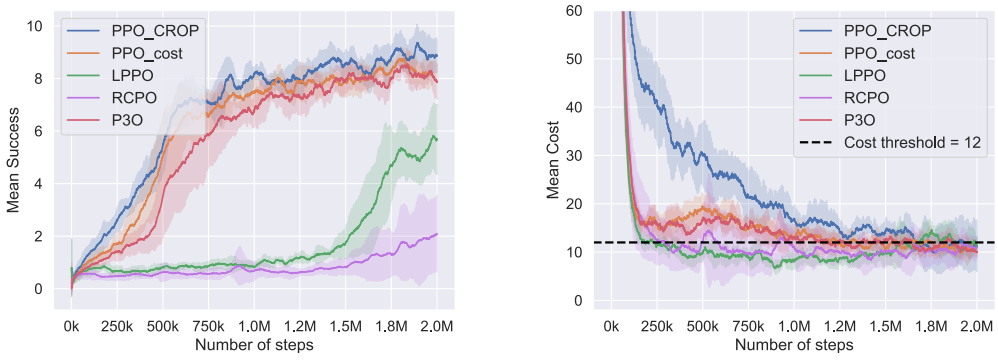
	Mean Violation		
	100k	**300k**	**650k**
*PPO_CROP	6.7 ± 1.0	4.8 ± 1.8	$\mathbf{2.2 \pm 1.2}$
LPPO	1.1 ± 0.5	2.1 ± 1.4	3.5 ± 1.2
RCPO	$\mathbf{0.37 \pm 0.23}$	$\mathbf{1.66 \pm 0.97}$	4.6 ± 5.3
PPO_cost	3.7 ± 1.0	2.5 ± 0.4	6.7 ± 5.8
P3O	2.4 ± 1.1	2.2 ± 0.8	5.8 ± 3.9

Fig. 4.21: Comparison between PPO_CROP, PPO_cost, RCPO, LPPO, and P3O in the *Fixed_obs_NT* environment. The y-axis limit of the mean cost plot has been set to 60 to better visualize the cost value at convergence. In the table, we report the mean violation value at three different global steps for each method tested.

When considering fixed obstacles (Fig. 4.21), our PPO_CROP outperforms other methodologies in terms of average success rate while significantly reducing the cost (i.e., number of collisions) and violation at convergence. Even though in lowering the mean cost and violation, PPO_CROP is slower than the other methodologies, at convergence, it shows a remarkable improvement. The nature of this result is clear: our approach requires experiencing different unsafe behaviors and thus collecting different safety properties before achieving in-depth navigation information. However, as the training steps and knowledge augment, our method outperforms the other

methodologies, augmenting the mean success rate while lowering the mean cost. Moreover, by focusing on the violation metric, which, as stated before, measures the probability of the policy to commit an unsafe action, at convergence, PPO_CROP shows a $\approx 4.5\%$ lower violation over PPO_cost. Considering the number of points used to compute the violation ($m = 10000$) and three common safety properties collected by all the approaches, such improvement maps to 1350 fewer collisions on average. Moreover, LPPO, RCPO, and P3O successfully maintain the cost accumulation under the specified threshold but fail to learn good navigation behaviors. PPO_CROP also achieves a $\approx 2.4\%$, $\approx 1.3\%$, and $\approx 3.6\%$ violation improvement, translating into 720, 390, and 1080 fewer collisions over RCPO, LPPO, and P3O at convergence, respectively. Crucially, the lower number of successes is not motivated by performing longer but safer paths, as PPO_CROP reaches a significantly higher number of successes while maintaining lower violations and cost values.

In general, the poor average success obtained by the Lagrangian baselines confirms the issues of employing indicator cost functions. On top of this, we notice the lack of information carried by the cost since all the methods achieve similar average cost at convergence, but result in significantly different violation values. This further motivates us to introduce the violation metric to foster safety, as the cost could not be sufficient to determine how safely a policy behaves in continuous space environments.



	Mean Violation		
	500k	**1M**	**2M**
*PPO_CROP	16.2 ± 2.8	10.9 ± 1.7	$\mathbf{7.8 \pm 3.7}$
LPPO	$\mathbf{8.9 \pm 1.4}$	$\mathbf{7.1 \pm 1.4}$	9.7 ± 2.2
RCPO	9.0 ± 3.4	9.1 ± 4.3	8.7 ± 1.6
PPO_cost	15.8 ± 2.9	11.5 ± 1.1	9.8 ± 1.2
P3O	12.6 ± 0.4	10.8 ± 2.8	8.1 ± 2.3

Fig. 4.22: Comparison between PPO_CROP, PPO_cost, RCPO, LPPO and P3O on the *Dynamic_obs_NT* environment. The y-axis limit of the mean cost plot has been set to 60 to better visualize the cost value at convergence.

We reach similar conclusions in the task with dynamic obstacles, where our method achieves better or comparable successes and cost values over the counterparts but significantly reduces the violations during the training, showing an $\approx 1.9\%$,

$\approx 0.9\%$, $\approx 2.00\%$, and $\approx 0.3\%$ violation improvement over LPPO, RCPO, PPO_cost, and P3O. Crucially, this difference translates to 570, 270, 600, and 90 fewer collisions on average when employing PPO_CROP over LPPO, RCPO, PPO_cost, and P3O, respectively.

For a fair evaluation, all algorithms have been trained over the same parameters and configurations of dynamic obstacles (i.e., they experience the same random obstacle movements during the training phase, which change over different seeds). Additionally, we note that the additional value functions commonly required by existing safe baselines lead to a comparable wall-clock time overhead (considering the same hardware architecture) over the online safety properties collection and verification procedures of our method, with a maximum deterioration of 30% on the unsafe PPO's total training time.

Formal Violation Value at Convergence.

As discussed in Sec. 4.2.2, the violation value computed using the proposed Monte Carlo sampling-based method approximates the one computed with FV tools. However, considering the gap between the two values, performing an additional analysis using FV before deploying a trained model is crucial. This allows us to provably guarantee whether a model is safe or not with respect to desired safety properties. We also leverage such analysis to confirm our claims on the superior performance of PPO_CROP in terms of violations.

	Method				
Property	**PPO_cost**	***PPO_CROP**	**RCPO**	**LPPO**	**P3O**
$\mathcal{P}_{\uparrow}$	2.1%	**1.12%**	1.7%	1.83%	1.26%
$\mathcal{P}_{\rightarrow}$	6%	1.81%	0.82%	**0.5%**	0.67%
$\mathcal{P}_{\leftarrow}$	0.7%	**0.1%**	0.89%	0.9%	1.4%
Mean	2,.93%	**1.01%**	1.14%	1.08%	1.1%

	Method				
Property	**PPO_cost**	***PPO_CROP**	**RCPO**	**LPPO**	**P3O**
$\mathcal{P}_{\uparrow}$	0.05%	0.04%	0.34%	0.134%	**0.02%**
$\mathcal{P}_{\rightarrow}$	0.04%	0.01%	0.03%	**0.005%**	0.06%
$\mathcal{P}_{\leftarrow}$	0.21%	**0.03%**	0.12%	0.35%	0.17%
Mean	0.1%	**0.03%**	0.16%	0.16%	0.08%

Table 4.4: Mean violation expressed in percentage values computed using formal verification on the best model at convergence. The results of *Fixed obstacles* environment are reported on the left, while those of *Dynamic obstacles* are reported on the right.

Table 4.4 shows the average formal violation value computed on the models at convergence in the *Fixed* and *Dynamic* tasks with respect to the hand-designed properties of Sec. 1. We consider hand-designed properties for a fair comparison since PPO_CROP is trained to minimize violations of the CROP's properties, but not these hand-designed ones. All methodologies achieve a violation value of a few percentage points, confirming the learning of basic safe navigation skills. Crucially, focusing on $\mathcal{P}_{\rightarrow}$, we notice that only for this specific property, LPPO is safer than PPO_CROP. This confirms our hypothesis regarding the significant trade-off between average cost and success that these cost-based methods are subject to. In fact, checking the behavior of LPPO for this property, we noticed that the policy chose a *"stay in place"* action, thus not raising the average cost but sacrificing mean success. In contrast, considering the mean violation on all three safety properties, PPO_CROP turns out to be the safest approach even when employing FV.[10]

Evaluating Trained Policies in an Unseen Scenario.

To test the generalization skills of the trained policies in a previously unseen scenario, we perform an additional experiment on the $Evaluation\_NT$ task depicted on the right of Fig. 4.17.

Table 4.5 reports each model's average success, cost, and (the approximate) violation at convergence. Results confirm the higher performance of PPO_CROP. In particular, our method shows superior navigation skills by achieving a higher number of successes while being safer than the cost counterparts, LPPO, RCPO, and P3O.

Table 4.5: Evaluation results in the *Evaluation_NT* environment

Method	Mean Success	Mean Cost	Mean Violation
*PPO_CROP	**8.8 ± 0.8**	**1.8 ± 0.9**	**1.3 ± 0.6**
LPPO	4.0 ± 0.9	2.1 ± 1.6	1.8 ± 1.3
RCPO	4.9 ± 1.2	2.1 ± 1.9	2.0 ± 1.5
PPO_cost	8.4 ± 1.5	2.4 ± 0.5	2.1 ± 0.5
P3O	6.3 ± 1.5	2.3 ± 0.2	1.9 ± 0.7

Transferring Policies to Real Robots.

Considering the features offered by the Unity game engine and its compatibility with ROS, designing a digital twin copy of a real scenario does not require significant effort. It is thus possible to transfer policies trained in simulation on ROS-enabled

[10] We note results in Table 4.4 differ from the ones published in our previous contributions as we refined the size of the input areas for the three hand-designed properties. In particular, we employed stricter bounds by leveraging the information returned by CROP's online collected properties.

platforms such as our Turtlebot3. Fig. 4.23 shows an example of the same environment in simulation (left) and in the real world (right), on which we tested our trained policies. In particular, our real-world experiments compare our PPO_CROP and LPPO (which resulted in the overall safest approach after ours) in several real corner-case scenarios. The test on the real robot confirms that PPO_CROP shows superior performance with respect to LPPO. In particular, we confirm our hypotheses on the low expressiveness of cost functions. As shown in the video PPO_CROP, by exploiting the collection and refinement of safety properties in combination with the violation value metric leads the agent to acquire safer navigation skills in many corner cases that are difficult to code with cost functions or hand-designed safety specifications. Crucially, we noticed that when the agent trained with LPPO experiences unobstructed forward lidar perception, it tends to favor selecting the shorter path toward the goal, often resulting in collisions with obstacle corners. In contrast, the agent trained using our approach enables the policy to opt for a longer yet safer trajectory toward the goal.

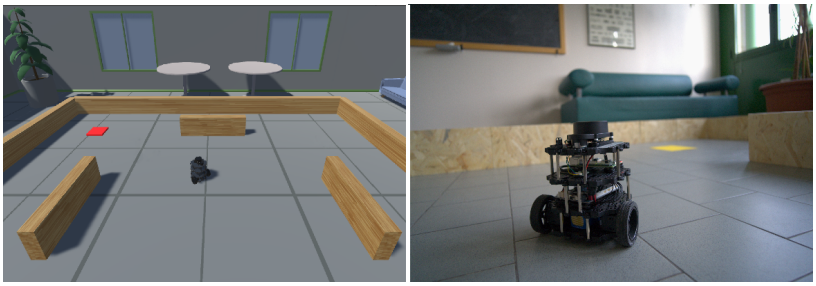
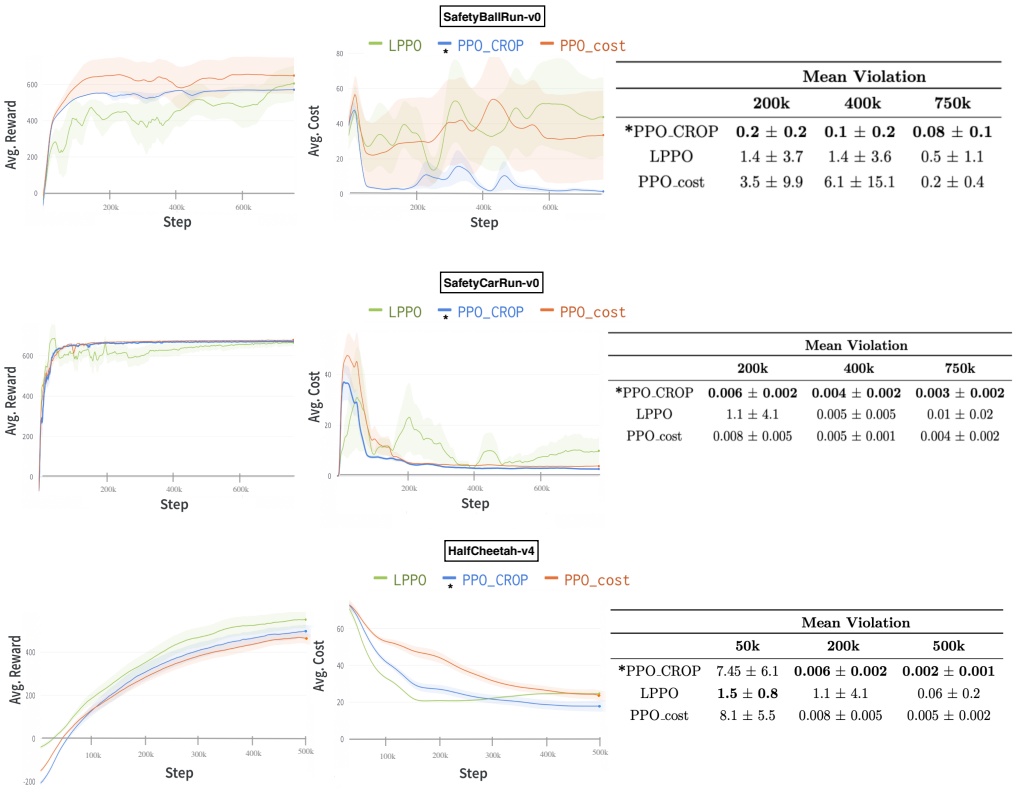


Fig. 4.23: Overview between our Unity simulation environment (left) and the real-world scenario (right).

Empirical Evaluation in Standard Safety Benchmarks

We performed several additional experiments in the SafetyBallRun, SafetyCarRun, and SafetyHalfCheetahVelocity tasks, which are standard benchmarks employed by the safe DRL research community. These tasks consider different simulated robots and safety specifications, which allow showing the general effectiveness of using the violation value as a metric for the safety level. These experiments aim to further confirm the benefit of CROP and the use of the violation value as a penalty in a different domain known in the literature.[11] In more detail, we consider the same hyperparameters of our navigation experiments. However, we only consider LPPO as a Lagrangian baseline since it showed the best trade-off between performance and safety over the other baselines. The following results consider the average reward, cost, and violation collected over ten runs with different random seeds. Fig. 4.24 shows the average performance obtained by our algorithm during the entire training

[11] We refer to the original works for more details about the environments and the cost functions used for the task[2], `github.com/SvenGronauer/Bullet-Safety-Gym`



	Mean Violation		
	200k	400k	750k
*PPO_CROP	**0.2 ± 0.2**	**0.1 ± 0.2**	**0.08 ± 0.1**
LPPO	1.4 ± 3.7	1.4 ± 3.6	0.5 ± 1.1
PPO_cost	3.5 ± 9.9	6.1 ± 15.1	0.2 ± 0.4

	Mean Violation		
	200k	400k	750k
*PPO_CROP	**0.006 ± 0.002**	**0.004 ± 0.002**	**0.003 ± 0.002**
LPPO	1.1 ± 4.1	0.005 ± 0.005	0.01 ± 0.02
PPO_cost	0.008 ± 0.005	0.005 ± 0.001	0.004 ± 0.002

	Mean Violation		
	50k	200k	500k
*PPO_CROP	7.45 ± 6.1	**0.006 ± 0.002**	**0.002 ± 0.001**
LPPO	**1.5 ± 0.8**	1.1 ± 4.1	0.06 ± 0.2
PPO_cost	8.1 ± 5.5	0.008 ± 0.005	0.005 ± 0.002

Fig. 4.24: Plots BulletSafetyGym envs

phase. Crucially, these results confirm the behavior of previous works, where LPPO struggles to keep the cost threshold in some scenarios due to the task complexity. Moreover, LPPO converges to comparable performance with PPO_cost. In contrast, the PPO_CROP maintains the best trade-off between reward, cost, and violation through the training phases.

Fig. 4.25 summarizes these results using the Pareto frontier of the average reward versus cost at convergence, which drastically improves (up and to the left) with PPO_CROP.

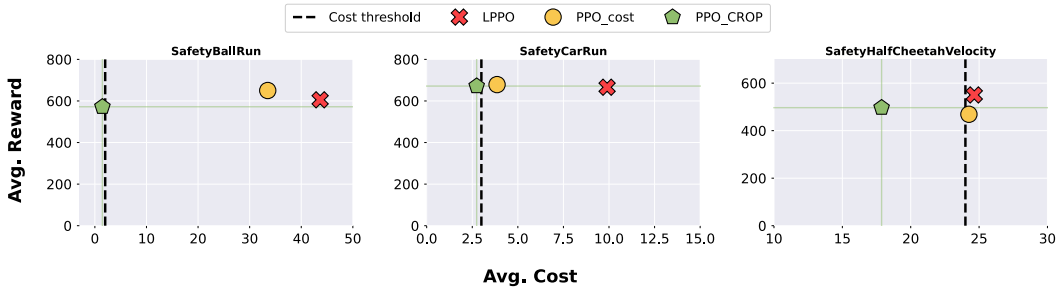


Fig. 4.25: Pareto frontiers of the methodologies tested in several BulletSafetyGym environments. The x-axis represents the average cost, while the y-axis the average reward at convergence. The best possible result is achieved by reaching the top left corner of the plot.

Summary. In this section, we introduced an unconstrained DRL framework that leverages local violations of input-output conditions to foster safety. In detail, we discussed the limitations of using cost functions as in safe DRL [77] presenting: (i) the CROP framework that allows the collection and refinement of safety properties during training, thus overcoming the limitations of hand-designed approaches; (ii) a sample-based approach to approximate a *violation* metric and use it as a penalty in DRL algorithms. We provided provable probabilistic guarantees on the correctness of our approximation, showing that such a violation introduces task-level safety specifications into the optimization, addressing the potential lack of safety information of cost-based approaches. We empirically demonstrated that CROP allows us to obtain a more robust approach with respect to other safe DRL methodologies, promoting safer behaviors while maintaining similar or better returns.

Building on this idea, in the next section, we draw inspiration for adopting a similar mechanism to identify and collect regions of the state space where the agent violates specific behavioral preferences. Instead of solely using these regions to compute penalties, we investigate whether, motivated by the way humans improve through repeated practice, retraining the agent specifically on these challenging regions can enhance both its sample efficiency and its safety.

4.3 ε-Retraining Reinforcement Learning Algorithms

Despite the recent advances in (deep) reinforcement learning (RL), deploying these solutions in safety-critical or preference-sensitive scenarios is challenging: reward-based training alone does not guarantee learning desired behaviors, such as respecting velocity limits or maintaining safe operating conditions in a power network [9]. These behaviors that we want the agent to exhibit are often referred to as *behavioral preferences*.

For example, consider training an RL controller for a power grid. The agent receives a positive reward for meeting electricity demand and a penalty for contingencies, with initial states sampled uniformly from historical operational time series. A natural behavioral preference in this setting is "maintain voltage levels within safe bounds." To satisfy this preference, the agent must learn effective control strategies, specifically in configurations where voltage bound violations are likely. Yet, because uniform initialization samples the state space evenly, the agent rarely re-encounters similar undesirable (unsafe) states, even though these are precisely the regions where preference adherence matters most. This uniform restart strategy also produces high variance in local performance estimates [127, 231], making it more challenging to learn stable and safe policies [68, 123].

To address these challenges, prior policy-based work has explored restarting the environment from carefully chosen states to improve exploration [139, 74, 163]. A prominent example is the *vine* trust region policy optimization (TRPO) algorithm [231], which restarts the agent in states visited by the current policy and generates additional rollouts to reduce gradient-update variance. While theoretically sound, these approaches have important limitations: (i) they are not explicitly designed to improve adherence to specific behavioral preferences; (ii) poorly chosen restart distributions can cause approximate methods to get stuck in local optima [127, 139]; and (iii) they can hinder sample efficiency, requiring many additional rollouts per update. On the value-based side of RL, recent work has also emphasized the benefits of exploiting simulator resets to select starting states [192]. However, these results focus on statistical complexity and low-rank problems, rather than high-dimensional, realistic tasks. Another direction widely studied is constrained optimization [6], where the agent maximizes the reward while keeping the accumulated costs (triggered by a preference violation) below a hard-coded threshold. While theoretically appealing, constrained RL suffers from practical drawbacks: (i) sparse cost signals require many unsafe interactions before learning meaningful value estimates; (ii) threshold choices often induce brittle trade-offs, either crippling task performance or offering little safety benefit; and (iii) constraints do not guarantee avoiding frequent violations during training. These drawbacks motivate the need for novel methods that can directly target the specific regions of the state space where violations occur, without depending on additional rollouts, sparse cost feedback, or finely tuned thresholds.

To overcome these limitations, we introduce ε-`retrain` [180], an exploration strategy for policy-based RL encouraging the learning of behavioral preferences. As illustrated in Fig. 4.39, ε-`retrain` augments the standard uniform restart distribution (blue) with a distribution over retrain areas (green)—regions of the state space where the agent previously violated a preference. At each episode, the restart distribution is chosen with probability ε, which decays during training (purple) to

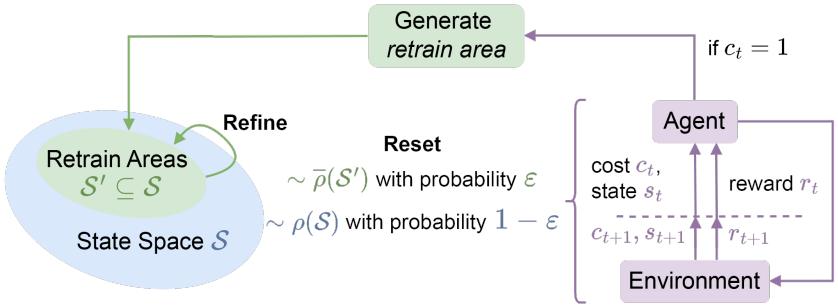


Fig. 4.26: Explanatory overview of ε-`retrain`.

preserve the asymptotic convergence properties of the underlying algorithm. Retrain areas are iteratively collected and refined (combining similar ones) to focus the training phase on learning how to satisfy behavioral preferences where needed. Our work shows that this mixed restart scheme preserves the monotonic improvement guarantees of TRPO [231], and experiments with TRPO, PPO [232] and their constrained variants [243] showed significant improvements in sample efficiency and preference adherence across locomotion, power network, and navigation tasks.

In this section, we explore the benefit of ε-`retrain` from policy-optimization based RL to value-based methods. Our contributions are summarized in the following:

- We explore ε-`retrain` for off-policy, value-based algorithms, replacing on-policy episodic restarts with off-policy sampling from retrain areas.
- We first prove that ε-`retrain` for using mixed uniform restart distributions for policy optimization methods leads, in the worst case, to the same monotonic improvement guarantees as in Schulman et al. [231]. Moreover, we show that an ε-`retrain` version of Q-learning, the underlying algorithm employed by most value-based (deep) RL algorithms, maintains its convergence properties to the optimal action-value function without additional assumptions [277]. On this theoretical side, we also derive a lower bound on the number of state visits required to guarantee a bounded error with respect to the optimal action-value function. We believe this analysis motivates how ε-`retrain` improves practical performance, given that ε-`retrain` increases the visitation counts in the retrain areas.
- We present an in-depth empirical analysis on the performance influence of ε-`retrain` parameters.

Our experiments cover diverse tasks, including simulated locomotion, power grid management, and navigation, demonstrating that ε-`retrain` improves sample efficiency and behavioral adherence across both policy and value-based RL.

Furthermore, we use formal verification of neural networks to provably quantify preference adherence and validate the method in a realistic robotic navigation experiment, confirming its potential in unsafe real-world settings.

4.3.1 *ε-retrain* algorithms

We introduce ε-`retrain` to restart an agent from regions of the state space where it previously violated a behavioral preference. Our goal is to encourage a policy to exhibit behaviors aligned with the preference while improving performance and sample efficiency. To this end, ε-`retrain` collects retrain areas—subsets of the state space $\overline{\mathcal{S}} \subseteq \mathcal{S}$ defined using an iterative procedure that merges parts of $\mathcal{S}$ where the agent violated the preference during training. Training then proceeds under a mixed restart distribution that combines: (i) the standard uniform restart distribution ρ over the full state space $\mathcal{S}$, and (ii) a uniform distribution $\overline{\rho}$ restricted to the retrain areas $\overline{\mathcal{S}}$. At the start of each episode, the environment is initialized from $\overline{\rho}$ with probability ε and from ρ with probability $1 - \varepsilon$, where ε decays over time to preserve the asymptotic properties of the underlying RL algorithm. Crucially, this design requires only the detection of preference violations and a simple merging procedure for retrain areas, making it easy to implement and broadly applicable. We show how ε-`retrain` can be seamlessly integrated into both policy-based and value-based RL algorithms, preserving their theoretical guarantees while enhancing preference adherence. Algorithm 13 presents the general template for ε-`retrain`.

We begin by initializing the memory buffer of retrain areas $\overline{\mathcal{S}}$ as an empty set, and the initial state s_0 is selected from the uniform distribution ρ over the entire state space $\mathcal{S}$. During training, whenever the agent encounters an unsafe interaction, signaled by a positive cost, we trigger the iterative procedure for collecting and merging retrain areas. As in the safe RL literature, this indicator cost acts as a reliable signal for detecting violations of the desiderata [77]. Specifically, upon a violation, the `generate_retrain_area` method is called, taking the previous state s_{t-1} and producing an ω-bubble. Intuitively, this ω-bubble defines a neighborhood around s_{t-1} by encoding each state feature as an interval of fixed size ω, thereby capturing a local region of the state space likely to exhibit similar unsafe behavior.

Algorithm 13: Template for *ε-retrain* methods

Input: *bubble* size ω for initial retrain area, *similarity* value β to merge similar areas, *decay*, *initial*, and *minimum* values for the ε scaling.
$\varepsilon\_decay \leftarrow (min\_\varepsilon - 1.0)/(decay \cdot (epochs \cdot steps\_per\_epoch))$
$\overline{\mathcal{S}} \leftarrow \emptyset$; $s_0 \leftarrow \rho(\mathcal{S})$ Initialize areas buffer and environment
for each episode **do**
  while episode is not done **do**
    Execute the training loop of the RL algorithm for each step t.
    if s_t is unsafe (i.e., cost > 0) **then**
      $r \leftarrow$ `generate_retrain_area`(s_{t-1}, ω)
      if $\exists\, r' \in \overline{\mathcal{S}}$ that is β-similar to r **then**
        $r \leftarrow$ `area_refinement`(r, r')
      $\overline{\mathcal{S}} \leftarrow \overline{\mathcal{S}} \cup r$
  if $random(0,1) < \varepsilon \wedge \overline{\mathcal{S}} \neq \emptyset$ **then**
    $s_0 \leftarrow \overline{\rho}($`sample_retrain_area`$(\overline{\mathcal{S}}))$
    $\varepsilon \leftarrow \max(\varepsilon\_decay \cdot (epoch) * steps\_per\_epoch + 1.0,\ min\_\varepsilon)$
  else
    $s_0 \leftarrow \rho(\mathcal{S})$

Fig. 4.27 illustrates this idea in a navigation task: when the agent collides with an obstacle, the resulting retrain area represents the range of states around the collision that are also at risk. Retraining the agent from such regions increases the chances of correcting unsafe actions more efficiently.

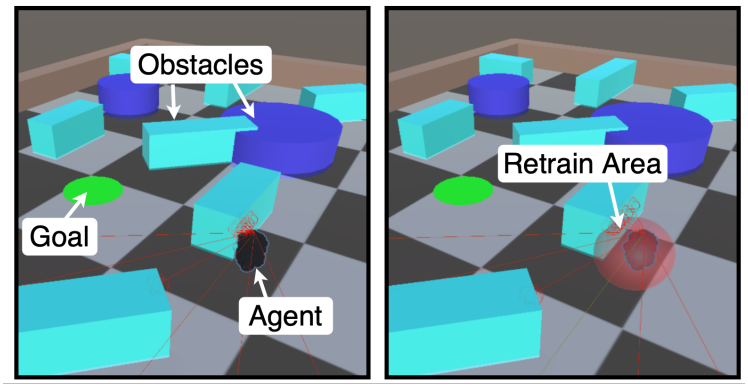


Fig. 4.27: (left) The agent collides with an obstacle, receiving a positive cost. (right) A retrain area is created from that state.

Once generated, this bubble becomes a retrain area r. The algorithm then automatically checks for an existing similar retrain area r' within $\overline{\mathcal{S}}$, using the similarity threshold β. If such an r' is found, the `area_refinement` method merges the two into a single, more compact retrain area. If no match is found, r is stored in the buffer $\overline{\mathcal{S}}$, which was initialized as empty. This refinement procedure provides two benefits: (i) it keeps the buffer size manageable by avoiding redundant retrain areas, and (ii) it clusters similar violations together, ensuring balanced sampling across distinct types of unsafe states. The updated buffer $\overline{\mathcal{S}}$ then becomes available for future retraining episodes. At the start of each new episode, if $\overline{\mathcal{S}}$ is non-empty, the environment is reset either from the full state space $\mathcal{S}$ with probability $1-\varepsilon$ or from one of the retrain areas with probability ε. To avoid biasing training toward suboptimal restart distributions, ε-`retrain` employs a linear decay of ε, gradually shifting the restart distribution back toward ρ as training progresses. When the initial state is drawn from a retrain area, the agent is effectively re-exposed to situations where it previously violated the behavioral preference, allowing it to improve its policy (or value estimates) in those critical regions. While the general structure of ε-`retrain` remains unchanged, its implementation differs depending on whether the underlying algorithm is on-policy or off-policy. These differences are particularly evident in how retrain trajectories are stored, reused, and incorporated into updates.

On-policy vs. Off-policy-based Implementations of ε-`retrain`. Minor differences arise when adapting ε-`retrain` from on-policy to off-policy algorithms. In the on-policy case, trajectories restarted from retrain areas are always used in the policy update but are then discarded, consistent with the standard trajectory-based data flow of on-policy policy-gradient methods. By contrast, off-policy methods that are more popular in value-based setups, typically rely on sampling mini-batches of transitions from a replay buffer, which means that restarted trajectories are not guaranteed to appear in the sampled batch, or may only appear partially. Since the

effectiveness of ε-`retrain` hinges on revisiting the initial steps of retrain trajectories (where the agent re-experiences or corrects actions that previously violated the preference), we must ensure that these samples are not lost during replay. To this end, we decouple the memory into two buffers: a standard buffer for all interactions and a retrain buffer dedicated to storing the first steps of retrain trajectories. At each update, we form a mini-batch by sampling from both buffers, ensuring that preference-violating regions are consistently revisited when updating the value functions. Finally, unlike the on-policy case where retrain trajectories are removed after each update, in the off-policy version these are retained. Hence, to prevent filling the retrain buffer with redundant copies of the same unsafe episodes, during retrained episodes we move trajectories in the replay buffer.

Generation and Refinement Processes

To better illustrate how ε-`retrain` works in practice, this section presents through two visual examples the key components of the retrain-area mechanism: (i) the generation of new retrain areas when a violation occurs, and (ii) the refinement process used to merge similar areas. We first present the generation step in the context of robotic navigation and then describe refinement in the *HalfCheetah* locomotion task.

Retrain Area Generation. Consider an agent navigating an environment that receives a positive cost signal after colliding with an obstacle. This unsafe interaction, illustrated in Fig. 4.28(a), triggers the generation procedure. The state s_{t-1} that immediately preceded the collision (Fig. 4.28(b)) is selected, and an ω-bubble is placed around it to create a retrain area (Fig. 4.28(c)).

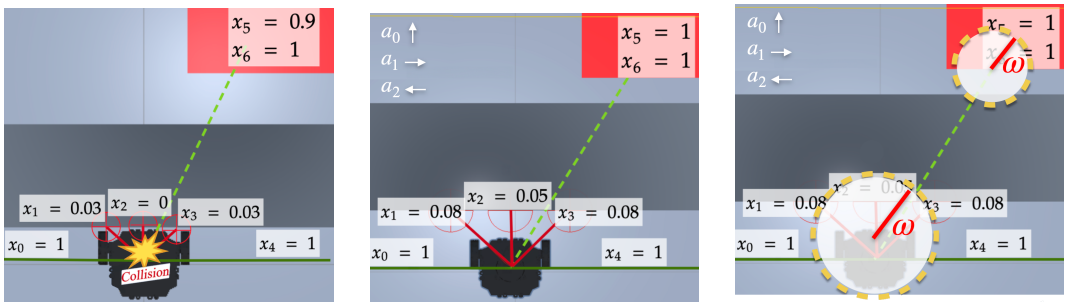
The ω-bubble is defined by expanding each feature of s_{t-1} into an interval of fixed size ω. For example, with $\omega = 0.05$, the resulting retrain area X can be written as:

$$X : \{x_0 = [0.95, 1],\ x_1 = [0.03, 0.08],\ x_2 = (0, 0.05],\\ x_3 = [0.03, 0.08],\ x_4, x_5, x_6 = [0.95, 1]\}.$$

States sampled from X are potentially risky, as they correspond to situations in which a violation is likely. By restarting training episodes from this region using ε-`retrain`, the agent is repeatedly exposed to unsafe conditions, giving it the opportunity to learn behaviors that better satisfy the desired safety preference.

Refinement Procedure. After a retrain area is created, ε-`retrain` checks whether it should be merged with an existing one. This is achieved by comparing the similarity of intervals that define two candidate retrain areas, X and X'. Each area is represented as a set of feature intervals, $X = \{x_0, x_1, \ldots, x_n\}$ and $X' = \{x'_0, x'_1, \ldots, x'_n\}$, and similarity is assessed using Moore's interval algebra [196]. The distance between intervals $[\underline{x_i}, \overline{x_i}]$ and $[\underline{x'_i}, \overline{x'_i}]$ is:

$$d([\underline{x_i}, \overline{x_i}], [\underline{x'_i}, \overline{x'_i}]) = \max(|\underline{x_i} - \underline{x'_i}|,\ |\overline{x_i} - \overline{x'_i}|).$$



(a) (b) (c)

Fig. 4.28: Retrain area generation. (a) Collision with an obstacle. (b) State that led to the collision. (c) ω-bubble used to initialize the retrain area. In this example, the ω-bubble size is the same for all features but is shown at different scales here for clarity.

Two retrain areas are considered similar if and only if:[12]

$$\forall i \in \{0, \ldots, n\}, \quad d(x_i, x_i') \leq \beta,$$

where β is a user-defined similarity threshold. If X and X' are deemed similar, ε-`retrain` merges them into a new area X'', with intervals defined as:

$$x_i'' = [\min(\underline{x_i}, \underline{x_i'}),\ \max(\overline{x_i}, \overline{x_i'})].$$

This union expands the coverage of the retrain area while avoiding redundancy. Otherwise, the two areas remain separate. Fig. 4.29 provides an example in the *HalfCheetah* locomotion task. The desired preference is a velocity bound along the x-axis. The red marker indicates a violation of this threshold. In the left panel, two unsafe states fall within distance β, leading to refinement into a larger retrain area. In the right panel, the states are too dissimilar, so the areas remain distinct.

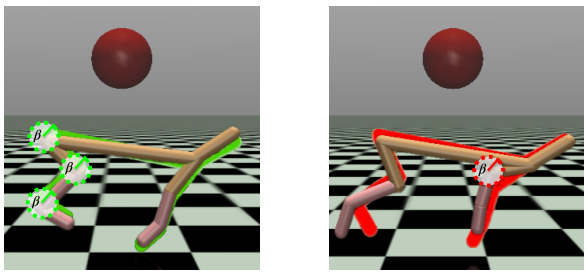


Fig. 4.29: Left: Explanatory similar unsafe states for a subset of the input features—the two states are within distance β. Right: Explanatory different unsafe situations—at least a couple of input features have a distance greater than β.

[12] Without loss of generality, we assume the ℓ_2-norm in the feature space is a meaningful distance metric. Other choices (e.g., task-specific metrics in robotic manipulation) do not affect the refinement procedure.

Improving Policy Optimization via *ε-retrain*

In this part, we derive a bound on the monotonic policy improvement for *ε-retrain*'s mixture of uniform restart distributions. We show that the original bound on monotonic policy improvement presented by [231] still holds and can be tighter. Notably, *this result motivates both the design of our method as well as its superior performance* (Sec. 5.1.4). For the sake of clarity, we first recall the definition of *α-coupled policies* and the related lemma introduced to derive the bound in case of a single uniform restart distribution over $\mathcal{S}$. First, Definition 4.3 couples two policies that behave in the same way (i.e., given a state, they pick the same action) with probability $\geq 1-\alpha$, and Lemma 4.1 bounds the gap between policy advantages satisfying such policies.

Definition 4.3 (*α*-coupled policies Schulman et al. [231]): *We say that π and π' are two α-coupled policies if $\forall s \in \mathcal{S}$, we can define a joint distribution $(a, a')|s$ such that $P(a \neq a'|s) \leq \alpha$.*

Lemma 4.1 (Schulman et al. [231]): *Given two α-coupled policies, π and π', we have that: $\left|\mathbb{E}_{s_t\sim\pi'}[\tilde{A}(s_t)] - \mathbb{E}_{s_t\sim\pi}[\tilde{A}(s_t)]\right| \leq 4\alpha(1-(1-\alpha)^t)\max_{s,a}|A_\pi(s,a)|$. It follows that: $|\psi(\pi') - L_\pi(\pi')| \leq \frac{4\alpha^2\gamma k}{(1-\gamma)^2}$ with $k = \max_{s,a}|A_\pi(s,a)|$.*

We extend Lemma 4.1 to the mixture of restarting distributions used by *ε-retrain* (right side of Equation 4.9, where $s_0 \sim \overline{\rho}$). To this end, we define the expected discounted return of a new policy π' over the current π under the ε mixture of restart policies as:

$$\begin{aligned}\overline{\zeta}(\pi') = (1-\varepsilon)\left[\zeta(\pi) + \mathbb{E}_{\substack{\tau\sim\pi'\\ s_0\sim\rho}}\left[\sum_{t=0}^{\infty}\gamma^t\tilde{A}(s_t)\right]\right] \\ + (\varepsilon)\left[\zeta(\pi) + \mathbb{E}_{\substack{\tau\sim\pi'\\ s_0\sim\overline{\rho}}}\left[\sum_{t=0}^{\infty}\gamma^t\tilde{A}(s_t)\right]\right].\end{aligned} \tag{4.9}$$

Hence, for the surrogate loss (see Equation 2.1), we compute the following local approximation:

$$\begin{aligned}\overline{L}_\pi(\pi') = (1-\varepsilon)\left[\zeta(\pi) + \mathbb{E}_{\substack{\tau\sim\pi\\ s_0\sim\rho}}\left[\sum_{t=0}^{\infty}\gamma^t\tilde{A}(s_t)\right]\right] \\ + (\varepsilon)\left[\zeta(\pi) + \mathbb{E}_{\substack{\tau\sim\pi\\ s_0\sim\overline{\rho}}}\left[\sum_{t=0}^{\infty}\gamma^t\tilde{A}(s_t)\right]\right].\end{aligned} \tag{4.10}$$

We then extend Lemma 4.1 to provide an upper bound on the distance between $\overline{\zeta}(\pi')$ and $\overline{L}_\pi(\pi')$ under *ε-retrain*.

Lemma 4.2: *Considering a mixed restart distribution over ρ and $\overline{\rho}$ using ε-retrain, it holds that*

$$|\overline{\zeta}(\pi') - \overline{L}_\pi(\pi')| \leq \frac{4\alpha^2\gamma}{(1-\gamma)^2}\Big[k(1-\varepsilon) + k'\varepsilon\Big] \leq \frac{4\alpha^2\gamma k}{(1-\gamma)^2},$$

with $k' = \max_{\substack{s_0 \sim \overline{\rho} \\ \tau \sim \pi'}} |A_{\pi'}(s,a)| \leq k = \max_{\substack{s_0 \sim \rho \\ \tau \sim \pi'}} |A_{\pi'}(s,a)|$, $\varepsilon \in [0,1]$.

Proof. In order to show the monotonic improvement guarantees of ε-*retrain*, we want to bound the difference $|\overline{\zeta}(\pi') - \overline{L}_{\pi}(\pi')|$ where

$$\overline{\zeta}(\pi') = (1-\varepsilon)\left[\zeta(\pi) + \mathbb{E}_{\substack{\tau \sim \pi' \\ s_0 \sim \rho}}\Big[\sum_{t=0}^{\infty} \gamma^t \tilde{A}(s_t)\Big]\right] + (\varepsilon)\left[\zeta(\pi) + \mathbb{E}_{\substack{\tau \sim \pi' \\ s_0 \sim \overline{\rho}}}\Big[\sum_{t=0}^{\infty} \gamma^t \tilde{A}(s_t)\Big]\right]$$

and

$$\overline{L}_{\pi}(\pi') = (1-\varepsilon)\left[\zeta(\pi) + \mathbb{E}_{\substack{\tau \sim \pi \\ s_0 \sim \rho}}\Big[\sum_{t=0}^{\infty} \gamma^t \tilde{A}(s_t)\Big]\right] + (\varepsilon)\left[\zeta(\pi) + \mathbb{E}_{\substack{\tau \sim \pi \\ s_0 \sim \overline{\rho}}}\Big[\sum_{t=0}^{\infty} \gamma^t \tilde{A}(s_t)\Big]\right].$$

Hence, following the proof of [231] we can derive a similar bound with simple algebra as:

$$\begin{aligned}
&|\overline{\zeta}(\pi') - \overline{L}_{\pi}(\pi')| = \\
&(1-\varepsilon)\left[\zeta(\pi) + \mathbb{E}_{\substack{\tau \sim \pi' \\ s_0 \sim \rho}}\Big[\sum_{t=0}^{\infty} \gamma^t \tilde{A}(s_t)\Big]\right] + (\varepsilon)\left[\zeta(\pi) + \mathbb{E}_{\substack{\tau \sim \pi' \\ s_0 \sim \overline{\rho}}}\Big[\sum_{t=0}^{\infty} \gamma^t \tilde{A}(s_t)\Big]\right] - \\
&(1-\varepsilon)\left[\zeta(\pi) + \mathbb{E}_{\substack{\tau \sim \pi \\ s_0 \sim \rho}}\Big[\sum_{t=0}^{\infty} \gamma^t \tilde{A}(s_t)\Big]\right] + (\varepsilon)\left[\zeta(\pi) + \mathbb{E}_{\substack{\tau \sim \pi \\ s_0 \sim \overline{\rho}}}\Big[\sum_{t=0}^{\infty} \gamma^t \tilde{A}(s_t)\Big]\right] \\
=&(1-\varepsilon)\underbrace{\left[\sum_{t=0}^{\infty} \gamma^t \Big|\mathbb{E}_{\substack{\tau \sim \pi' \\ s_0 \sim \rho}} \tilde{A}(s_t) - \mathbb{E}_{\substack{\tau \sim \pi \\ s_0 \sim \rho}} \tilde{A}(s_t)\Big|\right]}_{\leq \frac{4k\alpha^2\gamma}{(1-\gamma)^2} \text{ by Schulman et al. [231]}} + (\varepsilon)\left[\sum_{t=0}^{\infty} \gamma^t \Big|\mathbb{E}_{\substack{\tau \sim \pi' \\ s_0 \sim \overline{\rho}}} \tilde{A}(s_t) - \mathbb{E}_{\substack{\tau \sim \pi \\ s_0 \sim \overline{\rho}}} \tilde{A}(s_t)\Big|\right] \\
\leq&(1-\varepsilon)\Big[\frac{4k\alpha^2\gamma}{(1-\gamma)^2}\Big] + (\varepsilon)\left[\sum_{t=0}^{\infty} \gamma^t \Big|\mathbb{E}_{\substack{\tau \sim \pi' \\ s_0 \sim \overline{\rho}}} \tilde{A}(s_t) - \mathbb{E}_{\substack{\tau \sim \pi \\ s_0 \sim \overline{\rho}}} \tilde{A}(s_t)\Big|\right].
\end{aligned}$$

From the fact that the α-coupling definition (reported in Def. 4.3) is expressed over all possible states s, without any assumption or restriction on the initial state distribution, we have that π and π' are still α-coupled even when we chose s_0 using a $\overline{\rho}$ as initial state distribution. Hence,

$$\begin{aligned}
|\overline{\zeta}(\pi') - \overline{L}_{\pi}(\pi')| &\leq (1-\varepsilon)\Big[\frac{4k\alpha^2\gamma}{(1-\gamma)^2}\Big] + (\varepsilon)\Big[\frac{4k'\alpha^2\gamma}{(1-\gamma)^2}\Big] \\
&= \frac{4\alpha^2\gamma}{(1-\gamma)^2}\Big[k + \varepsilon(k'-k)\Big] = \frac{4\alpha^2\gamma}{(1-\gamma)^2}\Big[k(1-\varepsilon) + k'\varepsilon\Big].
\end{aligned}$$

Next, note that since $0 \leq \varepsilon \leq 1$, we can derive that $k' \leq k$. In fact, we have:

$$\begin{aligned} k(1-\varepsilon)+k'\varepsilon &\leq k \\ -\varepsilon k+\epsilon k' &\leq 0 \\ k &\geq k'. \end{aligned}$$

Thus we conclude that

$$\frac{4\alpha^2\gamma}{(1-\gamma)^2}\Big[k(1-\varepsilon)+k'\varepsilon\Big] \leq \frac{4\alpha^2\gamma k}{(1-\gamma)^2}.$$

The correctness of our result stems from the following facts: (i) α-coupled policies are coupled over the entire $\mathcal{S}$, ensuring that policies are coupled even when combining ρ and $\overline{\rho}$, and (ii) k is defined as the maximum over all $(s,a) \in (\mathcal{S} \supseteq \overline{\mathcal{S}}, \mathcal{A})$ and, as such, $k \geq k'$. In more detail, Lemma 4.2 shows that enforcing the agent to start from retraining areas can help in narrowing the gap between the $\zeta(\pi')$ and the local approximation $L_\pi(\pi')$. In fact, when $k = k'$, the difference reduces to the original case of Lemma 4.1. In contrast, when $k' < k$, the trajectories induced by $\overline{\rho}$ are not optimal since agents are penalized for violating the preference. Thus, the agent gains a more precise understanding of the portion of the state space where the policy does not yield the maximum possible advantage. This motivates the design of *ε-retrain* that linearly scales down $\varepsilon \to 0$, avoiding getting stuck in suboptimal restart distributions. □

Finally, by exploiting the relationship between the total variation divergence and the KL divergence [213], we derive the following corollary on the monotonic improvement guarantee under *ε-retrain*-based methods.

Corollary 2: *Let ρ and $\overline{\rho}$ be two different restart distributions. Combining ρ and $\overline{\rho}$ during the training as in ε-retrain, and by setting $\alpha = D_{KL}^{max}(\pi,\pi')$, the bound on the monotonic improvement for the policy update $\overline{\zeta}(\pi') \geq \overline{L}_\pi(\pi') - CD_{KL}^{max}(\pi,\pi')$ with $C = \frac{4k\gamma}{1-\gamma^2}$ is still guaranteed.*

The result naturally follows from Lemma 4.2 and the original derivation of Schulman et al. [231].[13]

Improving Value Function Estimation via ε-retrain

In this section, we show how ε-`retrain`'s mixture of uniform restart distributions $(1-\varepsilon)\rho+\varepsilon\overline{\rho}$ maintains the same convergence properties as Q-learning [277], since it is the underlying algorithm over which modern deep RL value-based approaches are built upon. In particular, Q-learning is known to converge to the optimal action-value function Q^* under standard stochastic approximation conditions, provided that every state–action pair is visited infinitely often Then, we derive the lower bound on the number of visits required to guarantee a bounded error on an estimated Q-function with respect to the optimal action-value function Q^*. We believe this analysis further clarifies how ε-`retrain` improve practical performance since ε-`retrain` increases the visitation counts in the retrain areas.

[13] We note the theoretically justified procedure motivating the policy improvement bound [231] does not hold for most practical deep RL policy optimization algorithms.

Following standard assumptions that (i) the MDP is finite (as stated in Chapter 2); (ii) the reward function $r(s,a)$ is bounded, for instance $r(s,a) \in [0, R_{\max}]$;[14] and (iii) the learning rate schedule satisfies the Robbins-Monro conditions [225]. The latter condition requires to specify how the sequence of learning rates $\alpha_t(s,a)$ decays for convergence to the true value function (i.e., by (i) diminishing but not too fast $\sum_{t=1}^{\infty} \alpha_t(s,a) = \infty$ and (ii) having squared summability $\sum_{t=1}^{\infty} \alpha_t^2(s,a) < \infty$, ensuring that the updates eventually stabilize and do not oscillate indefinitely).[15] These assumptions imply that Q-values are bounded by $V_{\max} = \frac{R_{\max}}{1-\gamma}$.

Corollary 3 (Mixed restart distribution does not affect asymptotic convergence): *Let $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \rho, R, \gamma)$ be a finite MDP with discount factor $\gamma \in (0,1)$ and bounded rewards $r(s,a) \in [0, R_{\max}]$. Consider a Q-learning algorithm with updates*

$$Q_{t+1}(s_t, a_t) \;=\; Q_t(s_t, a_t) \;+\; \alpha_t(s_t, a_t)\Big[r_t + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)\Big],$$

Assume the agent's episode initial states are drawn i.i.d. from the mixed restart distribution

$$d_\varepsilon \;=\; (1-\varepsilon)\rho \;+\; \varepsilon\,\overline{\rho}, \qquad \varepsilon \in [0,1],$$

and that under the resulting behavior policy, each state–action pair (s,a) is visited infinitely often almost surely.

Then the asymptotic limit of the Q-learning iterates is the unique optimal action-value function Q^* with probability 1; in particular, the choice of $\varepsilon \in [0,1]$ (and of $\overline{\rho}$) does not affect convergence to Q^*.

Proof. Under the stated assumptions, the Q-learning update may be viewed as a stochastic approximation of the fixed point equation $Q = TQ$, where T is the Bellman optimality operator. The operator T is a γ-contraction in the $\|\cdot\|_\infty$ norm and thus has a unique fixed point Q^*. Standard convergence results for Q-learning [277] show that the stochastic iterates Q_t converge almost surely to Q^* provided that: (i) the state–action space is finite; (ii) rewards are uniformly bounded; (iii) the step sizes $\alpha_t(s,a)$ satisfy the Robbins–Monro conditions; and (iv) every state–action pair is visited infinitely often (so that the noise in the stochastic approximation is adequately averaged out). The restart distribution d_ε affects only the sampling/visit frequencies of states (and thus which transitions are observed more or less often), but it does not change (a) the form of the Bellman operator T, (b) its fixed point Q^*, nor (c) the Q-learning update rule. Therefore, as long as the chosen restart distribution (and the induced behavior policy) ensures condition (iv) i.e., each (s,a) is visited infinitely often almost surely, all other assumptions required by the convergence theorem remain satisfied, and the iterates converge to the same unique Q^* regardless of ε or the particular $\overline{\rho}$ used. Hence, the mixed restart distribution $(1-\varepsilon)\rho + \varepsilon\overline{\rho}$ does not influence the global convergence of Q-learning under the stated conditions. □

[14] This assumption can be relax assuming $r \in [R_{\min}, R_{\max}]$, but we consider a simpler bound for our derivation.

[15] For notation consistency with the literature, in the following α is referred to as the learning rate and not to "coupled" policies as in the previous section.

Nonetheless, to assess the efficiency of Q-learning-based algorithms, it is useful to quantify how the accuracy of the learned action-value function depends on the number of samples collected for each state-action pair. In particular, finite-sample analyses show that achieving a bounded error with respect to the optimal value requires a sufficient number of visits to every (s, a). The following theoretical analysis formalizes this by providing a lower bound on the sample complexity needed to ensure that the estimation error is within a desired tolerance. Additionally, this result directly allows us to discuss the role of our ε-`retrain` strategy with respect to the improved sample efficiency that ε-`retrain` shows in the empirical evaluation. Notably, by biasing exploration toward states where the agent shows suboptimal behaviors, ε-`retrain` effectively accelerates the growth of the visit counts $N(s, a)$ in the most critical regions of the state space, thereby tightening the error bound in those regions.

Lemma 4.3 (Finite-sample bound for Q-values): *Let $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \rho, R, \gamma)$ be a finite MDP with discount factor $\gamma \in (0, 1)$ and bounded rewards $r(s, a) \in [0, R_{\max}]$. Define $V_{\max} := \frac{R_{\max}}{1-\gamma}$. For any accuracy parameter $\xi > 0$ and confidence parameter $\delta \in (0, 1)$, if each state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ is sampled at least*

$$N(s, a) \;\geq\; \frac{2V_{\max}^2}{\xi^2(1-\gamma)^2}\,\ln\!\left(\frac{2|\mathcal{S}||\mathcal{A}|}{\delta}\right) \;=\; \frac{2R_{\max}^2}{\xi^2(1-\gamma)^4}\,\ln\!\left(\frac{2|\mathcal{S}||\mathcal{A}|}{\delta}\right), \tag{4.11}$$

then with probability at least $1 - \delta$ we have

$$\left|Q'(s, a) - Q^*(s, a)\right| \;\leq\; \xi \qquad \text{for all } (s, a),$$

where $Q' := \hat{T}Q$ is the one-step empirical Bellman update of some Q, and Q^* is the optimal action-value function.

Proof. We prove this lemma using two well-known results, namely the contraction property of the Bellman operator and Hoeffding's concentration inequality [103].

Specifically, fix a state-action pair (s, a) and a reference value function Q. If we draw $N(s, a)$ independent samples $\{(r_i, s_i')\}_{i=1}^{N}$ with $s_i' \sim \mathcal{P}(\cdot \mid s, a)$ we can define the empirical Bellman backup

$$\widehat{TQ}(s, a) \;=\; \frac{1}{N(s, a)} \sum_{i=1}^{N(s,a)} \Big[r_i + \gamma \max_{a'} Q(s_i', a')\Big],$$

and its expectation

$$(TQ)(s, a) \;=\; \mathbb{E}_{s' \sim P(\cdot|s,a)}\big[r(s, a) + \gamma \max_{a'} Q(s', a')\big].$$

We note that each summand $X_i := r_i + \gamma \max_{a'} Q(s_i', a')$, which represents a stochastic sample of the target value, is bounded in $[0, V_{\max}]$, since $r \in [0, R_{\max}]$ and $\gamma \max_{a'} Q(s', a') \in [0, \gamma V_{\max}]$ and thus

$$\begin{aligned} R_{\max} + \gamma V_{\max} &= R_{max} + \gamma \frac{R_{max}}{1-\gamma} \\ &= \frac{R_{max}(1-\gamma) + \gamma R_{max}}{1-\gamma} \\ &= V_{max}. \end{aligned}$$

We can now bound the difference error between $\widehat{TQ}(s,a)$, the estimate of $Q(s,a)$ over N samples, and the true expectation $(TQ)(s,a)$ using the following.

Lemma 4.4 (Hoeffding [103]'s inequality): *Let $X_1, \ldots, X_n$ be N i.i.d. random variables with range $[0, C]$. Let $\kappa \in (0,1)$ be the error with the aim of bounding. Then, the probability that the sample mean deviates from the true mean is bounded by*

$$\Pr\left(\left|\frac{1}{N}\sum_{i=1}^{N} X_i - \mathbb{E}[X]\right| \geq \kappa\right) \leq 2e^{-2N\kappa^2/C^2}. \tag{4.12}$$

Specifically, by lemma 4.4, for any $\kappa > 0$ and $C \in [0, V_{\max}]$, i.e., the range of each X_i, we have

$$\Pr\Big(\big|\widehat{TQ}(s,a) - (TQ)(s,a)\big| \geq \kappa\Big) \;\leq\; 2\exp\left(-\frac{2N(s,a)\kappa^2}{V_{\max}^2}\right). \tag{4.13}$$

From known results in the Q-learning convergence analysis [244], if the empirical operator is κ-accurate uniformly, i.e., $\sup_{s,a} |\widehat{TQ}(s,a) - (TQ)(s,a)| \leq \kappa$, then the resulting update satisfies $\|Q' - Q^*\|_\infty \;\leq\; \frac{2\kappa}{1-\gamma}$. Hence, to guarantee a final error $\|Q' - Q^*\|_\infty \leq \xi$, is enough to set $\kappa = \frac{\xi(1-\gamma)}{2}$.

Now consider the event $E_{s,a}$ as the "bad event" where a Q-value estimate for a pair (s,a) exceeds the error κ. By Lemma 4.4, we know that $\Pr(E_{s,a}) \leq 2e^{-2N(s,a)\kappa^2/V_{max}^2}$. Now we want to guarantee the total probability of any bad event occurring across all $|\mathcal{S}||\mathcal{A}|$ pairs to be at most δ. By union bound, we have $\Pr(\bigcup_{s,a} E_{s,a}) \leq \sum_{s\in\mathcal{S}, a\in\mathcal{A}} \Pr(E_{s,a})$. Hence, substituting δ with $\frac{\delta}{|\mathcal{S}||\mathcal{A}|}$ for each (s,a) we enforce that

$$2\exp\left(-\frac{2N(s,a)\xi(1-\gamma)^2}{2V_{\max}^2}\right) \;\leq\; \frac{\delta}{|\mathcal{S}||\mathcal{A}|}, \tag{4.14}$$

so that a union bound across all $|\mathcal{S}||\mathcal{A}|$ pairs ensures failure probability at most δ, as desired. Solving for $N(s,a)$ yields exactly the claimed bound. □

Connecting the finite-sample bound to ε-retrain. We consider an episodic MDP with horizon H and a behavior policy π. Let the per-episode initial state be drawn from the mixed restart distribution $d_\varepsilon = (1-\varepsilon)\,\rho \;+\; \varepsilon\,\overline{\rho}$. For a fixed policy π, define the (per-episode) occupancy measure up to horizon H:

$$d^{\pi}_{\varepsilon,H}(s,a) \;:=\; \frac{1}{H}\,\mathbb{E}_{s_0\sim d_\varepsilon,\,\pi}\left[\sum_{t=0}^{H-1} \mathbf{1}\{s_t = s,\; a_t = a\}\right].$$

Thus, in M episodes, the expected visit count to (s,a) is $\mathbb{E}[N(s,a)] \;=\; M\,H\,d^{\pi}_{\varepsilon,H}(s,a)$.

Proposition 4.2 (Sample complexity under ε-retrain): *Fix $\xi > 0$ and $\delta \in (0, 1)$ and let N_0 be the per–(s, a) sample threshold. From Lemma 4.3 it holds that*

$$N_0 \;=\; \frac{2V_{\max}^2}{\xi^2(1-\gamma)^2}\,\ln\!\left(\frac{2|\mathcal{S}||\mathcal{A}|}{\delta}\right).$$

Assume episodes are independent through restarts from d_ε and that within each episode the process follows π for H steps. Then, for any (s, a) and any $\eta \in (0, 1)$, by a multiplicative Chernoff bound [194],

$$\Pr\!\Big(\, N(s,a) \;\le\; (1-\eta)\, M\, H\, d^{\pi}_{\varepsilon,H}(s,a) \,\Big) \;\le\; \exp\!\left(-\frac{\eta^2\, M\, H\, d^{\pi}_{\varepsilon,H}(s,a)}{2}\right).$$

Consequently, if

$$M \;\ge\; \frac{2}{\eta^2\, H\, d^{\pi}_{\varepsilon,H}(s,a)}\,\ln\!\left(\frac{|\mathcal{S}||\mathcal{A}|}{\delta}\right) \quad \textit{and} \quad (1-\eta)\, M\, H\, d^{\pi}_{\varepsilon,H}(s,a) \;\ge\; N_0,$$

then with probability at least $1 - \delta$, every pair (s, a) satisfies $N(s, a) \ge N_0$, hence $\|Q' - Q^*\|_\infty \le \xi$ by Lemma 4.3.

Proof. For fixed (s, a), the per-episode visit indicator sequence can be written as a sum of H bounded Bernoulli variables whose expectation is $H\, d^{\pi}_{\varepsilon,H}(s, a)$. Across independent episodes, $\{N(s, a)\}_M$ is a sum of i.i.d. bounded variables with mean $H\, d^{\pi}_{\varepsilon,H}(s, a)$. Apply a standard multiplicative Chernoff bound (see, e.g., [194, section 4.2.1]) to obtain the stated tail probability. Choose M so that (i) the tail is at most $\delta/(|\mathcal{S}||\mathcal{A}|)$, and (ii) the lower tail value exceeds N_0. The claim follows by union bound over all (s, a) and the derivation of Lemma 4.3. □

Why ε-retrain improves performance. The mixed restart distribution *directly* boosts occupancy in targeted regions $d^{\pi}_{\varepsilon,H}(s, a) \;=\; (1 - \varepsilon)\, d^{\pi}_{H}(s, a; \rho) \;+\; \varepsilon\, d^{\pi}_{H}(s, a; \overline{\rho})$, where $d^{\pi}_{H}(s, a; \nu)$ denotes the H-step occupancy under initial distribution ν. Hence, for any (s, a) that is reachable with nonzero probability under $\overline{\rho}$ and π, $d^{\pi}_{\varepsilon,H}(s, a) \;\ge\; \varepsilon\, d^{\pi}_{H}(s, a; \overline{\rho})$. Plugging this lower bound into Proposition 4.2 yields an explicit episode bound:

$$M \;\ge\; \max\!\left\{\frac{2}{\eta^2\, H\, \varepsilon\, d^{\pi}_{H}(s,a;\overline{\rho})}\,\ln\!\Big(\frac{|\mathcal{S}||\mathcal{A}|}{\delta}\Big),\quad \frac{N_0}{(1-\eta)\, H\, \varepsilon\, d^{\pi}_{H}(s,a;\overline{\rho})}\right\}.$$

Thus, choosing $\overline{\rho}$ to place mass on states where the agent underperforms (violates preferences) *provably increases* $d^{\pi}_{\varepsilon,H}(s, a)$ for those regions, reducing the episodes M needed to surpass N_0 and thereby shrinking the local value error ξ as per Lemma 4.3.

In summary, applying ε-`retrain` to value-based algorithms highlights two key insights:

- The convergence of Q-learning fundamentally relies on visiting every relevant state–action pair infinitely often. In practice, this condition translates into visiting them "sufficiently often." The mixed restart distribution $(1-\varepsilon)\rho + \varepsilon\overline{\rho}$ directly shapes these visitation counts $N(s,a)$ by biasing exploration toward particular regions of the state space.
- By defining $\overline{\rho}$ as a distribution over states where the agent violates behavioral preferences, ε-`retrain` explicitly drives exploration into the parts of the state space most critical for safe and reliable learning. This ensures that the visitation counts $N(s,a)$ for these pairs grow rapidly during the early phases of training, helping them reach the minimum threshold N_0 required for convergence more quickly.

In this way, ε-`retrain` not only preserves the convergence guarantees of value-based RL but also accelerates learning in practice by focusing updates on the regions of the state space most relevant for satisfying behavioral preferences.

4.3.2 Limitations

We identify the three following limitations in our work:

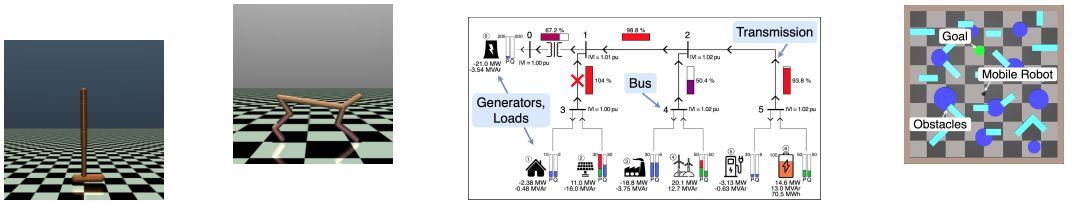
- Our algorithm requires a simulator to train the agent and the possibility of resetting the system to specific states. We believe this requirement is reasonable in the RL literature.
- In *ε-retrain* , we assume having access to an additional indicator cost signal from the environment. Such a signal is widely adopted in the safe RL literature, where system designers assume having access to a cost function that deems a state-action pair as safe or unsafe [6, 223, 114].
- We assume the area surrounding a collision state is also prone to violations of the desiderata. When such an assumption does not hold (e.g., in highly non-linear systems such as power grids), we use the exact feature values to determine a retrain area instead of intervals of size ω.

4.3.3 Experiments

We *present a comprehensive evaluation of ε-retrain applied to TRPO* [231] that approximates the policy optimization theory of Sec. 4.3.1, *and PPO* [232] which relaxes the computational demands of TRPO.[16] We refer to these methods as ε-TRPO and ε-PPO. *Additionally, we investigate the impact of the proposed approach on top and against safe RL baselines*, using the Lagrangian method with TRPO and PPO and modeling behavioral preferences as constraints. The resulting algorithms are named TRPOLagr, PPOLagr, ε-TRPOLagr, and ε-PPOLagr. Our experiments address the following questions:

- Does *ε-retrain* allow agents to better adhere to behavioral preferences while solving the task in both an unconstrained (where we penalize the reward upon violating the preference) and constrained formalization?
- How does *ε-retrain* impact existing CMDP-based methods aimed at satisfying these preferences?
- How often do agents satisfy the behaviors provably and empirically?
- How do *ε-retrain* parameters influence performance?

To answer these questions, we begin our experiments using two known safety-oriented tasks, "SafetyHopperVelocity-v1, and "SafetyHalfCheetahVelocity-v1", from the Safety-Gymnasium benchmark [113]. To evaluate our method in a variety of setups and different behavioral desiderata, we also employ two practical scenarios based on an active network management task for a power system [101], and navigation for a mobile robot [165]. Fig. 4.30 shows these tasks. For simplicity, we will refer to them as *Hopper*, *HalfCheetah* (Cheetah), *Active Network Management* (ANM), and *Navigation*, respectively.



(a) Hopper (b) HalfCheetah (c) Active Network Management (d) Navigation

Fig. 4.30: Environments employed in our experiments.

In the following, we briefly describe the tasks and the desired behavioral specifications, referring to the original works for more details [113, 101, 168].

- **Hopper, HalfCheetah (Fig.s 4.30a, b).** The robots have to learn how to run forward by exerting torques on the joints and observing the body parts' angles and velocities (for a total of 12 and 18 input features). The actions control the torques applied to the (3 and 6) joints of the robot. The agents are rewarded based on the distance between two consecutive time steps (i.e., positive and negative

[16] We also tested *vine* TRPO, which achieved comparable results with lower sample efficiency than TRPO. For this reason, our evaluation considers the TRPO algorithm.

values for forward and backward movements). In these tasks, we have a velocity preference—agents should not go faster than $0.7402\frac{m}{s}$ and $3.2096\frac{m}{s}$, respectively. These are the same limitations considered in the safe RL literature [114], and the indicator cost deeming a velocity violation triggers when the limits are violated. The positive cost has a three-fold role: it (i) starts the generation of a retrain area as described in Sec. 4.3.3, using an initial size $\phi = 0.01$ given by our initial grid search; (ii) triggers a small reward penalty to encourage the unconstrained baselines to avoid such an undesired behavior; and (iii) gets accumulated to model the constraints of the Lagrangian implementations. As in relevant literature, we set the constraints threshold to 25.

- **Active Network Management (Fig. 4.30c).** The agent has to reschedule the power generation of different renewable and fossil generators, to satisfy the energy demand of three loads connected to the power grid. Specifically, we observe the state of the power network through 18 features (i.e., active and reactive power injections, charge levels, and maximum productions). Moreover, we control power injections and curtailments using 6 continuous actions.The behavioral preference here models energy losses and a grid's operational and transmission constraints. For this reason, we want our agent to limit penalties associated with violating these constraints. This task is significantly more complex than the previous ones since the agent is negatively rewarded based on the energy losses, the generation cost, and the violation of the operational constraints—however, to successfully solve the task in this particular environment, the agent *must* receive penalties associated with energy losses, which are a natural consequence of power transmission. The indicator cost has the same role and consequences as the previous environments and is triggered when the agent violates the system's operational constraints. Based on the performance of the unconstrained baselines, we set the constraints threshold to 350. Due to the non-linear dynamics of power networks, a retrain area is generated using the exact violation state, with an initial bubble size $\omega = 0$.
- **Navigation (Fig. 4.30d).** A mobile robot has to control its motor velocities to reach goals that randomly spawn in an obstacle-occluded environment without having a map. The agent observes the relative position of the goal and sparse lidar values sampled at a fixed angle (for a total of 22 features) and controls linear and angular velocity using 2 continuous actions. Intuitively, the behavioral preference here models a safe behavior since we want the robot to avoid collisions. Similar to the previous environments, the agent is positively (or negatively) rewarded based on its distance from the goal in two consecutive timesteps. The indicator cost has the same consequences as the previous environments, and it is triggered upon every collision. We set the constraints threshold to 20, indicating the robots should not collide for more than 20 steps in a training episode that lasts for 500 steps. We consider the simulated lidar precision to initialize the bubble size $\omega = 0.025$.

Implementation Details

Data collection is performed on Xeon E5-2650 CPU nodes with 64GB of RAM. For policy-based baselines, we rely on existing implementations of PPO, TRPO,

and their Lagrangian variants from the `omnisafe` library [114]. For value-based baselines, we consider a CleanRL implementation of DDQN [108]. A complete list of hyperparameters is provided below.

Hyperparameters

Regarding the baselines, we performed an initial grid search, but the original parameters of the Omnisafe library resulted in the best performance [114]. Table 4.6 lists the key hyper-parameters considered in our initial grid search for TRPO, PPO, their Lagrangian versions, and ε-retrain. The best parameters used in our evaluation are highlighted in the last column.

	Parameter	Grid Search	Best Values
Policy Optimization	Steps per epoch	20000, 30000	20000
	Update iterations	10, 20	10
	Batch size	64, 128	128
	Target KL	0.01, 0.001	0.01
	Max grad. norm	20, 40	40
	γ	0.9, 0.95, 0.99	0.99
	GAE	0.95	0.95
	PPO clip	0.2	0.2
	Penalty	0.0, 0.1, 0.25	0.1
DDQN	Buffer size	250000, 500000	500000
	Batch size	64, 128	128
	γ	0.9, 0.95, 0.99	0.99
	Penalty	0.0, 0.1, 0.25	0.1
Networks	N° layers	2	2
	Size	64, 128	64
	Activation	tanh	tanh
	Optimizer	Adam	Adam
	Learning rate	1e-3, 3e-4, 5e-5	3e-4
Lagrangian	Multiplier init.	0.001	0.001
	Multiplier learning rate	0.035, 0.0035	0.035
	Cost limits	-	Section 4.3.3
	Cost γ	0.9, 0.99	0.99
	Cost GAE	0.95	0.95
ε-retrain	Bubble size init ω	0.0, 0.01, 0.025	0.0, 0.01, 0.025
	Similarity β	0.03, 0.06, 0.12	0.03
	Max retrain areas	500, 1250, 2500	500
	ε-decay	0.25, 0.5, 0.75	0.75
	Minimum ε	0.25, 0.5, 0.75	0.5

Table 4.6: Hyper-parameters candidate for initial grid search tuning, and best parameters.

We report the average return, cost, and standard error as shaded regions over 50 independent runs per method. Fig.s 4.31 and 4.34 (the latter reported in the supplementary) show the average return in the first row and the average cost in the second row, where each column represents a different task. Notably, *we are seeking agents that achieve a lower cost, which indicates they better adhere to the desired behavioral preferences while also solving the task*. Our claims on the performance improvement of ε*-retrain* are supported by over 1800 independent training runs, far exceeding the 3–10 runs per method commonly reported in prior policy optimization work [231, 232]. We note that due to employing a small penalty in the reward function to encourage specific behavioral preference, our results are not directly comparable to the published baselines [231, 232]. For a fair comparison, we first collect the baseline with this new setting and then compare the performance with our approach. Considering the computational resources used for our extensive evaluation, we address the environmental impact of our experiments.

Environmental Impact

Despite each individual training run being "relatively" computationally inexpensive due to the use of CPUs, the $\approx$ 1800 experiments of our evaluation led to cumulative environmental impacts due to computations that run on computer clusters for an extended time. Our experiments were conducted using a private infrastructure with a carbon efficiency of $\approx 0.275 \frac{\text{kgCO}_2\text{eq}}{\text{kWh}}$, requiring a cumulative $\approx$260 hours of computation. Total emissions are estimated to be $\approx$ 7.21kgCO_2eq using the Machine Learning Impact calculator, and we purchased offsets for this amount through Treedom.

Empirical Evaluation

Performance of ε-TRPO and ε-PPO.

Fig. 4.31 shows that TRPO and PPO enhanced with our ε*-retrain* improve sample efficiency and allow agents to better adhere to the behavioral preferences. In more detail, in Hopper and HalfCheetah, TRPO and PPO achieve substantially higher returns than their ε*-retrain* version; a result that could be easily misunderstood. In fact, this is related to the nature of the task, where the reward is directly proportional to the agents' velocities. For this reason, an agent that violates the behavioral preference *"limit velocity under a threshold"*, achieves higher returns.

This is clearly shown in the first two columns of Fig. 4.31, where ε-TRPO and ε-PPO resulted in notably lower cost compared to the baselines, indicating they lead the agents towards adhering to the velocity limit significantly more often than TRPO and PPO. Similar results are achieved in the ANM task where ε-TRPO and ε-PPO are notably safer and more sample efficient as shown in the Pareto frontier reported in Fig.4.32. Specifically, we report on the y-axis the average reward and on the x-axis the average cost at convergence. ε-retrain in general allows to achieve the best trade-off between average reward and cost (no other methods reach better performance in the upper-left corner), i.e., less violation of the behavioral desiderata while still successfully solving the task. It is important to notice that only in the navigation scenario the cost value of TRPO at convergence is slightly lower than

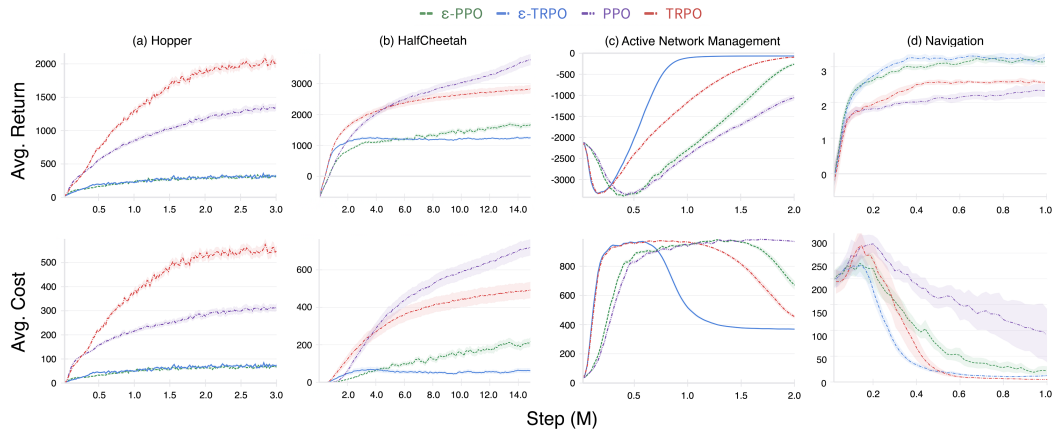


Fig. 4.31: Comparison of ε-PPO, ε-TRPO, PPO and TRPO.

$\varepsilon-$TRPO one. However, as reported in the learning curves, our approach results in significantly more sample efficiency than the baseline counterparts.

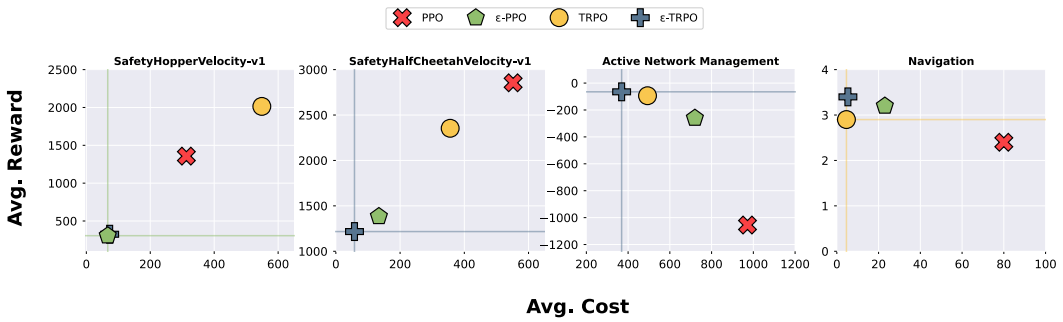


Fig. 4.32: Pareto frontier of ε-PPO, ε-TRPO, PPO and TRPO at convergence in four different environments. Each column (i.e., each task) shows the average reward and cost during the training. Learning curves are reported in Fig. 4.31.

The benefits of *ε-retrain* are also confirmed in the navigation task, where violating the safety desiderata *"avoid collisions"* leads to more collisions. Ultimately, achieving a higher cost (i.e., more collisions) hinders the navigation performance of the agent and leads to lower returns. Specifically, TRPO and ε-TRPO converge to the same average cost. However, the higher sample efficiency of the latter through the training, in terms of learning collision avoidance behaviors more quickly, allows ε-TRPO to learn better navigation behaviors, outperforming TRPO in terms of average return. Moreover, ε-PPO significantly outperforms PPO both in terms of average cost and return.

Performance of ε-DDQN Fig. 4.33 shows that DDQN enhanced with our `ε-retrain` also improves sample efficiency and allows agents to better adhere to the behavioral preferences. These results confirm what discussed in the policy-based case, with ε-DDQN outperforming the baseline DDQN in terms of average reward and cost in both tasks.

Fig. 4.33: Comparison of ε-DDQN and DDQN in two different environments. Each column (i.e., each task) shows the average reward and cost during the training.

Performance of ε-TRPOLagr and ε-PPOLagr.

During training for the Lagrangian algorithms, both ε-TRPOLagr and ε-PPOLagr drastically reduce the amount of constraint violations in the Hopper and HalfCheetah velocity environments as shown in Fig. 4.34. In detail, we report the training performance of original constrained policy optimization algorithms and the one enhanced with ε-retrain. Our approach allows us to reduce the constraint violations and, in the more complex tasks, to improve performance.

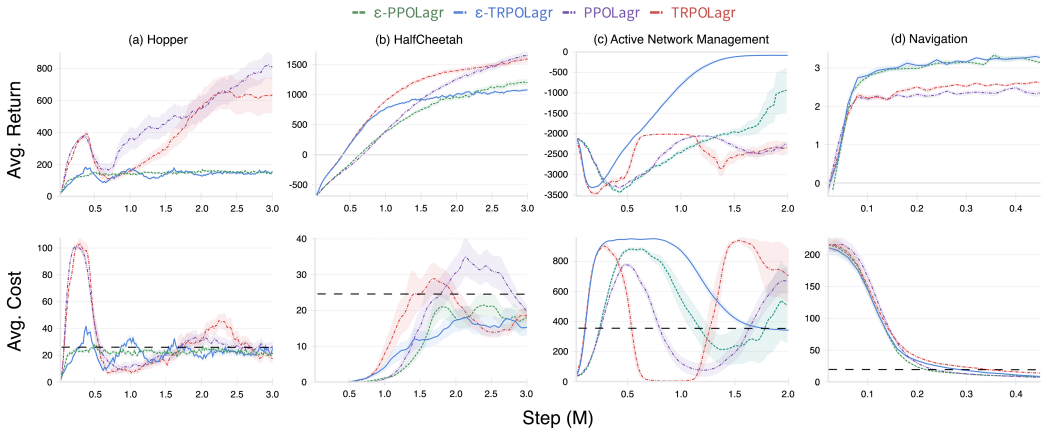


Fig. 4.34: Comparison of ε-PPOLagr, ε-TRPOLagr, PPOLagr and TRPOLagr in four different environments. Each column (i.e., each task) shows the average reward and cost during the training. The black dotted line represents the cost threshold.

To see this aspect in practice, we specify the fraction of training steps where agents violate their constraint in Table 4.7.

However, at convergence, all the approaches satisfy the imposed thresholds. Fig. 4.35 shows the Pareto frontier reporting on the y-axis the average reward and on

	Hopper	Cheetah	ANM	Navigation
PPOLagr	0.57	0.33	**0.44**	0.46
ε-PPOLagr	**0.04**	**0**	0.59	**0.44**
TRPOLagr	0.51	0.17	**0.58**	0.64
ε-TRPOLagr	**0.25**	**0**	0.79	**0.56**

Table 4.7: Average fraction of the training steps where agents violate the constraints (lower is better).

the x-axis the average cost at convergence. These results lead to some interesting considerations based on the setup of interest. In safety-critical contexts where it is crucial to satisfy constraints at training time, *ε-retrain* showed significant empirical benefits.

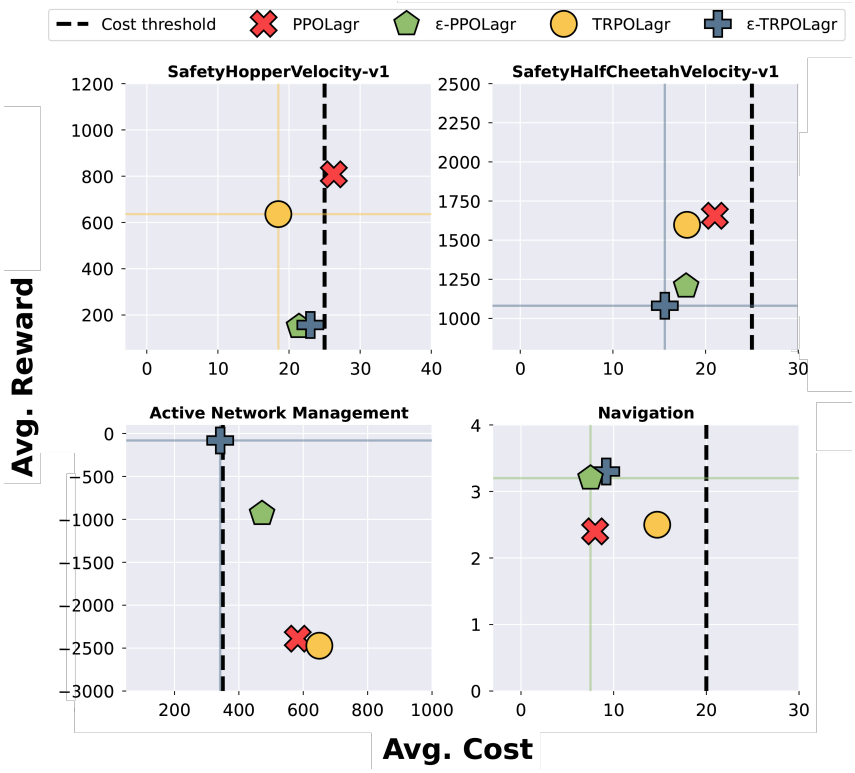


Fig. 4.35: Pareto frontier of reward versus cost for ε-PPOLagr, ε-TRPOLagr, PPOLagr and TRPOLagr at convergence.

On the other hand, in non-critical contexts where performance at convergence is the main evaluation metric, the naive Lagrangian methods have superior return performance. Intuitively, this relates to the fact that Lagrangian methods often violate the constraints at training time, allowing agents to explore more and thus learn higher-performing behaviors. In the more complex, realistic scenarios, our empirical analysis leads to different considerations. Specifically, in navigation, *ε-retrain*-based methods and the Lagrangian baselines achieve comparable results in

terms of cost (i.e., constraint satisfaction). However, retraining agents in areas that are collision-prone allowed them to learn policies with better navigation skills and higher performance. In the ANM task, retraining an agent during grid instability increases the frequency of constraint violations compared to the baseline. Nonetheless, our approach helps agents learn to manage the grid effectively over time in contrast to Lagrangian baselines, which in the end, fail to solve the problem efficiently.

Provably Verifying Navigation Behaviors

To further assess the benefit that ε-*retrain* has over the behavioral preferences, we formally verify the policies trained for the navigation task. We consider this problem as an explanatory task for clarity since it allows us to easily visualize the retrain areas generated for *"collision avoidance"* on top of the environment.

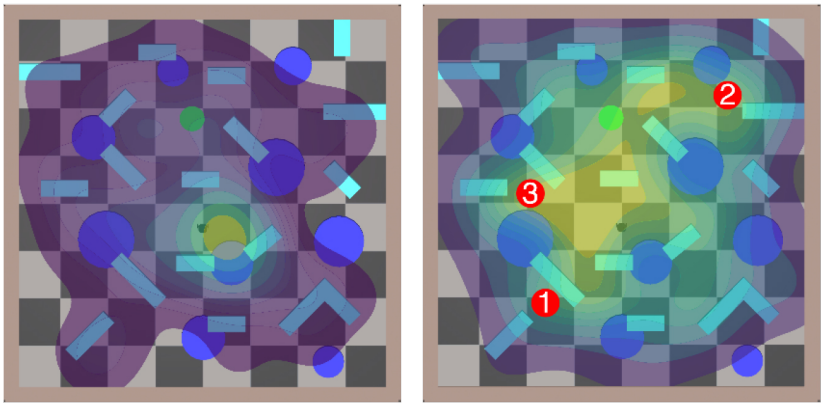


Fig. 4.36: Density map of the retrain areas collected in the first and last training epochs; yellow indicates higher density.

Fig. 4.36, shows a kernel density estimation map of the retrain areas distribution at the beginning (left) and final stages (right) of the training for the ε-TRPO agent. Here we can notice how the agent successfully learns to navigate the environment over time since the retrain areas are more equally distributed through the entire scenario. Using a recent verification tool [173], we aim to quantify the probability that a navigation policy violates the collision avoidance preference. To this end, we consider three representative retrain areas collected while training the different ε-*retrain*-based algorithms (depicted as red dots in Fig. 4.36). For each area, we encode the input-output relationship, considering the retrain area as the precondition, and the minimum linear and angular velocities that would cause a collision as the postcondition. Broadly speaking, the FV tool checks where the trained policies do not exceed such minimum velocities (i.e., they do not collide), and returns the portion of each retrain area for which the given policy violates the postcondition (i.e., the probability of colliding in that area).

Table 4.8 reports the probability that policies at convergence collide in the chosen retrain areas, averaged over all the runs. This additional FV-based analysis shows that

	Retrain areas (1, 2, 3)		
ε-PPO	**0.007%**	**0.011%**	**0.22%**
PPO	0.012%	0.017%	0.59%
ε-TRPO	**0.014%**	**0.67%**	1%
TRPO	0.015%	0.69%	**0.8%**
ε-PPOLagr	**0.006%**	**0%**	**0.012%**
PPOLagr	0.013%	0.05%	0.1%
ε-TRPOLagr	0.0004%	0**.007%**	**0.46%**
TRPOLagr	**0.00005%**	0.012%	0.58%

Table 4.8: Average behavioral violations percentage for policies trained with TRPO, PPO, PPOLagr, TRPOLagr, and their ε-*retrain* version (ours).

ε-*retrain* algorithms better adhere to the behavioral preference, further confirming our intuitions and the merits of our approach.

4.3.4 Sensitivity analysis of ε-parameters

We also conducted an additional empirical study in the mapless navigation environment to examine how the parameters introduced by ε-`retrain` influence performance, using our best ε-TRPO algorithm. This analysis allows us to assess not only the effectiveness but also the robustness of the method with respect to design choices such as ε decay, maximum number of retrain areas, minimum value of ε, bubble size, and similarity threshold.

The results are reported in Fig. 4.37. Overall, the parameter configuration selected through our grid search, shown with the ε-TRPO label, achieves the best performance across the evaluated metrics. Importantly, however, the performance differences among alternative parameter settings are relatively small. This indicates that ε-`retrain` is not overly sensitive to initialization and can deliver consistent improvements across a broad range of hyperparameter choices. This robustness further supports the practicality of deploying ε-`retrain` in diverse real-world scenarios without requiring extensive parameter tuning.

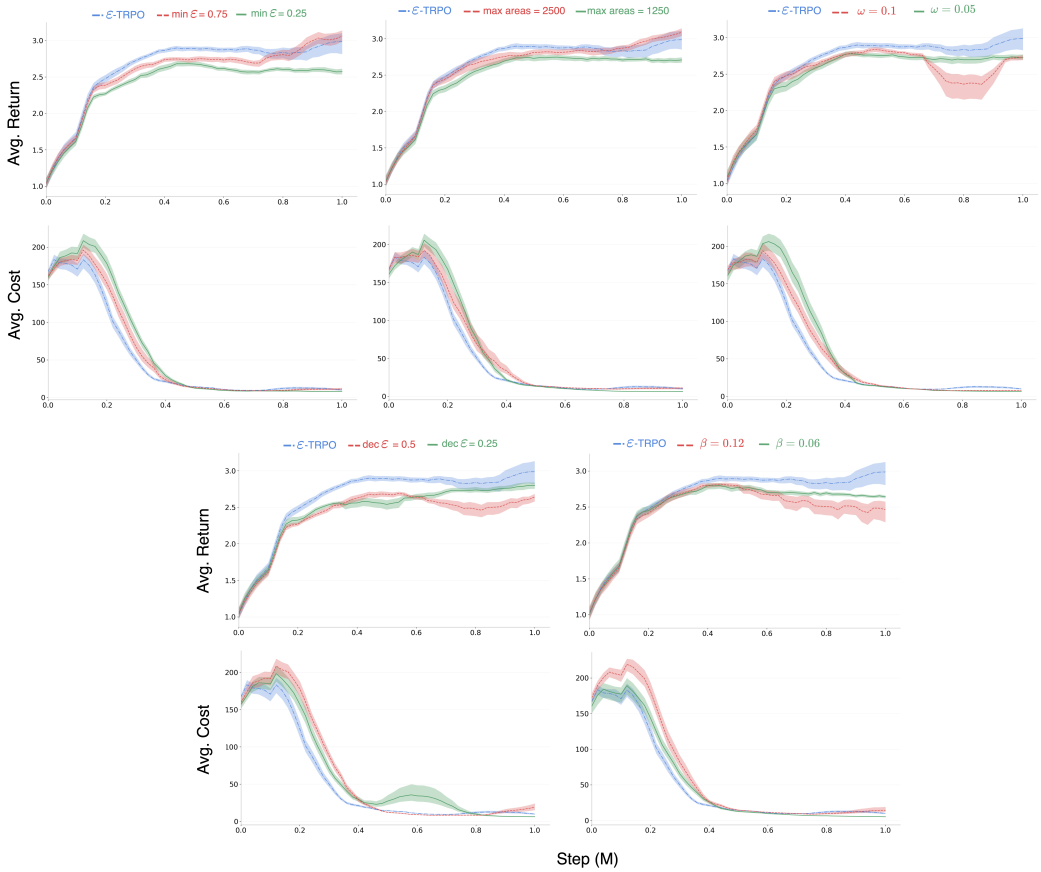


Fig. 4.37: Sensitivty analysis of ε-`retrain` parameters. The evaluation is conducted over 10 independent seeds per each algorithm (as such, ε-`retrain` TRPO shows slightly different performance than those in Fig. 4.31).

Real (embodied) experiments

To conclude our comprehensive evaluation, we perform an additional evaluation in realistic (embodied) unsafe mapless navigation settings. Due to the similar performance at convergence for TRPO and ε-TRPO in our simulated evaluations, we choose these two approaches for comparison. Specifically, we compare ε-TRPO and TRPO in scenarios where the agent has either all or only partially occluded LiDAR information. We hypothesize that if the agent is not exposed to multiple unsafe situations during the training, i.e., without an *ε-retrain* strategy, it is less likely to select a longer but safer trajectory, and eventually, the agent will prefer a straight trajectory leading to a collision. To test this, using the Unity framework [126] we used to create the navigation task, we transfer the policies trained in simulation onto ROS-enabled platforms such as our Turtlebot3. We then test several corner-case situations, comparing the safer (from the formal verification results) trained agent at convergence for both ε-TRPO and TRPO. In our experiments, we observe our hypothesis to be correct (see Fig. 4.46), showing the benefit of retraining the agent in specific regions of the state space that are deemed unsafe.

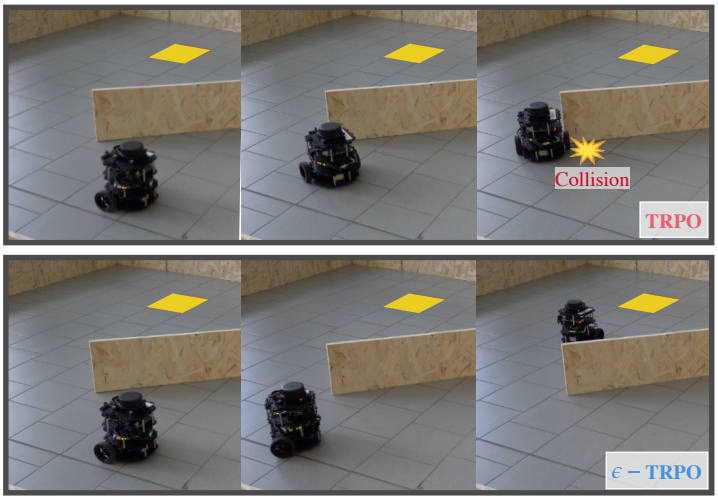


Fig. 4.38: Real-world experiments comparing ε-TRPO and TRPO in corner-case scenarios. Video available here.

Summary. This section presented ε-`retrain`, a lightweight exploration strategy designed to improve RL agents by explicitly encouraging adherence to behavioral preferences. The central idea is to retrain agents from *retrain areas*—regions of the state space where preferences have previously been violated. By doing so, ε-`retrain` directs training toward precisely those situations that matter most for satisfying the preferences, while maintaining the theoretical guarantees of the underlying algorithms.

We introduced and analyzed ε-`retrain` for policy-based RL and value-based methods. On the policy side, we showed that ε-`retrain` retains monotonic improvement guarantees and improves adherence to preferences when integrated with standard and constrained policy optimization algorithms. On the value-based side, we extended the framework to Q-learning and related methods, proving that ε-`retrain` preserves convergence to the optimal action-value function. We further provided a sample-complexity analysis that clarifies how emphasizing retrain areas accelerates learning by increasing visitation counts in critical regions of the state space. Our extensive empirical evaluation confirmed these theoretical insights. Across locomotion, power network management, and navigation tasks, ε-`retrain` consistently improved sample efficiency and adherence to behavioral preferences for both policy and value-based methods. A sensitivity study further demonstrated that the performance of ε-`retrain` is robust to different parameter choices, easing deployment in practice. Finally, we validated the approach in embodied navigation experiments on real robots, where policies trained with ε-`retrain` selected safer trajectories and better avoided unsafe states.

In summary, ε-`retrain` offers a general and practical mechanism for embedding behavioral preferences into reinforcement learning. By focusing training on the regions of the state space where agents are most likely to fail, it improves both learning efficiency and safety, without compromising the guarantees of the base algorithm. We believe this aspect positions ε-`retrain` as a promising step toward deploying reinforcement learning in safety-critical real-world applications.

4.4 Designing Control Barrier Function via Probabilistic Enumeration for Safe Reinforcement Learning Navigation

Achieving safe autonomous navigation is a crucial requirement to deploy robots in the real world, where human interactions and expensive hardware are often involved. For example, robots can assist humans in monitoring and mitigating pollution in a variety of environmental sustainability scenarios (e.g., aquatic ecosystems [32]) or perform complex autonomous navigation in precision agriculture tasks [212]. Traditionally, these applications are based on manual human intervention, which is costly and struggles to capture real-time changes reliably. These are challenges that a drone equipped with a safe autonomous navigation stack can successfully address [90].

To this end, online model-based learning methods embedding hierarchical control architectures have been used to deal with uncertainty in various navigation tasks [21, 256, 257, 258]. Nevertheless, these methods rely on extensive domain knowledge, which could be difficult to obtain in practice. To address this issue, deep reinforcement learning algorithms have been employed to learn how to navigate in complex and unknown (i.e., mapless) environments [222, 113, 114]. However, DRL-based policies are modeled as deep neural networks, which are known to be vulnerable to adversarial inputs—small perturbations to the input state leading to unexpected and unsafe actions [246]. Two main lines of work have been investigated to address such an issue. On the one hand, formal verification of neural networks can certify safety, providing rigorous theoretical guarantees that a DNN-based navigation policy lies within predefined safety constraints [150]. However, FV scales poorly, and novel probabilistic verification approaches have recently gained traction to address the limits of formal methods [284]. In detail, probabilistic enumeration introduced in Sec.3.5 can identify regions of the state space (i.e., parts of the environment) where a DNN exhibits unsafe behavior with a high degree of confidence, but this information has not been exploited to correct unsafe navigation behaviors. On the other hand, control barrier functions (CBFs) [7] also provide a principled way to enforce safety constraints by specifying a function that characterizes a safe set, ensuring its forward invariance [7]. Hence, CBFs can potentially guarantee that navigation trajectories remain within pre-defined safety regions. However, designing effective CBFs for high-dimensional, complex environments remains challenging when dealing with uncertainty and complex dynamics.

To mitigate these issues, recent work has explored the use of neural CBFs [151]. In detail, [96] proposed to learn a neural CBF used as a reactive safety filter that enforces collision avoidance for aerial robots. On top of these approaches, [57] proposed to enhance the safety of learning RL agents using a safety shield approach combining the robustness of the model predictive control (MPC) approach with the predictive capabilities of RL. Despite these different lines of research, correcting agents' actions upon detecting an unsafe situation at deployment time remains an open research question.

We address this gap by leveraging the advantages of probabilistic verification and CBF-based control, introducing a framework that guarantees safe DRL-based navigation policies and correct policy actions in potentially unsafe situations (Fig.

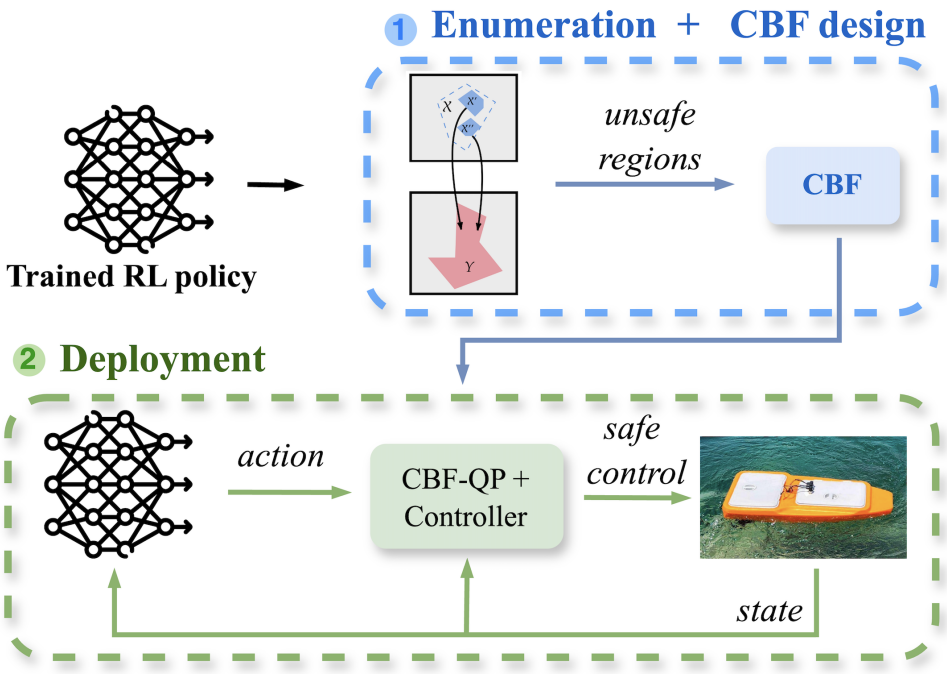


Fig. 4.39: Overview of the proposed approach.

4.39). At a high level, we compute a safe navigation set by first identifying unsafe regions through offline probabilistic enumeration presented in Sec.3.5 and then removing them from the state space of the DRL policy. We then design a CBF-based control mechanism to ensure the agent avoids both these pre-identified unsafe regions and any obstacles detected during navigation—ensuring the agent remains within the precomputed safe set. At deployment time, a quadratic programming (QP) optimization incorporating the CBF's safety constraints evaluates the policy's action, determining whether the chosen action keeps the agent within the precomputed safe set or not. If the action violates any safety constraints, a low-level controller modifies the action to ensure the agent returns to the safe set while maintaining effective navigation behaviors.

A key advantage of our approach is that the enumeration component, which employs an abstract interval representation to identify the (un)safe regions, is agnostic to the specific navigation environment where the agent is deployed, thus enhancing generalization. Furthermore, the proposed framework works on top of arbitrary DRL policies, and it is orthogonal to both the training and verification strategies employed by a neural CBF for the execution of safe policies.

To confirm the effectiveness of the proposed approach, we first validated our strategy in two different Unity-based simulated navigation scenarios using a mobile Turtlebot3 and an aquatic drone for water monitoring. We then performed a real-world deployment of a trained DRL policy to validate the correctness of our pipeline. Our empirical results demonstrate that the proposed method enables safe and scalable autonomous robotic navigation, achieving zero violations of safety restrictions at deployment while increasing navigation effectiveness.

4.4.1 Preliminaries

Control Barrier Function

Consider an affine nonlinear system of the form:

$$\dot{x} = f(x) + g(x)u, \tag{4.15}$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^n$ represents the state variable within the dynamics model state space $\mathcal{X}$, and $u \in \mathcal{U} \subseteq \mathbb{R}^m$ is the control input within the control input space $\mathcal{U}$. The functions $f(x)$ and $g(x)$ are nonlinear mappings from $\mathbb{R}^n$ to $\mathbb{R}^n$ and are assumed to be Lipschitz continuous. In this setting, a CBF is defined as follows.

Definition 4.4 (Control Barrier Function [7]): *Let $C \subset \mathbb{R}^n$ be the set defined by a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$ such that*

$$\begin{aligned} C &= \{x \in \mathbb{R}^n : h(x) \geq 0\}, \\ \partial C &= \{x \in \mathbb{R}^n : h(x) = 0\}, \\ Int(C) &= \{x \in \mathbb{R}^n : h(x) > 0\}. \end{aligned} \tag{4.16}$$

We say that h is a control barrier function if $\nabla h(x) \neq 0$ for all $x \in \partial C$ and there exists a class-$\mathcal{K}$ function α, such that for all $x \in C$, $\exists u$ such that $L_f h(x) + L_g h(x) u \geq -\alpha(h(x))$, with $L_f h(x) := \nabla h(x)^T f(x)$ and $L_g h(x) := \nabla h(x)^T g(x)$.

In other words, consider a single integrator system as an explanatory example $\dot{x} = u, \quad x \in \mathbb{R}, \ u \in \mathbb{R}$. Define the safe set as $C = \{x \in \mathbb{R} : h(x) = 1 - x^2 \geq 0\}$, which corresponds to keeping the state within the interval $[-1, 1]$. Let $h(x) = 1 - x^2$. Then we have $\nabla h(x) = -2x, \quad f(x) = 0, \quad g(x) = 1$. Therefore, the Lie derivatives become $L_f h(x) = 0, \quad L_g h(x) = -2x$. The CBF condition requires that there exists a control input u such that $L_f h(x) + L_g h(x) u = 0 + (-2x)u \geq -\alpha(h(x))$, for some class-$\mathcal{K}$ function α. Choosing $\alpha(h) = h$, the condition becomes $-2xu \geq -(1 - x^2)$. This implies $u \leq \frac{1-x^2}{2x}$ for $x > 0$, $u \geq \frac{1-x^2}{2x}$ for $x < 0$. Thus, the control input u must satisfy this inequality to ensure that the state remains within the safe set C. This formalization will be used in Sec 4.4.3 to define the CBF for navigation policies.

Dynamic Models

The design of the proposed pipeline requires the definition of the dynamic models to compute a corrective action for the navigation policy. Specifically, in this work, we consider (i) the kinematic equations of a mobile robot (Turtlebot3) and (ii) the nonlinear dynamics model of an aquatic drone.

Mobile robot

The kinematics equations of the robot are represented by $\dot{p}_x = v_1 \cos\theta, \quad \dot{p}_y = v_1 \sin\theta, \quad \dot{\theta} = \omega_3$, where p_x, p_y, θ are the positions and orientation of the robot and $\boldsymbol{u} = [v_1, \omega_3]$ are the controlled linear and angular velocities.

Aquatic drone

The aquatic drone motion is described following the 6-degrees-of-freedom nonlinear dynamic model and the fluid dynamics coefficients are computed through spline interpolation using a high-fidelity simulator following common formalisms in the literature [259, 83]. Hence, the aquatic drone state and control vector are defined as:

$$\begin{aligned} \boldsymbol{x} &= [v_1 \quad v_2 \quad v_3 \quad \omega_1 \quad \omega_2 \quad \omega_3 \quad p_x \quad p_y \quad p_z \quad \phi \quad \theta \quad \psi]^T, \\ \boldsymbol{u} &= [\delta_l \quad \delta_r]^T, \end{aligned} \tag{4.17}$$

where v_1, v_2, v_3 are linear velocities and $\omega_1, \omega_2, \omega_3$ are the rate of the angles in the body frame, respectively. With p_x, p_y, p_z and ϕ, θ, ψ, we refer to the boat position and the Euler angles in the Earth frame. Regarding the control variables, δ_l and δ_r represent the left and right thrusts of the boat. For readability purposes, in the next we call $a = (\cos\psi \sin\theta \sin\phi - \sin\psi \cos\phi)$, $b = (\cos\psi \sin\theta \cos\phi + \sin\psi \sin\phi)$, $c = (\sin\psi \sin\theta \sin\phi + \cos\psi \cos\phi)$, $d = (\sin\psi \sin\theta \cos\phi - \cos\psi \sin\phi)$, $e = \cos\theta \sin\phi$, $f = \cos\theta \cos\phi$. Hence, following the affine dynamic system of the form (4.15), the kinematic components of the positions and orientation (Euler angles) in the Earth frame of the dynamic model can be expressed in the following form:

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \underbrace{\begin{bmatrix} v_2 a + v_3 b \\ v_2 c + v_2 d \\ v_2 e + v_3 f \\ \omega_1 + \omega_2 \tan\theta \sin\phi \\ \omega_2 \cos\phi \\ \frac{\omega_2 \sin\phi}{\cos\theta} \end{bmatrix}}_{f(x)} + \underbrace{\begin{bmatrix} \cos\psi \cos\theta & 0 \\ \sin\psi \cos\phi & 0 \\ -\sin\theta & 0 \\ 0 & \tan\theta \cos\phi \\ 0 & -\sin\phi \\ 0 & \frac{\cos\phi}{\cos\theta} \end{bmatrix}}_{g(x)} r. \tag{4.18}$$

with $r = [v_1, \omega_3]^T$ is the controlled linear and angular velocities. Concerning the dynamic components of the linear and angular accelerations in the body frame, we have:

$$\begin{bmatrix} \dot{v}_1 \\ \dot{v}_2 \\ \dot{v}_3 \\ \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{\omega}_3 \end{bmatrix} = \underbrace{\begin{bmatrix} -\omega_2 v_3 + \omega_3 v_2 - g\sin(\theta) \\ -\omega_3 v_1 + \omega_1 v_3 + g\sin(\phi)\cos(\theta) \\ -\omega_1 v_2 + \omega_2 v_1 + g\cos(\phi)\cos(\theta) \\ (c_1\omega_2 + c_2\omega_1)\omega_2 \\ c_5\omega_1\omega_3 - c_6(\omega_1^2 - \omega_3^2) \\ (c_8\omega_1 - c_2\omega_3)\omega_2 \end{bmatrix}}_{f(x)} + \underbrace{\begin{bmatrix} X/m & 0 \\ Y/m & 0 \\ Z/m & 0 \\ 0 & K \\ 0 & M \\ 0 & N \end{bmatrix}}_{g(x)} u, \tag{4.19}$$

with X, Y, Z, K, M, N defined as:

$$\begin{aligned} X &= (\delta_r + \delta_l) - F_x & F_x &= \tfrac{1}{2}\rho v_1^2 C_{Fx} A_x, \\ Y &= -F_y & F_y &= \tfrac{1}{2}\rho v_2^2 C_{Fy} A_y, \\ Z &= F_b - F_z & F_z &= \tfrac{1}{2}\rho v_3^2 C_{Fz} A_z, \\ K &= -M_x - C_k(\phi) F_b & M_x &= \tfrac{1}{2}\rho \omega_1^2 C_{Mx} I_x, \\ M &= -M_y - C_m(\theta) F_b & M_y &= \tfrac{1}{2}\rho \omega_2^2 C_{My} I_y, \\ N &= \tfrac{1}{2}(\delta_r - \delta_l) d - M_z & M_z &= \tfrac{1}{2}\rho \omega_3^2 C_{Mz} I_z. \end{aligned} \tag{4.20}$$

In detail, $F_b = \rho g V_{\text{sub}}(p_z) C_b$ with V_{sub} representing the submerged volume, $C_{(\cdot)}$ are the drag coefficient on the translational and moment on the tree axes and the bouncing drag respectively, while A_x, A_y, A_z are the contact areas with the water, and d is the distance between the two motors.

4.4.2 Problem Statement

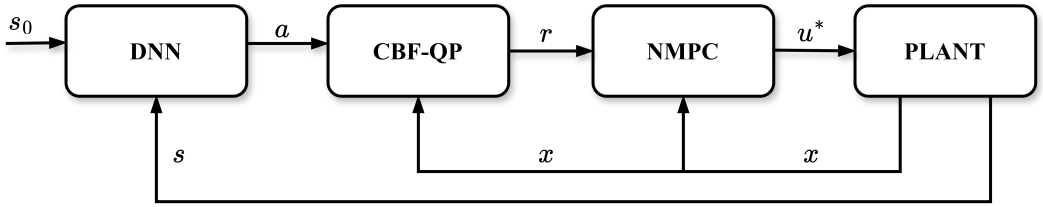


Fig. 4.40: Block diagram of the hierarchical architecture.

To ensure safe and successful deployment of the DRL-based navigation policy, we define a CBF to provide a reference for a low-level controller that corrects the policy action (if necessary) to satisfy the safety constraints (illustrated in Fig. 4.40). To this end, we first investigate how it is possible to synthesize a forward invariant safe set C for a DRL agent that has to navigate within a given environment.

In detail, we consider a neural network f that encodes a DRL policy π and define an input state s as part of the dynamics state vector $\boldsymbol{x}$ (Eq. 4.17). The input space of the policy consists of: (i) 15 sparse beam scans for the boat or LiDAR for the mobile robot, each representing the mean value measurements sampled within a cone in the range $[0, 360]$ degree; (ii) the agent's position $p = [p_x, p_y, p_z]^T$ and orientation $\eta = [\phi, \theta, \psi]^T$ (i.e., odometry values); and (iii) the target's relative position and orientation (i.e., distance and heading). Given a state s, the policy π predicts an action $a = [v_1^{\text{dnn}}, \omega_3^{\text{dnn}}]^T$, representing the agent's linear and angular reference velocities. To ensure that this action respects safety criteria, we aim to keep the agent within safe regions of the state space (in the context of navigation, these safety criteria relate to avoiding collisions with obstacles). Our intuition is to identify these regions and design a safe set C by exploiting the enumeration process. Given the computed safe set, we then formalize a CBF $h(x)$ to enforce safety constraints in the agent's trajectory toward the target. Our approach involves correcting the policy actions by solving a QP problem that incorporates the CBF constraints. This procedure generates reference velocities r, which serve as inputs for an optimal low-level controller based on nonlinear model predictive control (NMPC) [88]. The latter computes the optimal control input u^* for the plant, which evolves using the dynamic (4.19) providing both the new state $\boldsymbol{x}$ for the controller and CBF, and s for the policy.

4.4.3 Methodology

Compute the Forward Invariant Safe Set

To construct the forward-invariant safe set, we identify the regions where the agent does not adhere to safety specifications within a subset of the state space of interest, denoted as $\hat{\mathcal{S}}$. To this end, we rely on ε-`ProVe` presented in Sec.3.5.

This process begins at training time, where we leverage an indicator cost function to detect collisions. This function is the same used in constrained RL literature and deems a state-action pair as unsafe [113]. Hence, the cost allows us to identify portions (or regions, interchangeably) of the state space where the agent is prone to perform unsafe behaviors. We describe this process using the explanatory example in Fig. 4.41.

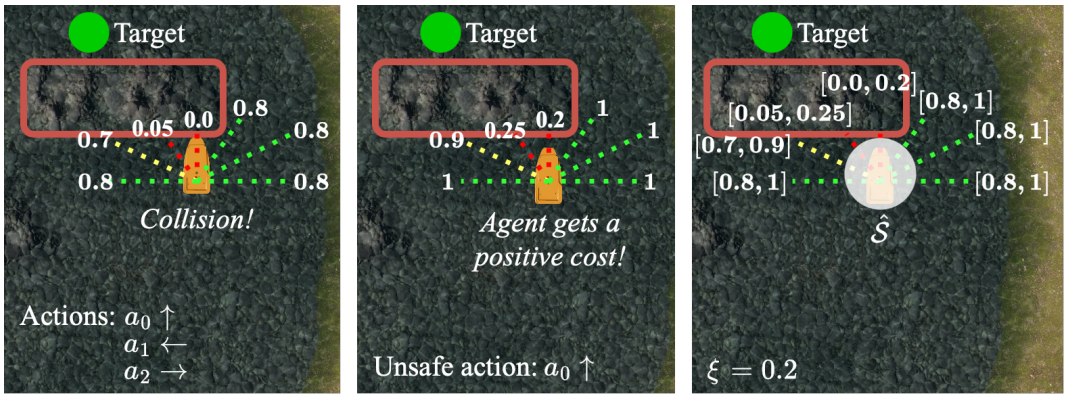


Fig. 4.41: Illustrative example of a safety property in aquatic mapless navigation. The left image shows a collision with an obstacle. In the center, we identify the unsafe state that leads to the collision, while on the right, we define a region of the state space $\hat{\mathcal{S}} \subset \mathcal{S}$ to verify that the agents never choose a forward movement. The property is formally encoded as: $\hat{\mathcal{S}} : s_0 \in [0.8, 1.0], s_1 \in [0.7, 0.9], s_2 \in [0.05, 0.25], s_3 \in [0.0, 0.2], s_{4,5,6} \in [0.8, 1]; \mathcal{Y} : \{a_1, a_2\}$.

Suppose that the aquatic drone agent receives a positive cost value from the environment. This indicates a collision with an obstacle (left figure). The generation procedure selects the state that led to the collision (center figure) and considers an ξ-ball around this state to generate a potentially unsafe area (right figure).[17] In the example, we encode an unsafe state deviation using $\xi = 0.2$ to reflect the dimension and maximum speed of the aquatic drone and an assumed maximum water current velocity. Hence, the selected value ξ is subtracted from the input features obtaining one interval for each input feature as: $\hat{\mathcal{S}} : \{s_0 = [0.8, 1.0], s_1 = [0.7, 0.9], s_2 = [0.05, 0.25], s_3 = [0.0, 0.2], s_{4,5,6} = [0.8, 1]\}$. We assume the states in $\hat{\mathcal{S}}$ are thus potentially risky and model the regions where the agent does not adhere to safety specifications. Therefore, within this area, we probabilistically enumerate the subset of unsafe regions, represented as $\overline{C}_i$. Importantly, by encoding

[17] We use ξ instead of ω, which was used in the previous section, since ω is already reserved for representing the dynamics.

the neighborhoods of unsafe state-action pairs, the approach becomes independent of the specific environment used to train the agent, enabling the identification and mapping of unsafe behaviors to different, potentially unseen scenarios.

Fig. 4.42 then shows an explanatory example on the safe set construction for a region of interest $\hat{\mathcal{S}}_i$.

Specifically, as illustrated on the right side of Fig. 4.42, each safe subset C_i is obtained by subtracting a finite union of verified unsafe regions $\overline{C_i}$ from a compact subset of the state space $\hat{\mathcal{S}}_i$, i.e., $C_i = \hat{\mathcal{S}}_i \setminus \overline{C_i}$. Since removing a finite union of closed sets from a compact set yields another compact set, each C_i remains compact and is thus a valid safe subset for a CBF [7, 148].

Hence, by taking the union of all such safe subsets computed in each verified safety property, we obtain a globally valid safe set C for the CBF. Formally,

Proposition 4.3: *The set $C = \bigcup_i C_i$ is a valid safe set for a control barrier function.*

In the next section, we show that from a safe set C as in Prop. 4.3 there exists a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$ such that $h(x) \geq 0$ for all $x \in C$.

Control Barrier Function and Low-Level Controller

After the computation of the safe set C, we need to define a CBF that keeps the agent in this set. To this end, starting from the dynamic models of the agent (see Sec. 4.4.1), we propose a strategy to compensate for the policy actions and ensure safety. Hence, we design a hierarchical approach that forces the agent to comply with the constraint generated by the verification process.

Control Barrier Function

Given the linear and angular reference velocities produced by the policy, our goal is to modulate them to satisfy the safety constraints. To achieve this, we design a CBF that enforces the position constraint defined by C. We then incorporate this CBF into a QP formulation that adjusts the policy's reference values, ensuring the

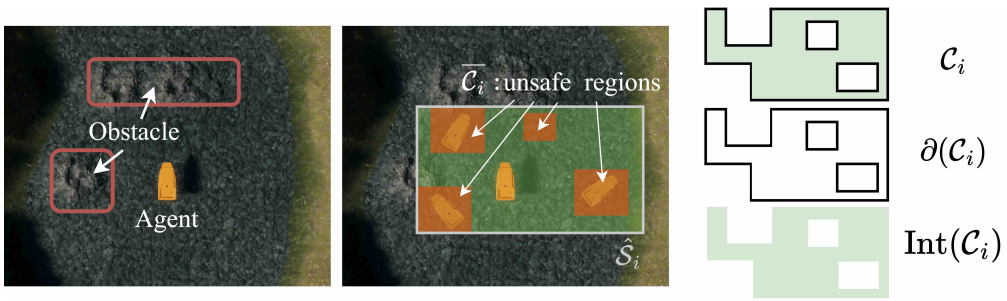


Fig. 4.42: Explanatory example of the relationship between the enumeration process and the CBF. (Left) We report an unsafe situation where the agent has two obstacles on the left and the coastline on the right. (Center) We define the input region to be verified in green, and we enumerate the unsafe regions where the agent has a high probability of colliding. (Right) We report the translation of the enumeration result into the CBF's formulation.

resulting actions keep the agent within the safe set. Given the constraint on the position, we focus only on the kinematic components of the dynamic model defined in equations (4.18). Therefore, we have the safe set computed by the enumeration $C = \bigcup_i C_i = \{x \in \mathbb{R}^n : h(x) \geq 0\}$, with the control barrier function designed as follows:

$$h(x) = ||p - p_{\text{obs}}||^2 - d_{\text{safe}}^2, \tag{4.21}$$

where $p = [p_x, p_y, p_z]^T$ is the position of the agent, $p_{\text{obs}} = [p_x^{\text{obs}}, p_y^{\text{obs}}, p_z^{\text{obs}}]^T$ is the position of the obstacle detected by the sensor and $d_{\text{safe}} = \max(\sigma, ||p - p_{\text{area}}||^2)$ is a safe distance to the unsafe areas. In particular, p_{area} is the centroid of each enumerated unsafe area, and σ is a safe threshold manually defined and based on the sensor precision. Hence, as stated in *Def.* 4.4, the CBF constraint enforces forward invariance of the set C, defining $\dot{h}(x) + \alpha h(x) \geq 0$. By computing the time derivative of $h(x)$ as $\dot{h}(x) = \frac{\partial h}{\partial x}\dot{x}$ we obtain

$$\nabla h(x) f(x) + \nabla h(x) g(x) r + \alpha(h(x)) \geq 0, \tag{4.22}$$

where $f(x)$ and $g(x)$ are the matrices in equation (4.18), and:

$$\nabla h(x) = \left[2(p_x - p_x^{\text{obs}}), 2(p_y - p_y^{\text{obs}}), 2(p_z - p_z^{\text{obs}}), 0, 0, 0\right], \tag{4.23}$$

At this point, we have the constraint on the linear and angular reference velocities, and we can formulate a QP problem to get a modulation action when the policy action does not comply with the constraint. We model the reference variable r as the sum of the policy DNN and CBF as follows:

$$r = \begin{bmatrix} v_1^{\text{dnn}} + v_1^{\text{cbf}} \\ \omega_3^{\text{dnn}} + \omega_3^{\text{cbf}} \end{bmatrix}. \tag{4.24}$$

To compute the contribution of the CBF on the linear and angular reference velocities ($r^{\text{cbf}} = [v_1^{\text{cbf}}, \omega_3^{\text{cbf}}]^T$), we solve an optimization problem formalized as QP in the form:

$$r^{\text{cbf}} = \min_{v_1^{\text{cbf}}, \omega_3^{\text{cbf}}} \quad ||v_1^{\text{cbf}}||^2 + ||\omega_3^{\text{cbf}}||^2,$$

subject to:

$$\nabla h(x) f(x) + \nabla h(x) g(x) \begin{bmatrix} v_1^{\text{dnn}} + v_1^{\text{cbf}} \\ \omega_3^{\text{dnn}} + \omega_3^{\text{cbf}} \end{bmatrix} + \alpha(h(x)) \geq 0.$$

In this way, if the action of the DNN complies with the constraint, the explicit solution to the QP problem does not provide a contribution on v_1^{cbf} and ω_3^{cbf}. Otherwise, the QP problem provides some linear and angular velocities in order to modulate the action of the DNN to comply with the constraints. Importantly, we note that the kinematic model of the Turtlebot3 mobile robot described in Sec. 4.4.1 and the barrier function $h(x)$ defined as (4.21), leads to the reformulated QP problem as:

$$
\begin{aligned}
r^{\text{cbf}} = \min_{v^{\text{cbf}}, \omega^{\text{cbf}}} \quad & ||v^{\text{cbf}}||^2 + ||\omega^{\text{cbf}}||^2, \\
\text{subject to:} \quad & \\
& \begin{bmatrix} 2(p_x - p_x^{\text{obs}}) \\ 2(p_y - p_y^{\text{obs}}) \end{bmatrix}^T \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} (v^{\text{dnn}} + v^{\text{cbf}}) + \\
& \begin{bmatrix} 2(p_x - p_x^{\text{obs}}) \\ 2(p_y - p_y^{\text{obs}}) \end{bmatrix}^T \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix} (\omega^{\text{dnn}} + \omega^{\text{cbf}}) + \alpha(h(x)) \geq 0.
\end{aligned} \tag{4.25}
$$

We report in Alg. 14 the complete pipeline of the proposed approach, considering a TB3 mobile robot with LiDAR scans to sense the environment.

Algorithm 14: Deploy algorithm

1: **Input:** robot, DNN, unsafe_areas, d_{safe}, σ, initial_cond
2: state ← initial_cond.state
3: goal ← initial_cond.goal
4: **while** target_not_reached or collision **do**
5: action_DNN ← DNN(state)
6: action_CBF ← QP+CBF(action, unsafe_areas, state, d_{safe},σ) As in (4.25)
7: action ← action_DNN + action_CBF As in (4.24)
8: state ← robot.low_controller(action)

The loop begins with the robot's initial state and goal, and iteratively computes control actions until the target is reached without collisions. At each step, the DNN predicts an action based on the current state. If either the robot's position p_{robot}, as defined in the state, is closer than the safe distance threshold d_{safe} to an unsafe area, or any LiDAR measurement falls below the threshold σ, a safety margin manually set based on sensor precision, indicating proximity to an obstacle, then a corrective action is computed using the QP formulation with CBF constraints to ensure safety. The final action is obtained by combining the DNN's prediction and the CBF-based correction, as specified in the referenced equation. This composite action is then applied through the robot's low-level controller (e.g., an NMPC) to update its state. The loop ensures goal-directed behavior while enforcing safety constraints in potentially hazardous regions.

Low-level controller

After computing safe reference values, we focus on reaching them in an optimal way by designing an NMPC to manage the control variables δ_l, δ_r. To this end, we focus on the dynamic model defined by equation (4.19). We formalize the NMPC to compute the optimal control u_k^*) over a finite time horizon $[k, k+H]$ at discrete time $t_k = k\Delta t, k \in \mathbb{N}$ as

$$
\begin{aligned}
u_k^* = \min_{u_k,\dots,u_{k+H}} \quad & \sum_{i=0}^{H-1} e_{k+i|k}^T Q e_{k+i|k} + u_{k+i|k}^T R u_{k+i|k}, \\
& \text{subject to:} \\
& x_{k+i+1|k} = f(x_{k+i|k}) + g(x_{k+i|k}) u_{k+i|k}, \\
& u_{k+i|k}, \in \mathcal{U} \qquad u_{\min} \leq u_{k+i|k} \leq u_{\max} \\
& \qquad\qquad\qquad\qquad \forall i \in \{0, \dots, H-1\}, \\
& x_{k|k} = x_k,
\end{aligned}
$$

where x_k are the initial conditions at time k equal to the actual state of the plant x_k and H is the horizon of NMPC. The error is deefined as $e_{k+i|k} = ||x_{k+i|k} - r||$ and the immediate cost function and $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{m \times m}$ are symmetric positive semi-definite matrices. Note that we indicate the predicted evolution of the state with $k+i|k$, where the right k is the actual time, and the left $k+i$ is the time in the prediction horizon.

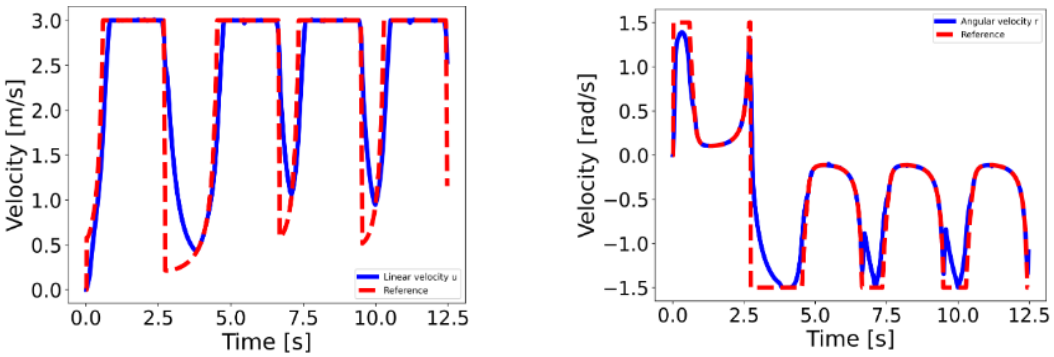


Fig. 4.43: Linear and angular velocities tracking via NMPC. Red dashed lines are the reference r while the blue lines are the optimal control action of the NMPC u^*.

To confirm the correctness of the proposed solution, Fig. 4.43 shows the action of the nonlinear model predictive control employed in our pipeline in order to track the reference (dashed red lines in the plot) provided by the policy modulated via QP problems. Specifically, the figure illustrates the control action on the linear and angular velocities of the boat (blue lines in the plot), indicating that the controller manages and minimizes the tracking error during the evolution. For readability purposes, we only report a short execution of the NMPC with different changing reference values.

4.4.4 Empirical Evaluation

To validate the proposed autonomous safe navigation framework, we conducted a thorough evaluation in simulation and real scenarios. Our goal is to assess, given a DRL-trained policy for a mapless navigation task, whether: (i) our verification-guided CBF layer can consistently correct unsafe actions (i.e., ensure zero collisions) while preserving goal-reaching efficiency; and (ii) empirically investigate if such a layer can unstuck the agents when the policy ends up in local minima. This is particularly relevant since, as we will show in our experiments, a policy trained to satisfy stringent safety constraints typically prefers actions that keep the agent still to satisfy the strict thresholds imposed at training time. For the DRL policies, we consider popular unconstrained, penalty-based, and constrained algorithms that have been widely employed in safe navigation tasks [222, 114, 168] - PPO [232], PPO_penalty, its Lagrangian (safe) version, PPOLag [243], SAC [93] and P3O [301]. In all training configurations employed, episodes are not terminated upon collision; instead, the agent continues its trajectory, and cost signals are accumulated throughout the episode to reflect the ongoing safety performance. In detail, PPO does not receive information regarding collisions, PPO_penalty uses a -0.01 reward penalty, and PPOLag uses a cost threshold set to 0 (aiming to achieve no collisions). As in related safe DRL mapless navigation works and as seen in the previous section, we performed an initial grid search to set the reward penalty, with -0.01 resulting in the best value to incentivize collision-free behaviors. We remark that our focus is on correcting unsafe actions at deployment time. Hence, the specific choice of reward penalty for training a navigation policy does not influence the proposed methodology. We train these policies on an RTX 2070 and an i7-9700k CPU equipped with 48 GB of RAM over 15 random seeds, using existing Omnisafe implementations of the algorithms [114].

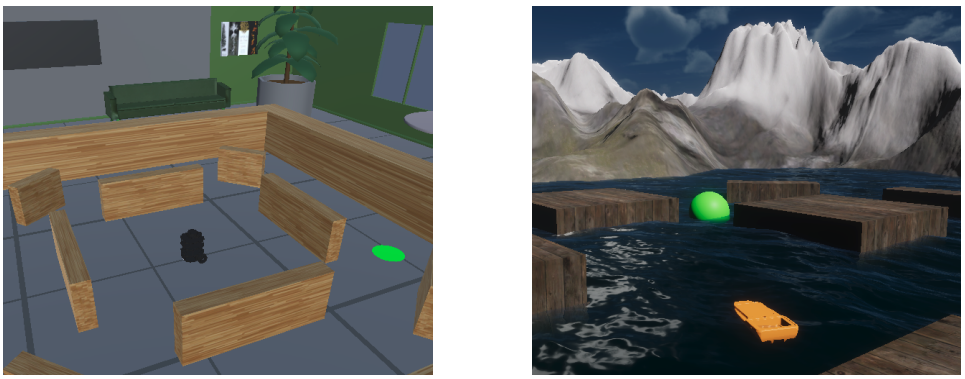


Fig. 4.44: Indoor mobile navigation scenario and aquatic navigation task employed in our simulation empirical evaluation.

For the environments (Fig. 4.44), we create two Unity-based scenarios considering a navigation task in a cluttered indoor environment with static obstacles for a Turtlebot3 mobile robot and a more challenging aquatic environmental monitoring task. The latter is characterized by an autonomous surface vehicle operating in a simulated aquatic environment with drift, sensor noise, and external disturbances.

Simulation Results

Our results are reported in Fig. 4.45. During training, all methods achieve a high success rate, indicating successful navigation behaviors towards random target locations. However, collisions still occur at convergence for all the DRL baselines. The average cost at convergence for the safety-oriented methods—PPOLag, PPO_penalty, and P3O approaches—is significantly closer to zero compared to the unconstrained baseline. Notably, even Lagrangian-based methods fail to fully satisfy the imposed strict cost threshold. The benefits of our CBF layer are clearly highlighted in Tab. 4.9 - all the given (unsafe) DRL policies combined with our method have a higher success rate while achieving safe navigation with 0 collisions. This result confirms both the effectiveness and the potential of our framework in guaranteeing safety while mitigating instances where the agent gets stuck or exhibits near-zero cost during training but fails to be deployed safely. This phenomenon is evident in the aquatic navigation task, where both PPO_penalty and PPOLag achieve the lowest collision rates (at the expense of lower success rates) compared to the unconstrained PPO baseline. Crucially, our method demonstrates strong recovery capabilities in these situations, enabling the agent to progress toward goal locations. These improvements are further confirmed by statistically significant differences ($p < 0.05$), as determined by pairwise Welch's t-tests conducted over 100 runs (in bold in the table).

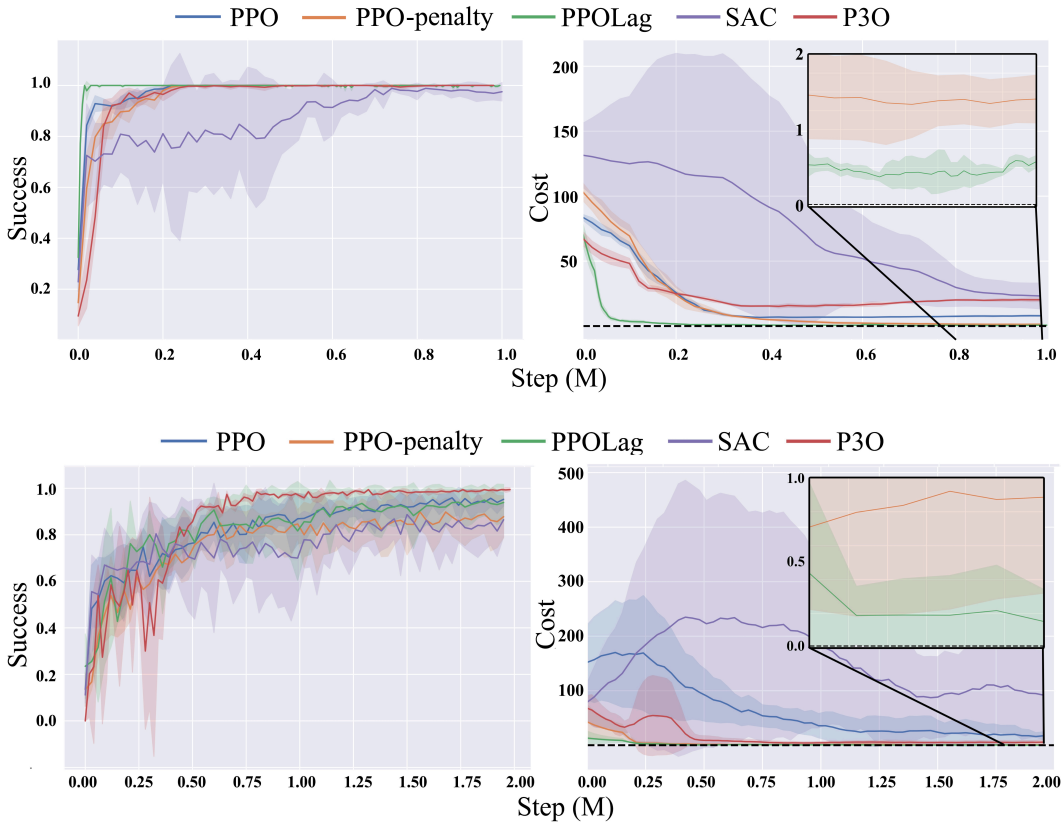


Fig. 4.45: Training results using PPO, PPO_penalty, PPOLag, SAC and P3O on TB3 (top) and Aquatic (bottom) navigation task, respectively.

This behavior is further highlighted in Fig. 4.47, which shows enumerated regions where the agent not only tends to collide but also moves with near-zero linear velocity. Interestingly, some of these red-marked areas lack any visible obstacles, suggesting that the agent gets stuck even in open space due to suboptimal local policy decisions. It is important to remark that our approach does not endow the agent with navigation skills; if the base policy lacks the ability to reach the goal from certain states, our method is not designed to compensate for this. Rather, our approach enhances the base policy by helping the agent recover from stuck situations, and reaching the goal is due only to the policy's capabilities to generalize to unseen scenarios properly.

	TB3 Evaluation		**Aquatic Evaluation**	
Method	**Success (%)**	**Collision (%)**	**Success (%)**	**Collision (%)**
PPO	100±0 %	14.99 ±0.02%	84.2 ± 9.75 %	11.22 ± 3.26 %
PPO+CBF	**100±0 %**	**0±0%**	**87.0 ± 12.72%**	**0.0 ± 0.0** %
PPO_penalty	100±0 %	3.32 ±1.03%	89.7 ± 5.19 %	0.007 ± 0.005 %
PPO_penalty+CBF	**100±0 %**	**0±0%**	**96.7 ± 2.30%**	**0.0 ± 0.0** %
PPOLag	99.66 ± 0.47%	0.47 ± 0.21%	84.0 ± 5.65 %	0.001 ± 0.002 %
PPOLag+CBF	**100±0 %**	**0±0%**	**94.0 ± 4.24%**	**0.0 ± 0.0** %
SAC	91.87 ± 2.63%	8.13 ± 0.49%	88.4 ± 7.83 %	11.6 ± 2.31 %
SAC+CBF	**100±0 %**	**0±0%**	**91.5 ± 3.67%**	**0.0 ± 0.0** %
P3O	99.80 ± 0.21%	0.2 ± 0.03%	91.03 ± 2.13 %	4.17 ± 0.3 %
P3O+CBF	**100±0 %**	**0±0%**	**95.67 ± 3.33%**	**0.0 ± 0.0** %

Table 4.9: Evaluation results of the proposed verification-guided CBF on the trained policies over 100 trajectories.

Real-World Results

On top of the simulation results, Unity allows us to deploy the policies onto ROS2-enabled platforms, such as the Turtlebot3 robot. We thus selected the three best-performing seeds of all the baselines tested at convergence (i.e., the baseline achieving the best trade-off between safety and performance) and evaluated the effectiveness of our CBF-based approach in the real setting depicted in Fig. 4.46. We set $d_{\text{safe}} = 0.15$ and $\sigma = 0.2$ as the safety thresholds (values are normalized between 0 and 1), based on the sensor precision of TB3 and the environment dimension employed in our real experiments. Results reported in Fig. 4.46 confirm that our approach consistently achieves zero constraint violations while maintaining effective navigation performance, even under real-world sensor readings. The complete video of our empirical evaluation is available in the supplementary material.

Importantly, we note a substantial difference between improvements in collision rate and success rate. This phenomenon can be explained by how collisions affect task completion in the real-world setting. In our experiments, a collision typically results in the robot getting stuck or unable to proceed toward the goal, thus leading to both a constraint violation and a failed episode. Therefore, the high collision rate of the baseline policy (e.g., SAC 24.99%) is directly responsible for its lower success rate

Method	Success Rate (%)	Collision Rate (%)
PPO	89.03 ± 3.19 %	10.97 ± 2.1 %
PPO+CBF	**100 ± 0.0%**	**0.0 ± 0.0** %
PPO_penalty	97.08 ± 4.82 %	2.92 ± 3.24 %
PPO_penalty+CBF	**100 ± 0.0%**	**0.0 ± 0.0** %
PPOLag	73.07 ± 5.19 %	16.93* ± 2.1 %
PPOLag+CBF	**98.73 ± 2.5%**	**0.0 ± 0.0** %
SAC	75.01 ± 4.35 %	24.99 ± 4.1 %
SAC+CBF	**100 ± 0.0%**	**0.0 ± 0.0** %
P3O	91.43 ± 6.7 %	8.57 ± 3.5 %
P3O+CBF	**100 ± 0.0%**	**0.0 ± 0.0** %

Fig. 4.46: Mean success and collision rate for the real-world experiments over 10 random start and goal positions with different baselines and the proposed approach. * Low collision is due to a suboptimal stuck policy that does not reach the goal. Bold indicates statistically significant differences ($p < 0.05$) over the 10 runs.

(75.01%). In contrast, our CBF-based approach consistently prevents collisions by design, ensuring that the robot always selects a longer but safer trajectory toward the goal. This safety guarantee translates directly into improved task success, resulting in both zero collisions and an almost perfect success rate (100%) for all the policies tested. The only exception is PPOLagr, which shows a lower success rate due to a policy that gets stuck and fails to reach the goal, but still without colliding. This outcome aligns with our simulation results, confirming that imposing a strict cost threshold can produce safer but suboptimal policies.

Ablation Study

To further support the validity of our approach, we conducted an additional experiment aimed at qualitatively comparing the unsafe regions inferred probabilistically with a ground truth set obtained by simulating the SE(2) dynamics using depth ray (LiDAR) observations. It is important to clarify that this experiment is purely illustrative: in our framework, we assume a mapless navigation setting, and thus do not rely on a priori knowledge of the environment. Although, in principle, one could compute the unsafe set by exhaustively simulating the SE(2) dynamics with LiDAR observations or beam scans to detect the obstacles' position in the environment, this process is highly inefficient in practice and does not accurately reflect the true unsafe regions associated with the DRL policy. Nevertheless, the results reported in Fig.4.47, which visualizes a comparison between the two approaches, show that the probabilistically inferred unsafe regions (highlighted in red) are consistent with those computed via dynamics-based exploration (orange dots). Moreover, due to the

inclusion of the parameter ω, which allows the safety property to be evaluated over a broader portion of the state space, our method can identify regions where the agent is likely to move toward obstacles. This insight is crucial from a safety perspective, as it enables the QP-CBF layer to provide corrective actions before the agent enters states from which unsafe behaviors are more likely. Additionally, as discussed earlier, our verification-based approach can also detect portions of the state space where the agent becomes stuck, enabling the design of ad hoc shielding or recovery strategies.

Additionally, our probabilistic enumeration approach provides a form of explainability for the DRL policy. By collecting and aggregating unsafe regions, we can generate density maps that highlight the areas of the state space where the agent is more prone to exhibit unsafe behavior. These visualizations, reported in the right part of Fig. 4.47, allow for the identification of agnostic start and goal position failure-prone zones and the analysis of the underlying causes of risk. In fact, as highlighted in the image, the clearer regions in the density map are the ones that present more enumerated unsafe regions. Such interpretable representations can provide a way to understand policy weaknesses, potentially guiding further training or enabling the integration of safety monitors that focus on high-risk regions.

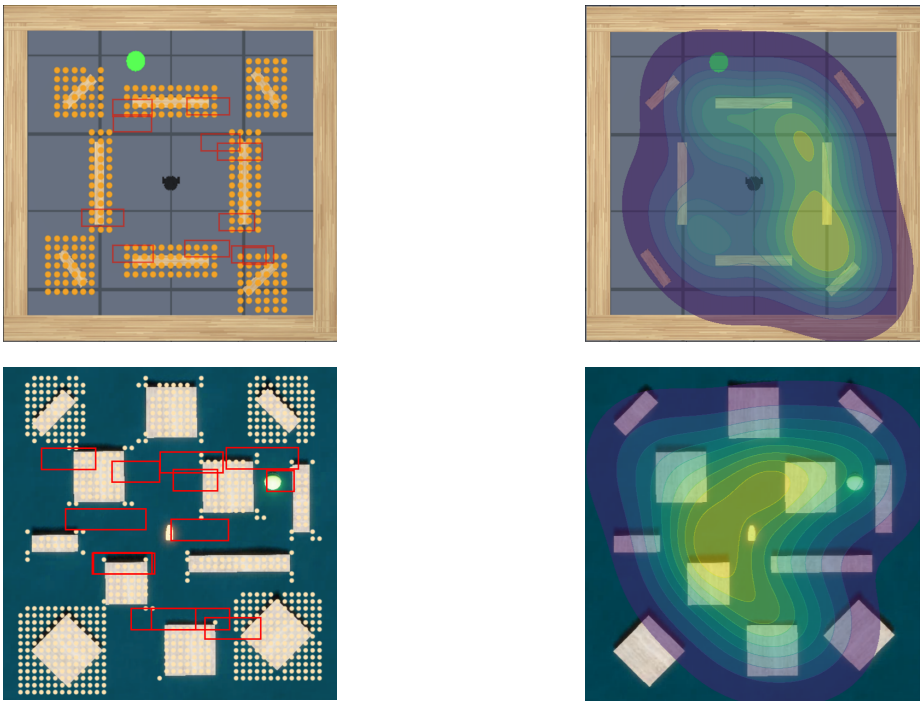


Fig. 4.47: On the left, comparison between unsafe region generated with SE (2) dynamic (in orange) and probabilistic enumeration (in red). On the right is the density map of the unsafe regions in the state space.

Limitations

While our approach demonstrates promising results in enhancing the safety of RL agents during the deployment phase, there are important considerations to be made. First, our approach, as common in the safe reinforcement learning literature [113], requires a simulator to train the agent and assumes access to a cost signal that indicates the safety of a given state-action pair. Second, we assume that the area surrounding an unsafe (e.g., collision) state is also prone to safety violations. While this assumption improves conservative behavior, it may lead to overly cautious policies, especially in systems with highly non-linear dynamics, by labeling larger regions of the state space as unsafe, potentially sacrificing an optimal trajectory towards the goal for safety. Nonetheless, we argue that longer but safer trajectories are preferable in safety-critical applications. Finally, our approach requires solving a convex QP problem at each step, which can be done in polynomial time using an interior-point algorithm. While this introduces additional computational overhead compared to the baseline, the increase is polynomial and, we believe, tolerable in real-world applications given the significant improvements in safety that our method provides.

Summary. In this last section, we proposed a novel framework that integrates probabilistic verification to design a CBF-based layer that ensures safety for given DRL policies trained for autonomous mapless navigation. By identifying unsafe regions in the state space through probabilistic enumeration and enforcing safety constraints via CBF-based control, our approach provides a principled and scalable method to correct unsafe actions while preserving and potentially improving the performance of the learned policies. Unlike traditional methods that rely heavily on domain-specific knowledge or incur high computational costs, our framework is agnostic to the deployment environment and operates independently of the DRL training process, enhancing its general applicability. Through extensive evaluations, we demonstrated that our approach achieves robust safety guarantees with zero violations of safety constraints. At the same time, it preserves effective navigation behaviors across different robot platforms, including an indoor robot ground and an aquatic drone.

Chapter 5

Towards Explainable AI via Verification Techniques

"I learned very early the difference between knowing the name of something and knowing something."

– Richard P. Feynman

Chapter Contributions[1]

In this chapter, we address **RQ.3**, studying the problem of generating robust counterfactual explanations for deep learning models subject to model changes. We focus on *plausible model changes*, altering model parameters, and propose a novel framework to reason about the robustness property in this setting. To motivate our solution, we begin by showing for the first time that computing the robustness of counterfactuals with respect to model changes is NP-hard. As this (practically) rules out the existence of scalable algorithms for exactly computing robustness, we propose a novel probabilistic approach that is able to provide tight estimates of robustness with strong guarantees while preserving scalability. Remarkably, and different from existing solutions targeting plausible model changes, our approach explores the use of a verification method for explainability purposes and does not impose requirements on the network to be analyzed, thus enabling robustness analysis on a wider range of architectures, including state-of-the-art tabular transformers. Throughout the sections, we provide experimental analysis on four binary classification datasets, revealing that our verification-based method improves the state of the art in generating robust explanations, outperforming existing methods. We conclude the chapter by presenting a novel Python library implementing a collection of generation and evaluation methods for CEs, with a focus on the robustness property. It provides easy-to-use interfaces for implementing new CE methods and benchmarking performances against a range of robust methods.

[1] The content of this chapter is based on the following articles:
[C.17] Marzari L., et al. (2024). *"Rigorous Probabilistic Guarantees for Robust Counterfactual Explanations"*, European Conference on Artificial Intelligence (ECAI).
[C.18] Marzari L., et al. (2025). *"Probabilistically Robust Counterfactual Explanations under Model Changes"*, Artificial Intelligence Journal (AIJ).
[C.19] Jiang J.*, Marzari L.*, Purohit A., and Leofante F. (2025). *"RobustX: a Python Framework to Benchmark the Robustness of Counterfactual Explanations"*, International Joint Conference on Artificial Intelligence (IJCAI).

5.1 Probabilistically Robust Counterfactual Explanations under Model Changes

Deep Neural Networks have emerged as a groundbreaking technology revolutionizing several fields ranging from autonomous navigation [249, 172] to image classification [203] and robotics for medical applications [52]. However, despite remarkable successes, their vulnerability to adversarial attacks [246, 8], i.e., imperceptible modifications to input data that can lead to wrong and potentially catastrophic decisions when deployed, has raised crucial safety concerns. Consequently, understanding and explaining the decisions of black-box deep learning models has become a dominant goal in AI research. In this section, we focus on counterfactual explanations (CE), a popular class of explanation methods that aim to demystify the decision-making of a DNN by showing how an input needs to be changed to yield a different, typically more desirable, decision (see [242, 129] for recent surveys).

To understand what makes CEs useful, consider the widely used example of a loan application, where a mortgage applicant represented by an input x with features *unemployed* status, 25 years of age, and *low* credit rating applies for a loan and is rejected by the bank's AI. A CE for this decision could be a slightly modified input, where increasing credit rating to *medium* would result in the loan being granted. Ideally, a counterfactual explanation should be as close as possible to the original input to ensure that the changes it suggests are feasible. The approach of [272], showed how this requirement can be mathematically achieved by generating a counterfactual as close as possible to the decision boundary of a DNN. However, producing explanations in this way raises critical concerns about their reliability (see [118] for a survey). For example, as illustrated in Fig. 5.1, fine-tuning the model with additional data can significantly alter its decision boundary, potentially invalidating previously generated counterfactuals. This sensitivity to the model changes for the counterfactuals poses critical questions about the reliability of explanations and long-term usability in dynamic settings.

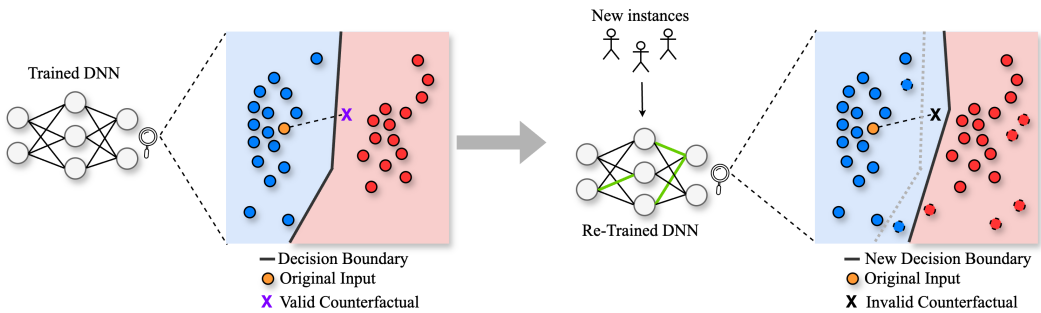


Fig. 5.1: Vignette illustrating the problem of robustness under model changes. A counterfactual explanation is initially generated for a trained model (left). Then, the model is updated to include new data (right). This step might induce slight changes in the decision boundary of the model, ultimately invalidating the counterfactual explanation generated in the first step.

While recomputing counterfactuals from scratch after every model update would be possible in principle, doing so might have some undesirable consequences. First of all, in domains such as credit scoring and medical decision-making, previously issued explanations form part of an auditable record. As such, arbitrarily discarding previously issued CEs may raise compliance concerns. Moreover, CEs are often advocated as tools to provide ground for recourse recommendations. This clearly introduces compliance requirements, as previously stated, but also cognitive ones: users would reasonably expect a certain degree of consistency in the behavior of the explainer. If counterfactual recommendations fluctuate arbitrarily across retrainings, user trust in the system may decrease dramatically. Finally, as we will show in this section, many CE generation procedures are iterative optimization processes, which are known to be computationally expensive. Hence, for organizations issuing thousands of explanations, re-computation after every minor update would be undesirable. Hence, recent work has highlighted issues related to the robustness of CEs against *Plausible Model Changes* (PMC) [262, 116], showing that the validity of CEs is likely to be compromised when bounded perturbations are applied to the parameters of a DNN, e.g., as a result of fine-tuning [262, 27, 200, 116, 94]. Consider the loan example: if retraining occurs while the applicant is working toward improving their credit rating, without robustness, their modified case may still result in a rejected application, leaving the bank liable due to their conflicting statements.

In this section, we focus on this troubling phenomenon and advance the state of the art in CE robustness research in several directions. More specifically, we start by studying the computational complexity of exactly determining whether a CE is robust to PMC in SubSec. 5.1.1. Our result formally shows for the first time that this is an NP-hard problem in the size of the neural network model, i.e., the number of parameters and layers defining the classifier, thus providing new insights into algorithmic developments in this area.

As our hardness results rule out the existence of practical algorithms to compute the CE robustness in an exact fashion, we argue that probabilistic approaches are needed to obtain answers on the CE robustness under model changes. Notably, the work by [94], proposes a probabilistic approach to compute the robustness of CE under *Naturally-Occurring Model Changes* (NOMC).[2] Even though both PMC and NOMC notions are commonly used in the literature, very little is known about their potential interplay, and whether robustness to NOMC subsumes robustness to PMC is still unresolved. In SubSec. 5.1.2, we report a complete study of the two notions and formally prove that these two notions capture profoundly different scenarios. As a result, we demonstrate that robustness guarantees given for NOMC do not directly extend to PMC. Having settled this, in SubSec. 5.1.3, we present an extended overview of our APΔS , a novel sampling-based certification algorithm that allows us to determine a provable probabilistic bound on the maximum shift a CE can tolerate under PMC. Unlike existing solutions for robustness under PMC, our approach comes with significantly reduced computational requirements and does

[2] In this work, we primarily use the term “model changes", following the notation used in recent surveys on the topic [118]. An alternative term “model shifts", with similar meaning, has also been used in related literature, as in [177]. The two terms will be used interchangeably throughout.

not make any assumption on the underlying DNN, thus making it applicable to a wider range of architectures, including state-of-the-art transformer architectures.

To confirm this aspect, in SubSec. 5.1.4, we present a thorough experimental evaluation analysing the performance of APΔS , providing a comprehensive comparison of the proposed approach against several state-of-the-art methodologies for CE robustness and different ablation studies. Crucially, we show that our approach outperforms existing methods on several metrics from the CE literature, including validity, proximity, and plausibility.

This section is structured as follows. SubSec. 5.1.1 presents our complexity analysis and offers complete proof of NP-hardness for both PMC and NOMC. Motivated by this result, in SubSec. 5.1.2, we study existing approaches to generate probabilistically robust CEs and analyze their interplay. Then, in SubSec. 5.1.3, we introduce our method to generate robust CEs, APΔS , and evaluate it extensively in SubSec. 5.1.4. The core contributions of this section can be summarized as follows:

- We prove, for the first time, that determining whether a CE is robust to model changes in a deep neural network is an NP-complete problem, for both existing notions of NOMC and PMC. This finding highlights the need for further research into probabilistic methods to address this problem effectively.
- We analyse existing approaches to generate probabilistic guarantees for CEs under NOMC and demonstrate that these guarantees do not extend to PMC.
- We present APΔS , a scalable procedure that is able to generate provably robust CEs. This approach introduces an iterative algorithm to generate probabilistically robust CEs, which are demonstrated to have superior performance against four robust baselines.
- To confirm the scalability and effectiveness of our solution, we employ APΔS to certify the robustness of CEs for state-of-the-art transformer architectures [11] employed in tabular data classification. To the best of our knowledge, we are the first to consider models of this size within the robust CE literature [118].

This section builds upon our previous work Marzari et al. [177] with significant extensions. Specifically, SubSec. 5.1.1 non-trivially extends the corresponding section in [177] and offers a full hardness proof for the problem of deciding robustness of counterfactual explanations under PMC. As a corollary of this result, we are also able to show the hardness with respect to NOMC, thus providing a rigorous characterization of the complexity of verifying CE robustness under existing notions of model changes. SubSec. 5.1.2 is also extended with a thorough experimental evaluation, complementing our theoretical findings of [177] and showing that PMC and NOMC capture very different robustness requirements in practice. Our experimental analysis in SubSec. 5.1.4 is also extended considerably. In particular, we present a novel analysis of the impact that the main hyper-parameters of APΔS can have on the quality of CEs it generates. Moreover, we demonstrate the scalability of our approach by presenting new results obtained on large-scale tabular transformers. To the best of our knowledge, this is the first time a method for robust CEs has been shown to scale to state-of-the-art transformer models. These results complement our previous analysis and demonstrate the versatility of APΔS , as well as its effectiveness in solving robustness issues in state-of-the-art machine learning models.

5.1.1 Checking Robustness to Model Changes is Hard

In this section, we study the computational complexity of deciding whether a given counterfactual explanation is robust in the presence of model shifts. Our aim here is to better understand the computational challenges arising from this problem and to use these results to guide the development of novel, more efficient certification procedures. Without loss of generality, we first focus on PMC and show the NP-hardness of verifying CE robustness with respect to this definition of model changes. Later, we show that the set of PMC used by our reduction also constitutes a set of NOMC, which implies that CE robustness is, in general, also hard to verify with respect to NOMC.

Deciding whether for a given $\mathcal{M}_\theta$ a CE x' is robust with respect to a set of PMC Δ requires to check if, for at least one model shift in Δ, there exists a realization $\mathcal{M}_{\theta'}$ which classifies CE x' differently from $\mathcal{M}_\theta$, i.e., $\mathcal{M}_{\theta'}(x) < 0.5 \leq \mathcal{M}_\theta$. This question is encoded in the following problem.

> Distinct-realizations Problem (DRP)
>
> **Input**: an instantiation $\mathcal{M}_{\theta_1}$ of a parametric classifier $\mathcal{M}_\Theta$, an input x such that $\mathcal{M}_{\theta_1}(x) \geq 0.5$, and a set Δ of PMC.
>
> **Output**: yes $\iff$ there exists an instantiation $\mathcal{M}_{\theta_2}$ of $\mathcal{M}_\Theta$ which is a realization of Δ and such that $\mathcal{M}_{\theta_2}(x) < 0.5$

Notably, in our analysis, we assume that the intervals parameter encoding the model change are discretized at the maximum resolution representable in machine precision, so that the set of model realizations is finite and the corresponding decision problem is well defined. To prove the hardness of the problem, we show a reduction from a prototypical NP-complete problem as typically done when studying the complexity of reasoning about neural networks (see, e.g., [132, 28, 153]). In detail, we consider a simple variant of 3-SAT, which we refer to as 3-NAF-SAT.

> 3-NotAllFalse-SAT (3-NAF-SAT)
>
> **Input**: a 3-CNF ϕ such that the assignment of all false values is not satisfying, i.e., $\phi(false, false, \ldots, false) = false$.
>
> **Output**: yes $\iff$ there exists an assignment $\mathbf{a}$ such that $\phi(\mathbf{a}) = true$.

The NP-completeness of 3-NAF-SAT immediately follows from the NP-completeness of 3-SAT. We provide a proof of this fact for the sake of self-containment of the section.

Theorem 5.1: 3-NAF-SAT *is NP-complete.*

Proof. We show a reduction from 3-SAT. Let ψ be a 3-CNF formula over n variables $x_1, \ldots, x_n$. Consider the 3-CNF formula ϕ over $n+1$ variables defined by $\phi(x_1, \ldots, x_n, x_{n+1}) = \psi(x_1, \ldots, x_n) \wedge (x_{n+1} \vee x_{n+1} \vee x_{n+1})$. Clearly for the assignment $\mathbf{a}$ such that $a_i = false$ for each $i = 1, \ldots, n+1$ we have $\phi(\mathbf{a}) = false$, hence ϕ is a proper instance of 3-NAF-SAT, which is obtainable in polynomial time from the

instance ψ of 3-SAT. Moreover, $\mathbf{a} = (a_1, \ldots, a_n, a_{n+1})$ is a satisfying assignment for ϕ if and only if $a_{n+1} = true$ and $\psi(a_1, \ldots, a_n) = true$, i.e., if and only if $a_1, \ldots, a_n$ is satisfying for ψ. □

Theorem 5.2: *Deciding* DRP *is NP-complete.*

Proof. The inclusion of DRP in NP is trivial. A certificate is a $\mathcal{M}_{\theta_2}$ which is of the same size as $\mathcal{M}_{\theta_1}$, hence polynomial in the input size. The verification of such a certificate, consists of a forward propagations of x through $\mathcal{M}_{\theta_2}$ in order to check that $\mathcal{M}_{\theta_2}(x) < 0.5$. This is clearly doable in time polynomial in the size of the classifier, i.e., polynomial in the input.

For the hardness of DRP we show a reduction from 3-NAF-SAT. In particular, we show that there is a $\delta \in (0, 1]$ such that, given a 3-CNF formula ϕ, not satisfied by the all-false assignment, we can construct an INN $\mathcal{I}$ whose edge intervals are all of the width 2δ and an input x such that

1. for $\mathcal{M}_{\theta_1}$ being the DNN with the same topology of $\mathcal{I}$ and such that for each edge e the weight w_e is taken as the central point of the interval assigned to e in $\mathcal{I}$, we have $\mathcal{M}_{\theta_1}(x) \geq 0.5$;
2. ϕ is satisfiable if and only if there exists another DNN $\mathcal{M}_{\theta_2}$ which is also a realization of $\mathcal{I}$ and such that $\mathcal{M}_{\theta_2}(x) < 0.5$.

Note that we are using the INN $\mathcal{I}$ to represent both the parametric classifier $\mathcal{M}_\Theta$ and the set of PMC Δ, consisting of all the possible DNN being a realization of $\mathcal{I}$.

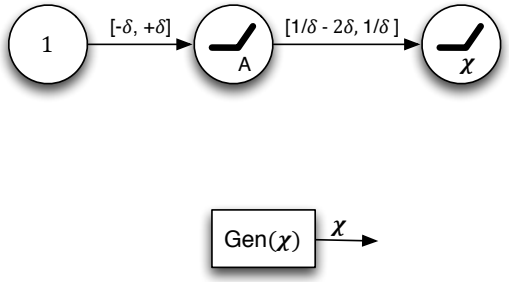


Fig. 5.2: Generating-gadget. The input to this gadget is the constant 1 represented by the leftmost node. The output is the value χ computed in the rightmost node, that depends on the weights chosen in the intervals on the two edges.

We start by analysing several gadgets that will be used as building blocks of $\mathcal{I}$. These gadgets are shown in Fig.s 5.2, 5.3, 5.4, 5.5.

Lemmas 5.1-5.6 provide the key properties of such gadgets which will be used in the reduction. The parameter δ is a number in $(0, 1)$ whose value will be fixed by the analysis.

Lemma 5.1 (Generating-gadget): *The value χ computed in the leftmost node of the Generating-gadget in Fig. 5.2, satisfies $\chi \in [0, 1]$.*

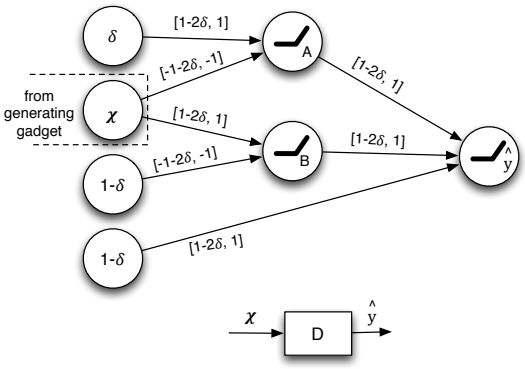


Fig. 5.3: Discretizer-gadget. The only non-constant input is the value computed in the node labelled χ. The output is the value computed in the node labelled $\hat{y}$.

Proof. The value computed by the first node satisfies $A \in [0, \delta]$. Hence, since $\chi = \max\{0, A \cdot w\}$ with $w \in [\frac{1}{\delta} - 2\delta, \frac{1}{\delta}]$ we have $\chi \in [0, 1]$. □

Lemma 5.2 (Discretizer-gadget): *Consider the* Discretizer-*gadget in figure 5.3 with χ being the the leftmost node of a Generating-gadget, i.e., the corresponding value satisfies $\chi \in [0, 1]$. Then, for the value $\hat{y}$ the following holds:*

1. *if $\hat{y} > 1 - \delta$ then $\chi \in [0, \delta] \cup [1 - \delta, 1]$;*
2. *if $\chi \in \{0, 1\}$ then there are possible choices of the weights yielding $\hat{y} = 1$;*
3. *if $\hat{y} \neq 1$ then $\chi \notin \{0, 1\}$.*
4. *$\hat{y} \in [0, 1]$.*

Proof. The claims are a direct consequence of the following observations (refer to Fig. 5.3 for the notation):

(a) if $\chi \in (\delta, 1 - \delta)$ the $A = 0$ and $B = 0$, hence $\hat{y} \in [(1 - \delta)(1 - 2\delta), 1 - \delta]$.

(b) if $0 < \chi \leq \delta$ then $B = 0$ and $A \in [\max\{0, \delta(1-2\delta) - \delta(1+2\delta)\}, \delta - \chi] \subseteq [0, \delta)$. Hence

$$\hat{y} \in [(1 - \delta)(1 - 2\delta), (1 - \delta) + (\delta - \chi)] \subseteq [(1 - \delta)(1 - 2\delta), 1).$$

(c) if $(1 - \delta) \leq \chi < 1$ then $A = 0$ and $B \in [\max\{0, (1 - \delta)(1 - 2\delta) - (1 - \delta)(1 + 2\delta)\}, \chi - (1 - \delta)] \subseteq [0, \delta)$. Hence,

$$\hat{y} \in [(1 - \delta)(1 - 2\delta), (1 - \delta) + \chi - (1 - \delta)] \subseteq [(1 - \delta)(1 - 2\delta), 1).$$

(d) if $\chi = 0$ then $B = 0$ and $A \in [\delta(1 - 2\delta), \delta]$, hence

$$\hat{y} \in [\delta(1 - 2\delta)^2 + (1 - \delta)(1 - 2\delta), 1].$$

In particular, for the realizations of $\mathcal{I}$ where the weights on the topmost edges and on the bottommost edge are chosen to be 1 we have $\hat{y} = 1$.

(e) if $\chi = 1$ then $A = 0$ and $B \in [\max\{0, (1-2\delta) - (1+2\delta)(1-\delta), \delta] = [0, \delta]$, hence

$$\hat{y} \in [(1-\delta)(1-2\delta), (1-\delta)+\delta] = [(1-\delta)(1-2\delta), 1].$$

In particular, for the realizations of $\mathcal{I}$ where all the weights on the edges are chosen to be the maximum possible value, we have $\hat{y} = 1$.

Item 1 in the statement follows directly from (a). Item 2 in the statement follows from (d) and (e). Item 3 in the statement follows from (b) and (c). Finally, Item 4 follows from the (a)-(e). □

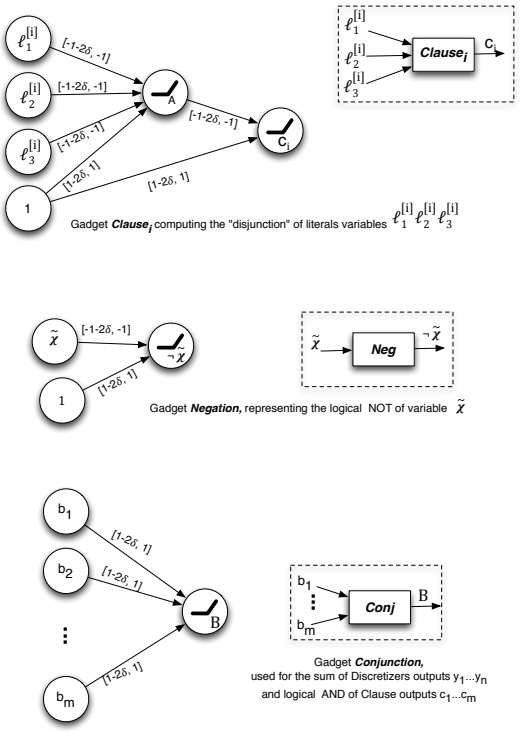


Fig. 5.4: LOGICALPORTS

Lemma 5.3 (Negation-gadget): *With reference to the* Negation-*gadget in Fig.5.4, for any* $0 < \delta < \frac{\sqrt{6}}{2} - 1$, *and* $\chi \in [0, \delta] \cup [1-\delta, 1]$, *the following holds:*

1. $\neg\chi \in [0, \delta] \iff \chi \in [1-\delta, 1]$;
2. $\neg\chi \in [1 - 3\delta - 2\delta^2, 1] \iff \chi \in [0, \delta]$.
3. *if* $\chi \in \{0, 1\}$ *then there is a choice of the weights of the Negation-gadget such that* $\neg\chi = 1 - \chi$. *In other words, if the input value is binary, then there is a choice of the weights such that the Negation-gadget computes the boolean NOT of the input* χ.

Proof. We have $\neg\chi = \max\{0, w_2 - |w_1|\chi\}$ where $w_1 \in [-1-2\delta]$ and $w_2 \in [1-2\delta, 1]$. Therefore

(i) if $\chi \in [0, \delta]$ it follows that

$$\neg\chi \in [(1 - 2\delta) - \delta(1 + 2\delta), 1 - 0] = [1 - 3\delta - 2\delta^2, 1];$$

(ii) if $\chi \in [1 - \delta, 1]$ then

$$\neg\chi \in [\max\{(1 - 2\delta) - (1 + 2\delta), 0\}, 1 - (1 - \delta)] = [0, \delta].$$

Moreover, because of the hypothesis $0 < \delta < \frac{\sqrt{6}}{2} - 1$, we have that $1 - 3\delta - 2\delta^2 > \delta$ and

$$[1 - 3\delta - 2\delta^2, 1] \cap [0, \delta] = \emptyset,$$

which implies that the implications hold also in the opposite direction.

For the third claim of the lemma, it is enough to consider the weight in the associated interval for each edge whose absolute value is equal to 1. □

Lemma 5.4 (Clause-gadget): *For the* Clause*-gadget in Fig.5.4, the following holds: Let* $0 < \delta \leq \frac{1}{12}$*, and for each* $t = 1, 2, 3$*, let* $\ell_t^{[i]} \in [0, \delta] \cup [1 - 3\delta - 2\delta^2, 1]$.[3] *We have that,*

1. $c_i \in [0, 5\delta + 6\delta^2]$ *if and only if for all* $t = 1, 2, 3$, $\ell_t^{[i]} \in [0, \delta]$*;*
2. $c_i \in [1 - 5\delta - 8\delta^2 - 4\delta^3, 1]$ *if and only if there is* $t \in \{1, 2, 3\}$ *such that* $\ell_t^{[i]} \in [1 - 3\delta - 2\delta^2, 1]$.
3. *if for each* $t = 1, 2, 3$, $\ell_t^{[i]} \in \{0, 1\}$ *then there is a choice of the weights of the Clause-gadget such that* $c_i \in \{0, 1\}$ *and* $c_i = 0$ *if and only if* $\ell_1^{[i]} = \ell_2^{[i]} = \ell_3^{[i]} = 0$. *In other words, if the input values are binary, then there is a choice of the weights such that the Clause-gadget computes the boolean OR of the inputs* $\ell_t^{[i]}$.

Proof. Consider a realization of the Clause-gadget. Let us denote by w_t^L the weight taken from the interval on the edge connecting $\ell_t^{[i]}$ to A. Moreover, let w_1 denote the weight taken from the interval on the edge connecting the fixed value node 1 to A. Let w_2 be the weight taken from the interval on the edge connecting the fixed value node 1 to the output node of the gadget. Finally, let w_A be the weight taken from the interval associated with the edge connecting the node A to the output node of the gadget.

(i) If for all $t = 1, 2, 3$, it holds that $\ell_t^{[i]} \in [0, \delta]$ then, using $A = \max\{0, w_1 - \sum_{t=1}^{3} |w_t^L| \ell_t^{[i]}\}$, we have that $A \in [\max\{0, 1 - 2\delta - 3\delta(1 + 2\delta)\}, 1] \subseteq [1 - 5\delta - 6\delta^2, 1]$. Since $c_i = \max\{0, w_2 - |w_A| \cdot A\}$, we have

$$c_i \in [\max\{0, (1 - 2\delta - (1 + 2\delta)\}, 1 - (1 - 5\delta - 6\delta^2)] \subseteq [0, 5\delta - 6\delta^2].$$

This shows the sufficiency of the condition in the first item of the statement.

(ii) Assume there exists $\hat{t} \in \{1, 2, 3\}$ such that $\ell_{\hat{t}}^{[i]} \in [1 - 3\delta - 2\delta^2, 1]$. Then

$$A = \max\{0, w_1 - |w_{\hat{t}}^L| \ell_t^{[i]} + \sum_{t \neq \hat{t}} |w_t^L| \ell_t^{[i]}\}.$$

[3] Note that this corresponds to the case when $\ell_t^{[i]}$ is either the output $\neg\chi$ of a Negation-gadget or the output χ of a Generating-gadget such that $\hat{y} > 1 - \delta$

It follows that

$$A \geq \max\{0, (1-2\delta) - (1+2\delta)\cdot 1 - 2(1+2\delta)\cdot 1\} = 0,$$

and

$$A \leq \max\{0, 1-1\cdot(1-3\delta-2\delta^2)-2\cdot(1)\cdot(0)\} = \max\{0, 1-(1-3\delta-2\delta^2)\} = 3\delta+2\delta^2.$$

Hence $A \in [0, 3\delta + 2\delta^2]$. Therefore,

$$c_i \in [\max\{0, (1-2\delta)-(1+2\delta)(3\delta+2\delta^2)\}, \max\{0, 1-(1)\cdot 0\}] = [1-5\delta-8\delta^2-4\delta^3, 1].$$

This shows the sufficiency of the condition in the second item of the statement. Finally, because of the assumption $\delta \leq \frac{1}{12}$ we have that $5\delta - 6\delta^2 < 1 - 5\delta - 8\delta^2 - 4\delta^3$ hence

$$[0, 5\delta - 6\delta^2] \cap [1 - 5\delta - 8\delta^2 - 4\delta^3, 1] = \emptyset,$$

which implies that the conditions in both items of the statement are also necessary.

For the third claim of the lemma, it is enough to consider the weight in the associated interval for each edge whose absolute value is equal to 1. □

Lemma 5.5 (Conjunction-gadget): *Consider the* Conjunction-*gadget in Fig.5.4.*

Let $\tilde{n}$ denote the output of a Conjunction-gadget whose inputs are the values $\hat{y}_1, \ldots, \hat{y}_n$ output by the n Discretizer-gadgets, with input $\chi_1, \ldots, \chi_n$, respectively, such that $\chi_j \in [0, 1]$.

Let $\tilde{c}$ denote the output of a Conjunction-gadget whose inputs are values $c_1, \ldots, c_m$. Assume also that for each $i = 1, \ldots, m$, c_i is the output of a Clause-gadget whose inputs are either the output χ_j of a Generating-gadget or the output of a Negation-gadget whose input is the output of a Generating-gadget.

1. *If $\tilde{n} > n - \delta$ then for each $i = 1, \ldots, m$ it holds that $\hat{y}_i \in (1-\delta, 1]$, and $\chi_i \in [0, \delta] \cup [1-\delta, 1]$.*
2. *If $\tilde{c} > m - (5\delta + 8\delta^2 + 4\delta^3)$ then for each $i = 1, \ldots, m$ it holds that $c_i \in (1 - (5\delta + 8\delta^2 + 4\delta^3), 1]$.*

Proof. The first claim follows from Lemma 5.2. In particular, by Lemma 5.2, the condition on the values χ_j implies $\hat{y}_i \in [0, 1]$. Moreover, by $\tilde{n} > n - \delta$, it follows that for each i we have $\hat{y}_i > 1 - \delta$. Again, by Lemma 5.2, this implies that $\chi_i \in [0, \delta] \cup [1-\delta, 1]$.

For the second claim, we first observe that the hypotheses on the Clause-gadget whose outputs are the values $c_1, \ldots c_m$, imply that the input to such gadgets satisfies the hypotheses of Lemma 5.4. Therefore, for each $i = 1, \ldots, m$, it holds that $c_i \in [0, 5\delta + 6\delta^2] \cup [1 - 5\delta - 8\delta^2 - 4\delta^3, 1]$. It follows that, if $\tilde{c} > m - 5\delta - 8\delta^2 - 4\delta^3$ for each $i = 1, \ldots, m$, it holds that $c_i > 1 - 5\delta - 8\delta^2 - 4\delta^3$. □

Lemma 5.6 (End-gadget): *Consider the* End-*gadget in Fig. 5.5). For any choice of edge weights, it holds that if $z < 1/2$ then*

- *$\tilde{n} > n - \delta$;*
- *$\tilde{c} > m - (5\delta + 8\delta^2 + 4\delta^3)$.*

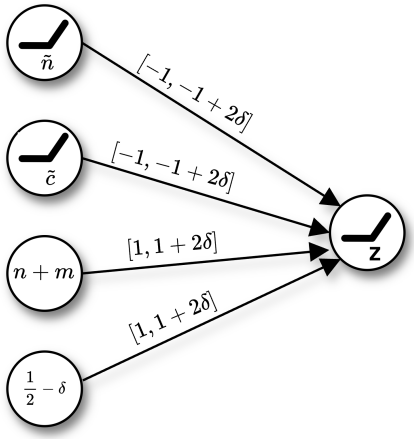


Fig. 5.5: End-gadget

Proof. We have that

$$z \geq \max\{0, n+m+\frac{1}{2}-\delta-\tilde{n}-\tilde{c}\}.$$

We show that if one of the inequalities in the statement is violated, then $z \geq 1/2$.

Suppose that $\tilde{n} \leq n-\delta$. Then, since $\tilde{c} \leq m$, it follows that $z \geq n+m+1/2-\delta-n+\delta-m = 1/2$.

Suppose now that $\tilde{c} \leq m-(5\delta+8\delta^2+4\delta^3) \leq m-\delta = m-\delta$. Then, since $\tilde{n} \leq n$, it follows that $z \geq n+m+1/2-\delta-n-m+\delta = 1/2$. □

The reduction $\mathcal{R} : \phi \mapsto I^\phi = (\mathcal{M}^\phi_{\theta_1}, x^\phi, \Delta^\phi)$. Fix a 3-CNF $\phi(x_1, \ldots, x_n)$, such that $\phi(\mathbf{a}) = false$, for the assignment $\mathbf{a} = (false, \ldots, false)$. Fix a positive rational number $\delta \leq \frac{1}{12}$. Consider the INN $\mathcal{I} = \mathcal{I}^\phi$ built as follows (refer to Fig. 5.6 for an example of this construction):

1. For each variable x_i add to the network a copy of the *Generating*-gadget (and refer to it as $Gen(\chi_i)$) and a copy of the *Discretizer*-gadget (and refer to it as $Disc_i$).
2. For each $i = 1, \ldots, n$, connect $Gen(\chi_i)$ to $Disc_i$ by identifying the output node χ_i of $Gen(\chi_i)$ with the non-constant input node χ of $Disc_i$. Refer to the output node/value of $Disc_i$ as $\hat{y}_i$ (see also Fig. 5.3).
3. For each clause $C_j = (\lambda^{(j)}_1 \vee \lambda^{(j)}_2 \vee \lambda^{(j)}_3)$ $(j = 1, \ldots, m)$ of ϕ add a *Clause*-gadget, henceforth referred to as $Clause_j$. For each $t = 1, 2, 3$,
   - if $\lambda^{(j)}_t$ corresponds to the positive variable x_i then create a connection so that the input of $Clause_j$ that is labelled $\ell^{(j)}_t$ is the output χ_i of the generating $Gen(\chi_i)$ associated to x_i.
   - if $\lambda^{(j)}_t$ corresponds to the negated variable $\neg x_i$ then make a connection so that the input of $Clause_j$ that is labelled $\ell^{(j)}_t$ is the output of a negation gadget, and the input of such negation gadget is the output χ_i of the Generating-gadget $Gen(\chi_i)$.

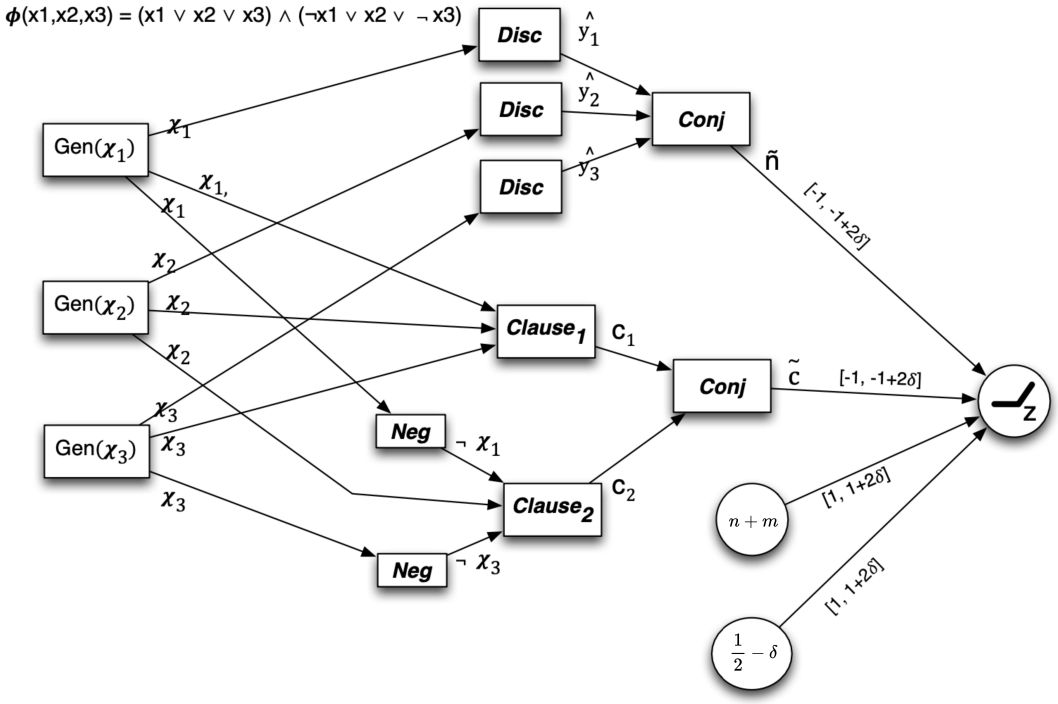


Fig. 5.6: A complete example of the reduction on a simple formula, with $n = 3$ variables and $m = 2$ clauses. All the interval weights not explicitly given are $[1 - 2\delta, 1]$

4. Add a conjunction gadget such that its inputs are the outputs $\hat{y}_i$ $(i = 1, \ldots, n)$ of the *Discretizer*-gadgets. Let $\tilde{n}$ denote the output of such conjunction gadget.
5. Add a conjunction gadget such that its inputs are the outputs c_j $(i = 1, \ldots, m)$ of the *Clause*-gadgets. Let $\tilde{c}$ denote the output of such conjunction gadget.
6. Finally, add an *End*-gadget (Fig. 5.5) and connect it to the rest of the network by making the output $\tilde{n}, \tilde{c}$ of the above conjunction gadgets (defined in items 4, 5) coincide with the *End*-gadget inputs marked with $\tilde{n}$ and $\tilde{c}$, respectively.

The above construction defines the topology of the DNN, representing the parametric classifier $\mathcal{M}_\Theta$. The classifier $\mathcal{M}^\phi_{\theta_1}$ is chosen to be the realization of $\mathcal{I}^\phi$ obtained by setting the weight on each edge e to the middle point of the interval associated to e. Such a classifier takes as input x a vector whose components are

- the value in the leftmost node of each Generating-gadget. In the input x^ϕ defined for our reduction these values are set to 1 as in Fig.5.7 ;
- the values in the first (top) and the two last (bottom) nodes in each Discretizer-gadget. In the input x^ϕ defined for our reduction these values are set to δ, $(1 - \delta)$, and $(1 - \delta)$, respectively as in Fig.5.3;
- the values in the lowest node of each Clause-gadget. In the input x^ϕ defined for our reduction these values are set to 1 as in Fig.5.4;

- the values in the lowest node of each Negation-gadget. In the input x^ϕ defined for our reduction these values are set to 1 as in Fig.5.4;
- the values in two bottom nodes of the End-gadget. In the input x^ϕ defined for our reduction these values are set to $n + m$ and $\frac{1}{2} - \delta$, respectively, as in Fig.5.5.

Finally Δ^ϕ is defined as the set of realizations of $\mathcal{I}$.

It is easy to see that by fixing the value δ so that it can be encoded by number of bits polynomial in the size of ϕ, **the instance I^ϕ can be constructed from ϕ in polynomial time**, since each gadget has a constant size, the number of gadgets is polynomial in the size of the formula, and the input vector x can be described by a number of bits polynomial in the size of δ and the size of $\mathcal{I}^\phi$.

We first prove a lemma that characterises realizations of $\mathcal{I}$ such that the output of each χ_i is binary.

Lemma 5.7: *The following two claims characterise the realizations of $\mathcal{I}$ such that for each $i = 1, \ldots, n$, it holds that $\chi_i \in \{0, 1\}$.*

1. *Fix a truth assignment* **a** *such that $\phi(\mathbf{a}) = false$. For any realization $\mathcal{M}_\theta$ of $\mathcal{I}$ such that for each $i = 1, \ldots, n$ it holds that $\chi_i = 0$ if $a_i = false$ and $\chi_i = 1$ if $a_i = true$ it holds that $\mathcal{M}_\theta(x) \geq \frac{1}{2}$.*
2. *Fix a truth assignment* **a** *such that $\phi(\mathbf{a}) = true$. Then, there exists a realization $\mathcal{M}_\theta$ of $\mathcal{I}$ such that for each $i = 1, \ldots, n$ it holds that $\chi_i = 0$ if $a_i = false$ and $\chi_i = 1$ if $a_i = true$, and $\mathcal{M}_\theta(x) < \frac{1}{2}$.*

Proof. We show the two claims separately.

1. For the first claim, we observe that
   a) $\tilde{n} \leq n$.
   b) Since $\phi(\mathbf{a}) = false$, there exists $i \in [m]$ such that the assignment **a** makes all the literals in the ith clause to be false. We also have that the values $\ell_t^{[i]}$s, input to the clause gadget encoding the ith clause, will satisfy $\ell_t^{[i]} \in [0, \delta]$— in particular, we have $\ell_t^{[i]} = \chi_j = 0$ if the literal corresponds to some variable $x_j = false$; and, if the the literal correspond to the negation of some variable $x_j = true$, hence $\ell_t^{[i]} = \neg\chi_j$ with $\chi_j = 1$ and by Lemma 5.3 $\neg\chi_j \in [0, \delta]$.

   Therefore, by Lemma 5.4, we have $c_i \in [0, 5\delta + 6\delta^2]$. It follows that $\tilde{c} < m - \delta$ and

$$z \geq \max\{0, \frac{1}{2} - \delta + n + m - \tilde{n} - \tilde{c}\} \geq \frac{1}{2} - \delta + n + m - n - m + \delta = \frac{1}{2}$$

2. For the second claim, consider the realization obtained by setting the weights as follows:
   - in the i-th generating gadget (the one associated to x_i) the weights are chosen in order to have output $\chi = 1$ if $a_i = true$ and $\chi = 0$ if $a_i = false$;
   - in all the other gadgets, the weights are set to the value w such that $|w| = 1$.

   Because of the correspondence $a_i = true \rightarrow \chi_i = 1$ and $a_i = false \rightarrow \chi_i = 0$, by Lemma 5.2, we have $\hat{y}_i = 1$ for each $i = 1, \ldots, n$. Hence $\tilde{n} = n$.
   Because of the choice of the weights being all of the absolute value one, it is also easy to see that, interpreting $true$ as 1 and $false$ as 0, for each $i = 1, \ldots, m$ and $t = 1, 2, 3$, we have an exact correspondence between the truth value assigned by

a to the tth literal of the ith clause and the value $\ell_t^{[i]}$. Hence, by the assumption that $\phi(\mathbf{a}) = true$, we also have that $c_i = 1$ for each $i = 1, \ldots, m$. It follows that $\tilde{c} = m$.

Therefore,

$$z = \max\{0, \frac{1}{2} - \delta + n + m - \tilde{n} - \tilde{c}\} = \frac{1}{2} - \delta < \frac{1}{2}$$

□

Let $\mathcal{M}_{\theta_1}$ be the realization of $\mathcal{I}$ obtained by setting the weight on each edge e to the middle point of the interval associated to e. Let **a** be the assignment for ϕ such that $a_i = false$ for each $i = 1, \ldots, n$. Therefore, $\mathcal{M}_{\theta_1}$ coincides with the realization of $\mathcal{I}$ such that for each $i = 1, \ldots, n$ it holds that $\chi_i = 0$ and the hypotheses of Lemma 5.7 are satisfied Hence, by Lemma 5.7, it holds that $\mathcal{M}_\theta(x) \geq \frac{1}{2}$. We have shown the following.

Lemma 5.8: *For each instance ϕ of* 3-NAF-SAT*, the reduction $\mathcal{R}$ produces in polynomial time a proper instance $(\mathcal{M}_{\theta_1}, \mathcal{I}, x)$ of* DRP.

In order to complete the proof of the Theorem, the following remains to be shown.

Lemma 5.9: *The formula ϕ is satisfiable if and only if there is a realization $\mathcal{M}_{\theta_2}$ of $\mathcal{I}$, such that $\mathcal{M}_{\theta_2}(x) < \frac{1}{2}$.*

Proof. The sufficiency of the condition directly follows from the second claim of Lemma 5.7, which shows that: *If there exists an assignment* **a** *such that $\phi(\mathbf{a}) = true$ then there is a realization $\mathcal{M}_{\theta_2}$ of $\mathcal{I}$, such that $\mathcal{M}_{\theta_2}(x) < \frac{1}{2}$.*

Let us now focus on the other direction. Assume that there is a realization $\mathcal{M}_{\theta_2}$ such that $\mathcal{M}_{\theta_2}(x) < \frac{1}{2}$. By Lemma 5.6, it follows that for the realization $\mathcal{M}_{\theta_2}(x)$ it holds that $\tilde{n} > n - \delta$ and $\tilde{c} > m - (5\delta + 8\delta^2 + 4\delta^3)$.

Then, by Lemma 5.5 it follows that

1. for each $i = 1, \ldots, n$ it holds that $\hat{y}_i \in (1 - \delta, 1]$, and $\chi_i \in [0, \delta] \cup [1 - \delta, 1]$;
2. for each $i = 1, \ldots, m$ it holds that $c_i \in (1 - (5\delta + 8\delta^2 + 4\delta^3), 1]$.

These two conditions together with $\delta < 1/12$ imply, by Lemmas 5.2, 5.3 and 5.4, that for each $i = 1, \ldots, m$, there is $t \in \{1, 2, 3\}$ such that one of the following holds

1. $\ell_t^{[i]} \in [1 - 3\delta - 2\delta^2, 1]$ and $\ell_t^{[i]}$ coincides with some output $\neg\chi_j$ of a negation-gadget, and the input value satisfies $\chi_j \in [0, \delta]$; moreover by construction, then literal $\lambda_t^{[i]}$ is $\neg x_j$;
2. $\ell_t^{[i]} \in [1 - \delta, 1]$ and $\ell_t^{[i]}$ coincides with some output χ_j of a generating-gadget, whence by construction, then literal $\lambda_t^{[i]}$ is x_j

Let $\mathbf{a} = (a_1, \ldots, a_n)$ be the truth assignment defined by

$$a_i = \begin{cases} true & if\ \chi_j \geq 1 - \delta \\ false & if\ \chi_j \leq \delta. \end{cases}$$

Then, for each clause $i = 1, \ldots, m$ at least one literal is set to $true$, and $\phi(\mathbf{a}) = true$.

□

The proof is complete. □

From Theorem 5.2, it follows that deciding whether a CE x' is not robust to a set of PMC Δ is NP-complete. We now show that the above reduction can be used to prove the NP-completeness of deciding robustness with respect to a set Δ of NOMC models. In particular, we have the following:

Theorem 5.3 (Hardness of DRP for NOMC): *Given classifier $\mathcal{M}_{\theta_1}$, an input x and a set Δ of NOMC, deciding whether $\exists\ \mathcal{M}_{\theta_2} \in \Delta$ s.t $\mathbb{E}[\mathcal{M}_{\theta_2}(x)] < \frac{1}{2} \leq \mathcal{M}_{\theta_1}(x)$ is NP-complete.*

Proof. The proof of the inclusion in NP is analogous to the one of DRP for PMC models.

For the hardness, we use again a reduction from 3-NAF-SAT: given a 3-CNF ϕ that is not satisfied by the all-false assignment, build an interval neural network exactly like in Theorem 5.2 but for one difference consisting in the interval of weights on the first edge of the Generating-gadget, which are now set to $[0, 2\delta]$ as in Fig. 5.7.

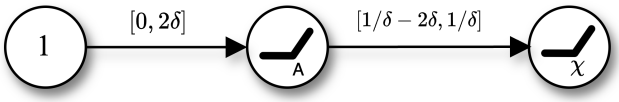


Fig. 5.7: *Generating-gadget* used in this proof.

We denote by $\mathcal{I}_{NOMC}$ this interval neural network. obtained from this reduction starting from a 3-CNF ϕ. Note that the new generating-gadget can also produce any value in $[0, 1]$. More generally, we have the following important remark.

Remark 1: Lemmas 5.1-5.6 also hold for the interval neural network $\mathcal{I}_{NOMC}$.

We define $\mathcal{M}_{\theta_1}$ to be the realization of $\mathcal{I}_{NOMC}$ obtained by setting the weight on each edge e to the middle point of the interval associated with e. The input value x is defined as in the proof of Theorem 5.2. We also let Δ to be the set of realization of $\mathcal{I}_{NOMC}$.

From the above remark and the proof of Theorem 5.2, it follows that the 3-CNF ϕ has a satisfying assignment if and only if there exists a realization $\mathcal{M}_{\theta_2}$ in Δ such that $\mathcal{M}_{\theta_2}(x) < 0.5$.

Then, in order to complete the proof, we only need to show that $\mathcal{M}_{\theta_1}(x) \geq \frac{1}{2}$ and Δ properly defines a set of NOMC for $\mathcal{M}_{\theta_1}$ (Def. 2.9), which we recall here for readability purposes:

1. $\mathbb{E}[\mathcal{M}_\theta(x)] = \mathcal{M}_{\theta_1}(x)$; where the expectation is over the randomness[4] of $\mathcal{M}_\theta$;
2. $\mathrm{Var}[\mathcal{M}_\theta(x)] = \nu_x$, where ν_x represents the maximum variance of the prediction of $\mathcal{M}_\theta(x)$, and whenever x lies on the data manifold $\mathcal{X}$, ν_x is upper bounded by a small constant ν;

[4] In our case this is a *random realization of $\mathcal{I}_{NOMC}$*, i.e., a realization of $\mathcal{I}_{NOMC}$obtained by independently choosing the weight on each edge sampling uniformly at random from the interval associated to that edge.

3. If $\mathcal{M}_{\theta_1}$ is Lipschitz continuous for some γ_1, then $\mathcal{M}_\theta$ is also Lipschitz continuous for some γ_2.

For a (random or fixed) realization $\mathcal{M}_\theta$ of $\mathcal{I}_{NOMC}$ and a node v, let us denote by v_θ the value computed in the node v by $\mathcal{M}_\theta$ on input x. In accordance with the analysis in Theorem 5.2, we assume $\delta = 1/12$.

To show that $\mathcal{M}_{\theta_1}(x) \geq \frac{1}{2}$ and that Δ satisfies property 1 for being a set of NOMC, we prepare the following.

Lemma 5.10: *Let $\mathcal{M}_\theta$ be a random realization of $\mathcal{I}_{NOMC}$. It holds that*

1. *for the output node χ of each Generating-gadget, we have $\mathbb{E}[\chi_\theta] = \chi_{\theta_1} = \frac{1}{2} - \delta^2$.*
2. *for the the output node $\neg\chi$ of each Negation-gadget we have $\mathbb{E}[\neg\chi] = \neg\chi_{\theta_1} = \frac{1}{2} - \frac{3}{2}\delta + \delta^2 + \delta^3$.*
3. *for the the output node $\hat{y}$ of each Discretizing-gadget we have $\mathbb{E}[\hat{y}_\theta] = \hat{y}_{\theta_1} = 1-\delta$*
4. *for the output node c of each Clause-gadget we have $\mathbb{E}[c_\theta] = c_{\theta_1} = 1 - \delta$*
5. *for the output node $\tilde{c}$ of the Conjunction-gadget collecting the outputs of the Clause-gadgets we have $\mathbb{E}[\tilde{c}_\theta] = \tilde{c}_{\theta_1} = m(1-\delta)^2$*
6. *for the output node $\tilde{n}$ of the Conjunction-gadget collecting the outputs of the Discretizing-gadgets we have $\mathbb{E}[\tilde{n}_\theta] = \tilde{n}_{\theta_1} = n(1-\delta)^2$*
7. *for the output node z of the End-gadget, we have $\mathbb{E}[z_\theta] = z_{\theta_1} \geq \frac{1}{2}$.*

Proof.

1. Let w_1, w_2 denote the weights on the edges of the Generating-gadget, respectively, in order from left to right. Because of the independence of the choices of the weights of the random realization $\mathcal{M}_\theta$, we have

$$\mathbb{E}[\chi_\theta] = \mathbb{E}[w_2]\mathbb{E}[A_\theta] = \mathbb{E}[w_2]\mathbb{E}[w_1] = \left(\frac{1}{\delta} - \delta\right)\delta = \frac{1}{2} - \delta^2.$$

   It is immediate to verify that this value is equal to χ_{θ_1}.
2. Let w_1, w_2 denote the weights on the top edge and the bottom edge, respectively, of the Negation-gadget. Using 1. the independence in the choice of the weights, and the fact that with $\delta = \frac{1}{12}$ the argument of the ReLU is always non-negative, we have that

$$\mathbb{E}[\neg\chi_\theta] = \mathbb{E}[\chi_\theta] \cdot \mathbb{E}[w_1] + \mathbb{E}[w_2].$$

   The claim then follows by the fact that the expected values of the weights are given by the middle point of the intervals from which they are respectively taken.
3. Item 1. and the first claim of Lemma 5.2, together with $\delta = 1/12$ imply that both for a random realization and for the realization $\mathcal{M}_{\theta_1}$, the (expected) values computed in nodes A and B of the discretizing-gadget are both 0. Hence, we have $\mathbb{E}[\hat{y}] = \mathbb{E}[w_\theta]$, where w denotes the weight on the lowest edge of the gadget. By noticing that this expected value is equal to the middle point of the interval, we have the desired result.
4. Because of 1. and 2. we have that the expected value (as well as the value computed by $\mathcal{M}_{\theta_1}$ on x) of the input nodes $\ell_t^{[i]}$ of each clause-gadgets are from the set $\{\frac{1}{2} - \delta^2, \frac{1}{2} - \frac{3}{2}\delta + \delta^2 + \delta^3\}$ It follows that the argument of the ReLU function computed in the node A is negative. Hence, we have $\mathbb{E}[c] = \mathbb{E}[w_\theta]$, where w

denotes the weight on the lowest edge of the gadget. Again, noticing that this expected value is equal to the middle point of the interval gives the desired result.

5. For a realization $\mathcal{M}_\theta$ let $w_{\theta,i}$ denote the weight on the ith edge (counting from top to bottom) of the conjunction-gadget collecting the outputs of the Clause-gadgets, and $c_{\theta,i}$ the output value of the ith clause gadget, as computed by $\mathcal{M}_\theta$ on input x. Then, we have

$$\mathbb{E}[\tilde{c}_\theta] = \sum_{i=1}^{m} \mathbb{E}[c_{\theta,i}] \cdot \mathbb{E}[w_{\theta,i}].$$

The results follow from 4. and the fact that the expected value of a uniformly sampled weight is equal to the middle point of the interval from which it is taken.

6. The proof of this point is analogous to the proof of 5.
7. For a realization $\mathcal{M}_\theta$ let $w_{\theta,i}$ denote the weight on the ith edge (counting from top to bottom) of the End-gadget. We start by observing that from the results of the previous points, it follows that the argument of the RELU function in node z is always non-negative. Hence, we have

$$\mathbb{E}[z_\theta] = \mathbb{E}[\tilde{n}_\theta] \cdot \mathbb{E}[w_{\theta,1}] + \mathbb{E}[\tilde{c}_\theta] \cdot \mathbb{E}[w_{\theta,2}] + (n+m) \cdot \mathbb{E}[w_{\theta,3}] + \left(\frac{1}{2} - \delta\right) \cdot \mathbb{E}[w_{\theta,4}]$$

Then, the equality $\mathbb{E}[z_\theta] = z_{\theta_1}$ in the claim follows again from the fact that the expected values of the weights of a random realization are equal to the middle point of the interval, i.e., the value of the weight on the edge in the realization $\mathcal{M}_{\theta_1}$. The inequality in the claim follows from Lemma 5.6 since, with $\delta = 1/12$, it holds that $n(1-\delta)^2 < n - \delta$.

□

Claim 7 of the lemma directly implies that the first property of an NOMC is satisfied by Δ, i.e., $\mathbb{E}[\mathcal{M}_\theta(x)] = \mathcal{M}_{\theta_1}(x)$. The same claim also proves that $\mathcal{M}_{\theta_1}(x) \geq \frac{1}{2}$.

For the second property, namely $\mathrm{Var}[\mathcal{I}(x)] = \nu_x$, we use the uniform continuity of the function computed by the realizations of $\mathcal{I}_{NOMC}$, which is a direct consequence of being linear combinations of RELU functions which are Lipschitz continuous functions, hence uniform continuous. By the Extreme Value Theorem (see, e.g., [228, Thm. 4.16]) any realization of $\mathcal{I}_{NOMC}$ will be bounded and achieve its minimum and maximum on the compact domain, and thus the variance will be indeed bounded.

Finally, the last property follows from the fact that the linear composition of Lipschitz continuous operations (ReLU) is also Lipschitz continuous, which is indeed the case of any realization of $\mathcal{I}_{NOMC}$. □

Summarizing, we have shown the following.

Corollary 4: *Given a model $\mathcal{M}_\theta$, a CE x' and a set of either NOMC or PMC model changes Δ, the problem of verifying the Δ-robustness of x' is NP-complete.*

These hardness results motivate the introduction of novel approximate solutions to estimate the robustness of a counterfactual under a set of PMC Δ.

5.1.2 Probabilistic Guarantees for Existing Notions of Model Changes

As we have established in the previous section, exact methods for computing robustness under model changes are bound to lack scalability. This motivates the design of approximate and/or probabilistic approaches to solve the problem. Previous work by [94] presented an approach to obtain counterfactual explanations that are probabilistically robust under NOMC. A natural question that arises then is whether guarantees obtained for NOMC also transfer to the PMC setting. As we show for the first time below, this is not the case in general.

Lemma 5.11: *Naturally-Occurring model changes may not be Plausible, and vice-versa.*

Proof. Consider the DNN $\mathcal{M}_\theta$ depicted in Fig. 5.8 (a) with two input nodes, one hidden layer with two ReLU nodes and one single output.[5] The parameters $\theta = [w_1, \ldots, w_6]$ are the weights on the edges listed top-bottom and left-right.

Fig. 5.8: (a) The model $\mathcal{M}_\theta$ used as an example to prove the lemma. (b) An interval neural network representing the realizations that can be obtained from $\mathcal{M}_\theta$ considering a set of PMC Δ_δ with $\delta = 0.3$.

Propagating an input vector $x = [x_1, x_2]^T$ through $\mathcal{M}_\theta$, we obtain $\mathcal{M}_\theta(x) = y = w_5 \cdot \max\{0, w_1 \cdot x_1 + x_2 \cdot w_3\} + w_6 \cdot \max\{0, w_2 \cdot x_1 + x_2 \cdot w_4\}$. Now assume an input vector $x = [0.9, 0.9]^T$ and weights $w_1 = 1, w_2 = 0, w_3 = 0, w_4 = 0.6, w_5 = 1, w_6 = -1$. The corresponding output generated by the DNN is $\mathcal{M}_\theta(x) = 0.46$. A counterfactual for x could be given as a new input vector $x' = [1, 0.8]^T$, for which we obtain $\mathcal{M}_\theta(x') = 0.52 > 0.5$. Now, following Definition 2.10, we consider a set of plausible model changes obtained for $\delta = 0.3$. This can be captured by defining on each weight w_i the corresponding interval in $[w_i - \delta, w_i + \delta]$ depicted in Fig. 5.8 (b) that represents the set of all the possible models obtained from $\mathcal{M}_\theta$, replacing each w_i with a weight in the interval $[w_i - \delta, w_i + \delta]$. We then have that the expected result of a model $\mathcal{M}_{\theta'}$ sampled uniformly from such a set satisfies:

[5] In this proof, we consider a DNN with only ReLU activation functions. However, we notice that it is possible to have a similar counterexample even with other activations, e.g., Tanh, Sigmoid.

$$\begin{aligned}\mathbb{E}[\mathcal{M}_{\theta'}(x')] &= \mathbb{E}[w_5] \cdot \mathbb{E}[\text{ReLU}(x_1 \cdot w_1 + x_2 \cdot w_3)] + \\ &\quad \mathbb{E}[w_6] \cdot \mathbb{E}[\text{ReLU}(x_1 \cdot w_2 + x_2 \cdot w_4)] \\ &= \mathbb{E}[[0.7, 1.3]] \cdot \mathbb{E}[\max\{0, x_1 \cdot [0.7, 1.3] + \\ &\quad x_2 \cdot [-0.3, 0.3]\}] + \mathbb{E}[[-1.7, -0.3]] \cdot \\ &\quad \mathbb{E}[\max\{0, x_1 \cdot [-0.3, 0.3] + x_2 \cdot [0.3, 0.9]\}] \\ &\quad > 0.52 \neq \mathcal{M}_\theta(x')\end{aligned}$$

Definition 2.9 states that a model change is naturally occurring if $\mathbb{E}[\mathcal{M}_{\theta'}(x)] = \mathcal{M}_\theta(x)$. This implies that Δ contains models that cannot be characterized as naturally occurring model changes. Vice versa, the existence of Naturally-Occurring model changes not being plausible is implicit in the definition, and for the sake of completeness, we provide an example network in Fig. 5.9.

Consider a DNN having a single input value x and a single parameter θ and computing the function $\mathcal{M}_\theta(x) = \text{ReLU}(0.5 - \text{ReLU}(x - \theta))$

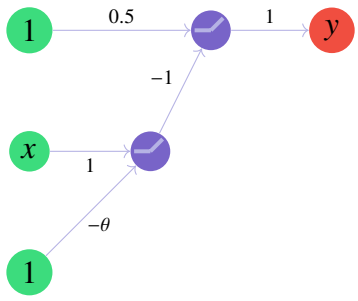


Fig. 5.9: The DNN considered in this proof

Fix a data set $\mathcal{X}$ and let $\theta = \max_{x \in \mathcal{X}} x$. Let us consider the set of model changes $\Sigma = \{S_\tau \mid \tau \in \mathbb{R}_+\}$ defined by $S_\tau(\mathcal{M}_\theta) = \mathcal{M}_{\theta+\tau}$. Clearly for any $\tau \geq 0$, we have

$$\mathcal{M}_{\tau+\theta}(x) = \mathcal{M}_\theta(x) = 0.5,$$

for any $x \in \mathcal{X}$. This trivially implies that Σ is a set of naturally occurring model changes (all changes considered have exactly the same value in all points in $\mathcal{X}$).

The claim now follows by observing that there is no finite δ such that the corresponding set of plausible model changes $\Delta = \{S \mid d_p(\mathcal{M}_\theta, S(\mathcal{M}_\theta)) \leq \delta\}$ contains Σ.

□

Lemma 5.11 shows the existence of witnesses proving that Definition 2.9 (NOMC) and Definition 2.10 (PMC) may capture very different model changes in general.[6] To complement this observation, we also ran experiments to determine how often these definitions disagree empirically. In particular, we considered three binary classification datasets commonly used in Explainable AI:

[6] We stress that this distinction does not preclude the existence of model changes that satisfy both definitions simultaneously.

- the *credit* dataset [60], which is used to predict the credit risk of a person (good or bad) based on a set of attributes describing their credit history;
- the *spambase* dataset [105] is used to predict whether an email is to be considered spam or not based on selected attributes of the email;
- the *online news popularity* dataset [70], referred to as *news* in the following, is used to predict the popularity of online articles.

We trained a neural network classifier with two hidden layers (20 and 10 neurons, respectively) for each dataset and used a Nearest-Neighbor Counterfactual Explainer [89] to generate counterfactual explanations for 10 different inputs. After generating a counterfactual, we produce n different perturbations $\mathcal{M}_{\theta'}$ of the original neural network $\mathcal{M}_\theta$ for $n \in \{1000, 10000\}$ under *plausible* model change with $\Delta \in \{0.05, 0.1, 0.2, 0.3\}$. We then considered two measures:

- average difference in output between $\mathcal{M}_\theta$ and $\mathcal{M}_{\theta'}$, for each of the n model $\mathcal{M}_{\theta'}$ and across all CEs;
- for each counterfactual, we perform a one-sided t-test to check whether the average prediction generated by n models $\mathcal{M}_{\theta'}$ equals the original prediction of $\mathcal{M}_\theta$. We report the percentage of CEs for which the null hypothesis was rejected (p-value used 0.05).

	Credit				***Spam***				***News***			
	n = 1000		n = 10000		n = 1000		n = 10000		n = 1000		n = 10000	
	Avg diff.	Rej. (%)	Avg diff.	Rej. (%)	Avg diff.	Rej. (%)	Avg diff.	Rej. (%)	Avg diff.	Rej. (%)	Avg diff.	Rej. (%)
δ = 0.05	0.008	90	0.022	90	0.018	50	0.017	70	0.034	70	0.033	80
δ = 0.1	0.017	100	0.047	100	0.034	100	0.035	100	0.064	80	0.063	100
δ = 0.2	0.046	100	0.086	100	0.0748	90	0.064	100	0.127	90	0.141	100
δ = 0.3	0.110	100	0.140	90	0.121	100	0.087	100	0.207	90	0.173	100

Table 5.1: Empirical evaluation across model perturbations of increasing magnitude δ and different sample sizes n.

Table 5.1 reports our results. We observe that the requirement that the expected output of perturbed models remains equal to the original prediction is often violated. These results complement the result of Lemma 5.11, confirming that the two notions indeed capture two different settings in general. In particular, our results show that (probabilistic) methods devised for NOMC may fail to guarantee robustness under PMC, thus motivating the development of dedicated approaches for probabilistic guarantees under PMC. These results complement observations made in prior work (see, e.g., Table 3 in [119]) and motivate the development of dedicated approaches for probabilistic guarantees under PMC.

Indeed, having clarified the relationship between the two notions of model changes, in the following, we focus on certification approaches for robustness under PMC, presenting a novel approximate solution with probabilistic guarantees.

5.1.3 Robustness under PMC with Probabilistic Guarantees

[116, 119] proposed to use INNs to enable a compact representation of a superset of the models that can be obtained by a perturbation of the starting model under a set Δ. By exploiting an exact reachable set computation method, e.g., based on MILP [253], the authors could determine whether or not a CE is robust under the chosen Δ via a single forward propagation of the CE. However, in view of the NP-hardness of the problem discussed in the SubSec. 5.1.1 and the typical non-linear nature of the classifiers, it presents some computational limitations.

In general, interval neural networks map inputs to intervals representing an over-approximation of all possible outcomes that can be produced by any shifted model $\mathcal{M}_{\theta'}$ obtained under Δ. Given this property, if the output reachable set is completely disjoint from the decision threshold 0.5, then one can assert – in a sound and complete fashion – whether or not a given CE is robust (Fig. 5.10 (a,c)). On the other hand, if we run into a situation such as the one depicted in Fig. 5.10 (b), one cannot assert robustness with certainty. In this scenario, [116] propose to classify the CE as not robust, which preserves the soundness of their result. Nonetheless, this might lead to discarding a CE even when the actual probability that after retraining, we incur in plausible model changes for which the CE is not robust is extremely low. As we will show in SubSec. 5.1.4, this worst-case notion of robustness affects the CEs generated by [116], which may end up being unnecessarily expensive (in terms of proximity) and having low plausibility. Additionally, computing the exact output reachable set of an interval abstraction may be costly (e.g., MILP is known to be NP-hard). This is expected: Theorems 5.2 and 5.1.1 show that there is no polynomial time algorithm able to return an exact estimate of the fraction of plausible changes for which the CE is robust (hence a fortiori deciding whether it is Δ-robust), unless P=NP. In the following, we propose a novel certification approach that aims to alleviate this problem.

A Provable Probabilistic Approach

One possible idea to avoid exact reachable set computation to determine the robustness of a CE under PMC is to use naive interval propagation. Given an input CE, we propagate this input through the network, keeping track of all the possible activation

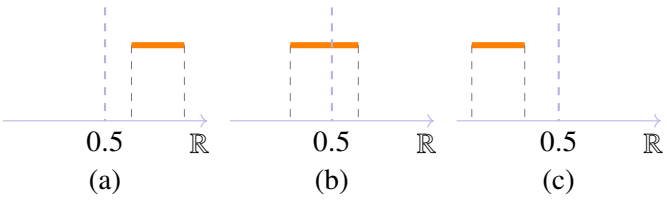


Fig. 5.10: Visual representation of the possible output reachable set for an interval abstraction for a binary classification model. (a) For a given Δ, we classify an input as 1 (robust) if the output range for that input is always greater 0.5. Otherwise, the input is classified as 0, i.e., not robust (b),(c).

values that can be obtained under Δ until the output layer is reached. However, the non-linear and non-convex nature of DNNs may result in a significant overestimation of the actual reachable set, thus resulting in a spurious decision of non-robustness. In such cases, a CE may end up being labeled as non-robust even though the CE is actually robust. Additionally, even with exact methods, a CE may be discarded even though the fraction of plausible model changes in Δ for which the CE is not robust is negligible.

To avoid these problems, we propose an approximate certification approach based on Monte-Carlo sampling that draws sample realizations directly from Δ to obtain an underestimation of the space of possible classifications under PMC. The idea of using a sample-based approach stems from the fact that the Δ set, representing all the plausible model changes, abstracts an infinite number of models to test. As testing this infinite number of models may be impossible in practice, efficient sampling-based solutions hold great promise. In detail, given a CE x' we can compute an underestimation of the output reachable set under Δ by sampling n random realizations $\mathcal{M}_{\theta_1}, \ldots, \mathcal{M}_{\theta_n}$ from Δ, and compute the output reachable set by taking, respectively, the $\min_i \mathcal{M}_{\theta_i}(x')$ and the $\max_i \mathcal{M}_{\theta_i}(x')$ for $i \in \{1, \ldots, n\}$.

This approach is very effective and allows us to obtain an estimate of the output reachable set without using an exact solver. Nonetheless, the number n of realizations to sample in order to achieve a good reachable set estimation remains unclear, as well as what kind of guarantees one could obtain from this approach. To answer these questions, we leverage previous results on the *statistical prediction of tolerance limits* [286]. Indeed, we observe that for each realization $\mathcal{M}_{\theta_i}$ sampled from Δ, the resulting output of the DNN $\mathcal{M}_{\theta_i}(x')$ can be interpreted as an instantiation of a random variable X whose tolerance interval we are trying to estimate. Following this observation, we can derive a probabilistic bound on the correctness of the solution returned from n samples, using the following lemma based on [286]:

Lemma 5.12: *Fix an integer $n > 0$ and an approximation parameter $R \in (0, 1)$. Given a sample of n models $\mathcal{M}_{\theta_1}, \ldots \mathcal{M}_{\theta_n}$ possible realizations $\mathcal{M}_{\theta_1}, \ldots \mathcal{M}_{\theta_n}$ from the set of PMC Δ, the probability that for at least a fraction R of the models in a further possibly infinite set of realizations $\mathcal{M}^{(2)}_{\theta_1}, \ldots \mathcal{M}^{(2)}_{\theta_m}$ from Δ we have*

$$\min_i \mathcal{M}^{(2)}_{\theta_i}(x) \geq \min_i \mathcal{M}_{\theta_i}(x) \tag{5.1}$$

$$\textit{(respectively } \max_i \mathcal{M}^{(2)}_{\theta_i}(x) \leq \max_i \mathcal{M}_{\theta_i}(x)\textit{)}$$

is given by $\alpha = n \cdot \int_R^1 x^{n-1}\, dx = 1 - R^n$.

Informally, Lemma 5.12 allows us to derive the minimum number n of realizations that it is enough to sample and check in order to guarantee that with probability α at least a fraction R of the models in Δ satisfy the robustness property. More precisely, from these n realizations, we can obtain an underestimation of the reachable set of any realization in Δ that is guaranteed to be correct with confidence α for at least a fraction R of a possibly infinite further sample of realizations from Δ. In practice, if we set, e.g. $\alpha = 0.999$ and $R = 0.995$, we can derive n as $n = \log_R(1 - \alpha) = 1378$. After having selected 1378 random realizations from Δ, if the lower bound of the

underestimated reachable set computed as $\min_i \mathcal{M}_{\theta_i}(x')$ is greater than 0.5, then with probability $\alpha = 0.999$, R is a lower bound on the fraction of plausible model changes in Δ for which x' is robust. In other words, Lemma 5.12 allows us to assert with a confidence α that x' is not Δ-robust for at most a fraction $(1 - R) = 0.05$ of models from Δ.

The APΔS Algorithm

Using the result of Lemma 5.12, we now present our approximation method APΔS to generate probabilistic robustness guarantees. The procedure, shown in Algorithm 15, receives as input a model $\mathcal{M}_\theta$, a CE x' for which robustness guarantees are sought, and the two confidence parameters α, R. The algorithm then searches for the largest δ_{max} such that, with probability α, the CE x' is robust for at least a fraction R of the set of plausible model changes $\Delta = \{S \mid d_p(\mathcal{M}_\theta, S(\mathcal{M}_\theta)) \leq \delta_{max}\}$.

Algorithm 15: Approximate Plausible Δ-Shift (APΔS)

1: **Input:** Model $\mathcal{M}_\theta$, set of PMC Δ, CE x', α, R, δ_{init}
2: **Output:** δ_{max}
3: $n \leftarrow \log_R(1 - \alpha)$ ▷ number of samples
4: rate $\leftarrow$ `realizations`$(\mathcal{M}_\theta, x', \delta_{init}, n)$
5: **if** rate $\neq 1$ **then**
6: **return** 0 ▷ not robust for δ_{init}
7: $\delta \leftarrow \delta_{init}$
8: **while** rate = 1 **do**
9: $\delta \leftarrow 2\delta$
10: rate $\leftarrow$ `realizations`$(\mathcal{M}_\theta, x', \delta, n)$

▷ we exit from the while because we have found at least one model in the realizations with an output < 0.5, and we have $[\delta/2,\ \delta)$ to search for a δ_{max}.

11: $\delta_{max} \leftarrow \delta/2$
12: **while** True **do**
13: **if** $|\delta - \delta_{max}| \leq \delta_{init}$ **then**
14: **return** δ_{max}
15: $\delta_{new} \leftarrow (\delta_{max} + \delta)/2$
16: rate $\leftarrow$ `realizations`$(\mathcal{M}_\theta, x', \delta_{new}, n)$
17: **if** rate = 1 **then**
18: $\delta_{max} \leftarrow \delta_{new}$
19: **else**
20: $\delta \leftarrow \delta_{new}$

The algorithm starts by computing the size n of a sample of realizations that is sufficient to guarantee the condition in Lemma 5.12 (line 3). APΔS then initializes a small δ_{init} and checks if x' is at least robust to a small model shift. To this end, it employs `realizations`$(\mathcal{M}_\theta, x', \delta, n)$ which samples n realizations, pertubating each model parameter by at most a factor δ and checks if for each of these realization $\mathcal{M}_{\theta_i}(x') \geq 0.5$, thus computing a robustness rate. If not all these realizations result in a robust outcome, thus achieving a final rate not equal to 1, the algorithm discards

the CE x' as non-robust. Otherwise, it combines an exponential search and a binary search to find $\delta_{\max}$. At each step of this search, the procedure checks whether for each of the n realizations from $\Delta = \{S \mid d_p(\mathcal{M}_\theta, S(\mathcal{M}_\theta)) \leq \delta_{max}\}$ the condition $\mathcal{M}_{\theta_i}(x') \geq 0.5$ is verified.

Proposition 5.1: *Fix $\delta_{init} > 0$. Given a model $\mathcal{M}_\theta$ and a CE x', let δ^* be the (exact) maximum magnitude of model changes such that x' is robust with respect to the set of PMC $\Delta_{\delta^*} = \{S \mid d_p(\mathcal{M}_\theta, S(\mathcal{M}_\theta)) \leq \delta^*\}$. Then, with probability α, $\texttt{AP}\Delta\texttt{S}$ returns a $\delta_{\max} \geq \delta^* - \delta_{init}$ such that the CE x' is robust for at least a fraction R of the set of PMC $\Delta_{\delta_{\max}}$. Moreover, the computation of δ_{max} is polynomial.*

Proof. Let $\delta^* = \sup\{\delta \geq 0 \mid x' \text{ is robust for all models in } \Delta_\delta\}$ denote the (unknown) exact maximal certified radius such that for every $\delta' \leq \delta^*$ and every $S \in \Delta_{\delta'}$ we have $\mathcal{M}_{\theta_S}(x') \geq 0.5$ (i.e., the maximum δ s.t. the CE remains valid).

The $\texttt{AP}\Delta\texttt{S}$ algorithm uses an exponential then binary search to locate the largest δ by drawing n independent random realizations from Δ_δ for various δ values, checking whether all n realizations satisfy $\mathcal{M}_{\theta_i}(x') \geq 0.5$. In the following we show that (i) the chosen sample size n yields the stated probabilistic guarantee (invoking Lemma 5.12), and (ii) the final numerical estimate $\delta_{\max}$ satisfies $\delta_{\max} \geq \delta^* - \delta_{init}$ with polynomial time complexity.

(i) Fix a desired $\alpha, R \in (0, 1)$ and any δ and let q_δ denote the true (unknown) fraction of models in Δ_δ for which $\mathcal{M}_{\theta_S}(x') \geq 0.5$. Sampling a realization S uniformly at random from Δ_δ yields a Bernoulli trial with success probability q_δ (success = the realization is robust). Draw n independent such realizations; the probability that all n trials succeed equals q_δ^n. Suppose the true fraction satisfies $q_\delta < R$. Then the probability of observing n successes is bounded by $\Pr[\text{all } n \text{ sampled realizations are robust} \mid q_\delta < R] \leq R^n$. Choose n such that $R^n \leq 1 - \alpha$, i.e., $n \geq \frac{\ln(1-\alpha)}{\ln R}$. With this choice, $\Pr[\text{all } n \text{ trials succeed} \mid q_\delta < R] \leq 1 - \alpha$. Equivalently, observing all n trials succeed is an event whose probability is at most $1 - \alpha$ under any hypothesis $q_\delta < R$. Therefore, by standard inversion of such an event (this is the same argument used in Lemma 5.12), if we observe all n successes then with confidence at least α the true fraction satisfies $q_\delta \geq R$. This establishes the probabilistic correctness of $\texttt{AP}\Delta\texttt{S}$ where observing n successful realizations for a given δ, then (by the choice of n) with probability at least α at least a fraction R of Δ_δ yields robustness.

(ii) The $\delta_{\max}$ returned by the algorithm is obtained by iteratively increasing δ, sampling n models from the corresponding Δ_δ and verifying that $\mathcal{M}_{\theta_i}(x) \geq 0.5$ for each model $\mathcal{M}_{\theta_i}$ sampled. By definition, δ^* is the actual value we are trying to estimate. When the algorithm stops, with values $\delta_{\max}$ and δ such that $\delta - \delta_{\max} \leq \delta_{init}$, we have by (i) that for n models in $\Delta_{\delta_{\max}}$ the CE x' is robust and for at least one model in Δ_δ the x' is not robust. Since for each model in Δ_{δ^*} the CEs is robust, it must hold that $\delta > \delta^*$, hence $\delta_{\max} \geq \delta^* - \delta_{init}$. Each call to $\texttt{realizations}(\mathcal{M}_\theta, x', \delta, n)$ requires n forward evaluations (one per sampled realization). The exponential search and binary search performs $O(\log(\delta_{\max}/\delta_{init}))$ in the worst case iterations to reach interval width δ_{init}. Therefore with a total time complexity of $O\big(n \cdot \log(\delta_{\max}/\delta_{init})\big)$, which is polynomial in the input instance, $\texttt{AP}\Delta\texttt{S}$ computes $\delta_{\max}$.

This completes the proof.

□

5.1.4 Experimental Analysis

Sec. 5.1.3 laid the theoretical foundations of a novel sampling-based method that allows the obtaining of provable probabilistic guarantees on the robustness of CEs. In this section, we evaluate our approach by considering five experiments:

- In SubSec. 5.1.4 we show how to instantiate APΔS in practice using a synthetic example. Specifically, we first demonstrate the interplay of parameters n, α, and R used to obtain a probabilistic guarantee. Then, using the maximum δ_{max} discovered by APΔS , we precisely characterize the subsets $\hat{\Delta}$ of the set of PMC $\Delta_{\delta_{max}}$ for which the given CE x' cannot be proved to be robust. In our experiments at most a fraction $(1 - R)$ of $\Delta_{\delta_{max}}$ is in $\hat{\Delta}$, so complementing empirically our theoretical results.
- In SubSec. 5.1.4 we compare our certification approach with the one proposed in [116]. In particular, we focus on the difference between the worst-case guarantees offered by their approach and compare them with the average-case guarantees of APΔS in terms of maximum changes that can be certified. These experiments confirm our intuition that worst-case guarantees might be too conservative in practice, leading to a larger number of CEs being discarded.
- In SubSec. 5.1.4, we consider the problem of generating robust CEs and compare with two state-of-the-art approaches for robustness under PMC, [116] and [262]. We show that our approach produces CEs that are less expensive (in terms of ℓ_1 distance from the original input) and more plausible, without sacrificing robustness.
- In SubSec. 5.1.4, we perform an in-depth analysis of the impact that the two main hyper-parameters of APΔS , α and R, have on the quality of generated CEs. We show that higher values of there parameters typically lead to tighter estimates that result in improved robustness. These results also align with existing literature on CEs in revealing that improved robustness appears to be correlated with higher plausibility and cost.
- Finally, in SubSec. 5.1.4 we analyse the scalability of APΔS . We consider tabular transformer architectures, such as *TabNet* [11] and show that APΔS scales well even when employed in recent architectures employed at the state of the art and containing hundreds of thousands of parameters, thus confirming the wide applicability of our method.

An implementation of APΔS is integrated in the *RobustX* library [122] available at `https://github.com/RobustCounterfactualX/RobustX.git`. Additional material is available at `https://github.com/lmarza/APAS`.

APΔS in Action

This experiment is designed to demonstrate how the three main parameters of APΔS , i.e., n, α, and R, can be used to obtain probabilistic robustness guarantees. To this end, we focus on the synthetic example depicted in Fig. 5.11. Weights for the original network $\mathcal{M}_\theta$, as well as the input used for testing robustness, are generated randomly. Considering a random input $x = -2.57$, we use APΔS to estimate a δ_{max} for which we seek the guarantee that for at least $R = 90\%$ of the plausible model changes induced

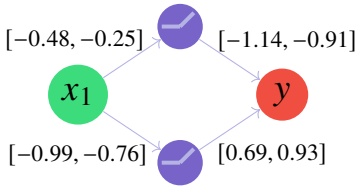


Fig. 5.11: The interval neural network used for exact enumeration.

by such δ_{max}, the CE x' is robust. Following Proposition 5.1, we set a confidence level $\alpha > 1 - 10^{-40}$ (i.e., with certainty, in practice), which yields $n = 100k$ realizations. For this setting, APΔS identifies a $\delta_{max} = 0.115$.

To validate this result, we define a procedure to exactly characterize, following the intuitions of [173, 176], the models within $\Delta_{\delta_{max}}$ for which the robustness property does not hold. The interval abstraction proposed by [116] can be used to exactly compute the portion of the model changes from Δ for which a CE x' is not robust. In fact, it is possible to build an interval neural network using the δ_{max} value identified by APΔS , setting each weight w_i in θ to $[w_i - \delta_{max}, w_i + \delta_{max}]$. Then, recursively splitting each interval weight of the network in half allows to identify portions of Δ that are not robust. Employing the following strategy (reported in Algorithm 16), after $s = 7$ splits, we obtain that for $\sim$ 92% of sub-interval networks, the CE is robust. The remaining 8% produced an *unknown* answer (i.e., the situation depicted in Fig. 5.10(b)) that would require further splits, corresponding to only ten nodes to explore in the next iteration. In the worst case, even considering all the remaining ten nodes left to explore as non-robust, we would have a maximum percentage of non-robustness still lower than the desired upper bound $(1 - R) = 10\%$, confirming that the guarantees produced by APΔS indeed hold in practice.

Algorithm 16: *Exact CE Δ-Robustness*

Input: An INN $\mathcal{N}$ and a CE x' and an maximum ϵ-precision for the splitting phase
Output: set of INNs for which x' is robust.
robust_INNs $\leftarrow \emptyset$
non-robust_INNs $\leftarrow \emptyset$
unknown $\leftarrow$ `Push`($\mathcal{N}$)
while (unknown $\neq \emptyset$) or (ϵ-precision not reached) **do**
  $\mathcal{I} \leftarrow$ `GetINNToVerify`(unknown)
  $\mathcal{R}_{\mathcal{I}} \leftarrow$ `ComputeReachableSet`($\mathcal{I}$, x')
  if `lower`($\mathcal{R}_{\mathcal{I}}$) ≥ 0.5 **then**
    robust_INNs $\leftarrow$ `Push`(I)
    unknown $\leftarrow$ `Pop`($\mathcal{I}$)
  else if `upper`($\mathcal{R}_{\mathcal{I}}$) < 0.5 **then**
    non-robust_INNs $\leftarrow$ `Push`($\mathcal{I}$)
    unknown $\leftarrow$ `Pop`($\mathcal{I}$)
  else
    $\mathcal{I}', \mathcal{I}'' \leftarrow$ `ChooseIntervalToSplit`($\mathcal{I}$)
    unknown $\leftarrow$ `Push`($\mathcal{I}', \mathcal{I}'$)
return robust_INNs

Worst-case vs Average-case Guarantees

This set of experiments aims to compare the probabilistic guarantees offered by APΔS with the worst-case guarantees offered by [116]. What we aim to show here is that adopting an average-case certification perspective may be more practical in some circumstances, as worst-case guarantees may be unnecessarily conservative.

The comparison between probabilistic certification and provable verification approaches is not meant to position our method as a replacement for sound and complete robustness certification techniques. Rather, our probabilistic certification framework is designed as a complementary tool that provides additional robustness insights, particularly in scenarios where provable solvers tend to be overly conservative and not scale to large models. Our approach aims to obtain a δ_{max} for which the CE is robust with confidence α for at least a fraction R of model changes in Δ. This is in stark contrast with the worst-case reasoning of [116], where even a single realization of Δ for which the CE is not robust results in the corresponding δ being discarded.

To show why such strict guarantees may not be needed, we use an analogous experimental setup and the training process of [116], which considers four datasets: *Diabetes* (continuous) [240], *Credit* (heterogeneous) [60], *no2* (continuous) [268] and *Small Business Administration* (SBA) (continuous features) [147]. In detail, for the training procedure of the classifier, we randomly shuffle each dataset and split it into two halves, denoted $\mathcal{D}_1$ and $\mathcal{D}_2$. First, we use $\mathcal{D}_1$ to train a base neural network; then we use both $\mathcal{D}_1$ and $\mathcal{D}_2$ to train a shifted model. We then generate 50 robust CEs for the base network using the MILP-R and the same δ values as in [116] for a fair comparison. Specifically, we use $\delta = 0.11$ for *Diabetes*, $\delta = 0.02$ for *no2*, $\delta = 0.11$ for *SBA* and $\delta = 0.05$ for *Credit*. Subsequently, we evaluate the resulting CEs by looking at two metrics: *(i)* **VM1**, the percentage of CEs that are valid on the base neural network and *(ii)* **VM2**, the percentage of CEs that remain valid for the shifted neural network trained using both $\mathcal{D}_1$ and $\mathcal{D}_2$. The table in Fig. 5.12 reports the results we obtained for this experiment. As previously observed by [116], the training procedure used to generate shifted models may result in changes that exceed the δ used to generate provably robust CEs. Indeed, after inspecting the networks obtained, we noted that the maximum empirical difference observed after retraining (denoted as δ_e) is well above the δ values used during CE generation. In particular, we recorded $\delta_e = 0.27$ for *Diabetes*, $\delta_e = 0.07$ for *no2*, $\delta = 0.25$ for *SBA* and $\delta_e = 1.28$ for *Credit*. Given the magnitude of these changes, the robustness of the CEs generated by MILP-R cannot be guaranteed in practice. However, the

	Diabetes				***no2***				***SBA***				***Credit***			
	VM1 $\delta = 0.11$	**VM2** $\delta_e = 0.27$	ℓ_1	**lof**	**VM1** $\delta = 0.02$	**VM2** $\delta_e = 0.07$	ℓ_1	**lof**	**VM1** $\delta = 0.11$	**VM2** $\delta = 0.25$	ℓ_1	**lof**	**VM1** $\delta = 0.05$	**VM2** $\delta_e = 1.28$	ℓ_1	**lof**
Wacht-R	100%	100%	0.122	1.00	100%	100%	0.084	1.00	92%	92%	0.023	-0.78	-	-	-	-
Proto-R	100%	96%	0.104	1.00	100%	100%	0.069	1.00	90%	88%	0.011	-0.02	32%	30%	0.300	-1.00
MILP-R	100%	100%	0.212	-0.48	100%	100%	0.059	1.00	100%	100%	0.018	-0.88	100%	100%	0.031	1.00
ROAR	82%	14%	0.078	0.95	88%	34%	0.074	1.00	82%	78%	0.031	-0.80	62%	60%	0.047	1.00
APΔS	100%	100%	0.072	1.00	100%	100%	0.042	1.00	100%	100%	0.009	0.44	100%	94%	0.028	1.00

Fig. 5.12: Comparison on the robustness of CFXs using five state-of-the-art methods and APΔS proposed in this work.

results show a rather intriguing picture: the **VM2** metric appears to be unaffected by retraining, and all CEs remain valid on the respective final models.

These results suggest that certification approaches based on worst-case reasoning may be too strict in practical scenarios. To further understand the implications of worst-case vs average-case reasoning, we adapted Algorithm 15 to use the certification procedure of Jiang et al., i.e., a MILP solver instead of a sampled-based approach, and compute the maximum provable δ^* for which the previously generated CEs are robust (Algorithm 17).

Algorithm 17: Provable Plausible Δ-Shift

Input: Model $\mathcal{M}_\theta$, CE x', α, R
Output: δ_{max}
$\delta_{init} \leftarrow 0.0001$
rate $\leftarrow$ `MILP`$(\mathcal{M}_\theta, x', \delta_{init})$
if rate $\neq 1$ **then**
    return 0 ▷ no robustness
$\delta \leftarrow \delta_{init}$
while rate = 1 **do**
    $\delta \leftarrow 2\delta$
    rate $\leftarrow$ `MILP`$(\mathcal{M}_\theta, x', \delta)$
$\delta_{max} \leftarrow \delta/2$
while True **do**
    if $|\delta - \delta_{max}| \leq \delta_{init}$ **then**
        return δ_{max}
    $\delta_{new} \leftarrow (\delta_{max} + \delta)/2$
    rate $\leftarrow$ `MILP`$(\mathcal{M}_\theta, x', \delta_{new})$
    if rate = 1 **then**
        $\delta_{max} \leftarrow \delta_{new}$
    else
        $\delta \leftarrow \delta_{new}$

Fig. 5.13 shows a comparison between the average maximum provable δ obtained by this procedure and `APΔS` . As we can observe, our average-case guarantees allow to obtain δ values that are much higher, exceeding the MILP-certified in all instances. This is expected, given the results discussed in Proposition 5.1. However, what remains unclear is how these differences may affect the cost and plausibility of CEs when certification procedures are embedded in procedures to generate CEs.

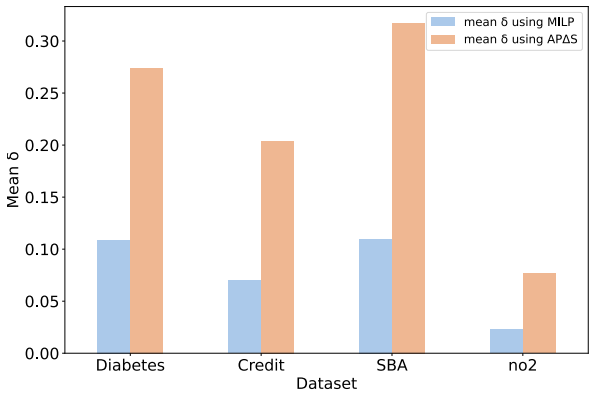


Fig. 5.13: Average robust δ obtained using MILP-based certification and `APΔS` .

Generating Robust CEs using APΔS

The results discussed in the previous section have important implications on algorithms for the generation of robust CEs. Recent works, e.g. [62, 116, 94], have proposed iterative procedures that generate provably robust CEs by alternating two phases. First, a CE is generated solving (variations of) Definition 2.5; then, a robustness certification procedure is invoked on the CE. If the CE is robust, then it is returned to the user; otherwise, the search continues, allowing for CEs of increasing distance to be found. Clearly, the certification step has the potential to affect the CEs computed in several ways. A robustness test that is too conservative may discard potentially good explanations and keep relaxing the distance constraint until the CE is deemed robust. Ultimately, this may result in CEs that exhibit poor proximity and plausibility.

To test this hypothesis, we adapt the CE generation algorithm of [116] and replace their Δ-robustness test with the one performed by APΔS . The complete procedure is shown in Algorithm 18. In detail, after some initialization steps, we compute the first CE using ComputeCE$(x, \mathcal{M})$, which employs the solution proposed in [116] and presented above. Given a CE x' and a plausible model shift Δ, at line 6, we employ APΔS setting $\alpha = 0.999$ and $R = 0.995$, thus obtaining 1378 realizations to perform in the robustness test. If the CE x' returned by our approximation results robust for all these realizations, then we return it to the user. Otherwise, we increased the allowed distance for the next CE generation and the iteration number t.

We then compare the resulting procedure with the four generation algorithm studied in [116]: Wacht-R, Proto-R, MILP-R, and finally, ROAR [262]. Notably, ROAR is specifically designed to generate robust CEs under plausible model changes using average-case certification. Using the same datasets and training procedures of SubSec. 5.1.4, we generate 50 CEs for each dataset. We evaluate CEs based on their proximity, measured by the ℓ_1 distance, and plausibility, measured by the local outlier factor (**lof**) which determines if an instance is within the data manifold by quantifying the local data density [34] (+1 for inliers, −1 otherwise). We average ℓ_1 and **lof** over the generated CEs. We also report **VM1** and **VM2** for completeness.

Algorithm 18: Generation of Robust CEs

1: **Input:** Model $\mathcal{M}$, input x such that $\mathcal{M}(x) = c$, set of plausible model changes Δ, maximum iteration number τ
2: **Output:** Δ-robust CE x'
3: $t \leftarrow 0$ ▷ iteration number
4: **while** $t < \tau$ **do**
5: $x' \leftarrow$ ComputeCE$(x, \mathcal{M})$
6: rate $\leftarrow$ APΔS $(\mathcal{M}, x', \Delta)$
7: **if** $rate = 1$ **then**
8: **return** x' ▷ x' is approx. Δ-robust
9: **else**
10: increase allowed distance of next CE
11: increase iteration number t
12: **return** no robust CE can be found

The results obtained, which we report in Table 3.25, confirm our hypothesis. Indeed, APΔS produces the best results across all datasets, always generating CEs with high plausibility and better proximity. Notably, APΔS outperforms ROAR as well, producing CEs that retain a higher degree of validity after retraining.

Impact of hyper-parameters on validity, plausibility and cost

The previous set of experiments demonstrated that APΔS is able to outperform existing approaches and generate CEs that are robust, but also plausible and less expensive than other robust approaches. What remains unclear is the role that the main hyperparameters of our algorithm, α and R, might play in obtaining these results. We therefore conducted additional experiments to evaluate the interplay between the tightness of the probabilistic guarantees offered by APΔS and the quality of resulting explanations. In particular, focusing on the same datasets used in previous experiments, we started by checking the influence that α and R have on the validity of CEs after retraining. We generated 50 CEs for each dataset using an instantiation of Algorithm 18 that uses MILP encodings to generate candidate CEs as done in [116]. For clarity, Figs. 5.14-5.17 report only the results obtained for the *Diabetes* and *SBA* datasets.

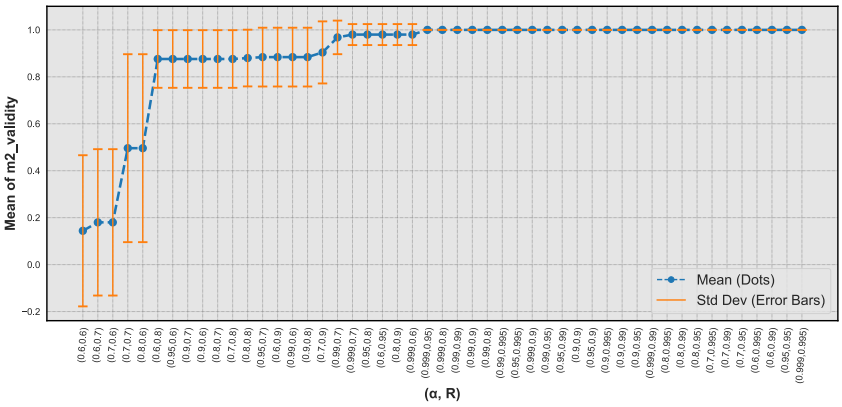


Fig. 5.14: Mean validity after retraining visualized for increasing α, R values using the *Diabetes* dataset.

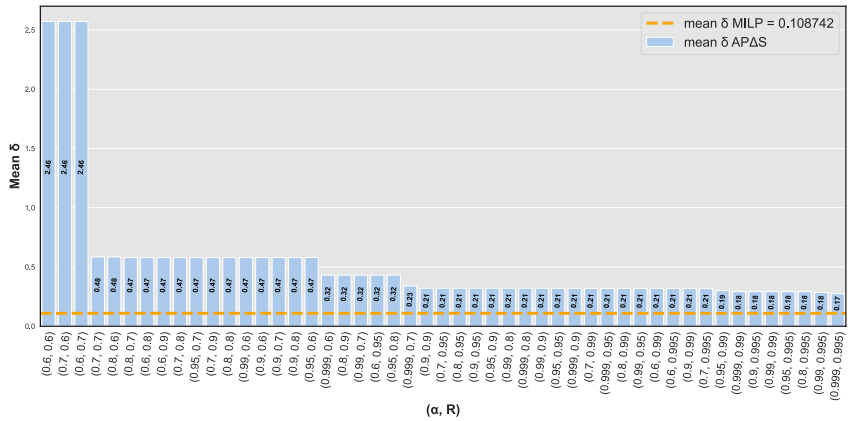


Fig. 5.15: Mean certifiable δ obtained for increasing α, R values using the *Diabetes* dataset.

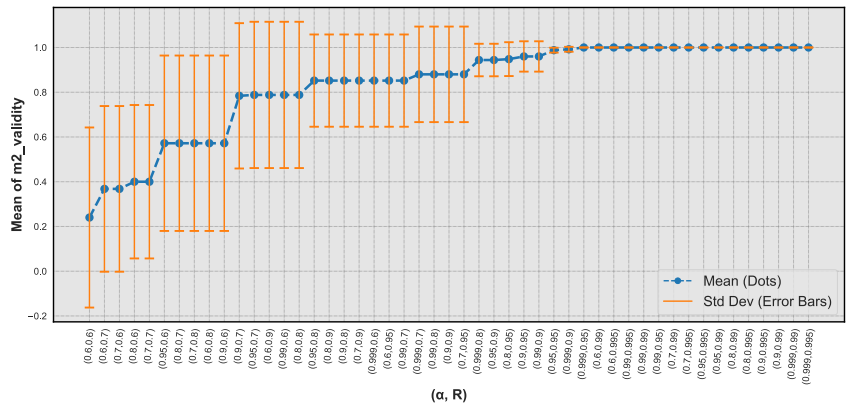


Fig. 5.16: Mean validity after retraining visualized for increasing α, R values using the *SBA* dataset.

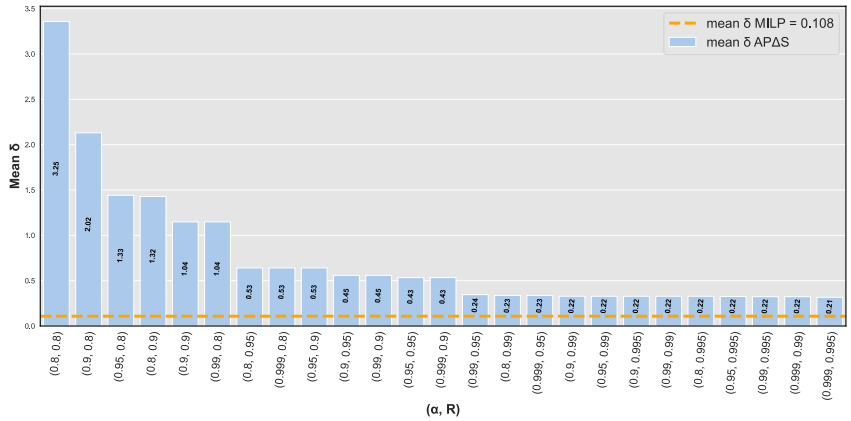


Fig. 5.17: Mean certifiable δ obtained for increasing α, R values using the *SBA* dataset.

Our intuition is that lower values for α and R should result in coarser robustness guarantees (i.e., larger δ values) and, thus, lower validity rates. As we can observe, our intuition is confirmed across all datasets, further clarifying the nature of the

probabilistic guarantees that APΔS can offer. Next, we investigate the impact that α and R have on the plausibility and cost of CEs generated by APΔS . As per our previous experiments, we measure plausibility using the *LOF* score, and we use ℓ_0, ℓ_1, and ℓ_∞ norms to measure the proximity of CEs. For conciseness, we only report results for the *Diabetes* dataset in Fig. 5.18 below. To improve the readability of our results, we decided to separate CEs that achieved 100% validity after retraining from the rest. Overall, we can observe a clear trend, whereby increasing α, R results in CEs that are further away from the decision boundary and thus more plausible. These results are in line with observations made in other works on robustness to model changes, where it has been suggested that increasing cost improves the robustness and plausibility of CEs [262, 209, 118].

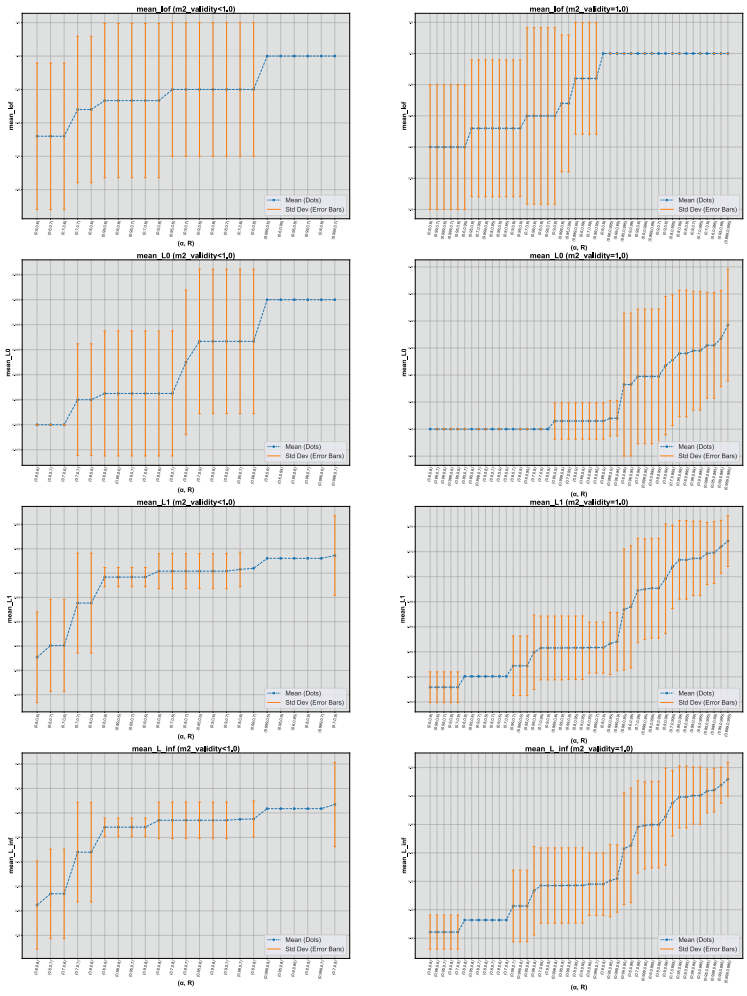


Fig. 5.18: Mean *LOF*, ℓ_0, ℓ_1, ℓ_∞ metrics for increasing α, R values using the *Diabetes* dataset. CEs with 100% validity after retraining are shown on the right, while the remaining CEs are shown on the left.

Scalability analysis of APΔS

In this section we demonstrate that APΔS is able to scale to state-of-the-art architectures used for tabular data, thus providing further empirical evidence of the practical viability of our approach. More specifically, we focus on *TabNet* [11], a tabular transformer recently introduced that leverages sparse attention and sequential feature selection to learn interpretable feature representations. At its core, *TabNet* processes data in a series of decision steps, with each step using a learned attention mask to select a subset of features, which allows the model to focus on the most relevant attributes at each stage. This sparse attention mechanism makes *TabNet* computationally efficient and helps to improve interpretability in tabular datasets. From our perspective, this architecture is interesting as it comprises an attention mechanism, with encoder-decoder components typical of other recent transformer architectures and a consequent significant number of parameters to test the scalability of APΔS . To the best of our knowledge, this is the first time that CEs with robustness guarantees are generated for such a complex architecture with tens of thousands of parameters.

Diabetes

	# Parameters	Mean Accuracy	Mean δ_{max}	Mean Comp. Time
MLP	81	79%	0.32	0.01s
TabNet	30992	82%	0.48	5.21s

no2

	# Parameters	Mean Accuracy	Mean δ_{max}	Mean Comp. Time
MLP	145	64%	0.11	0.008s
TabNet	30676	68%	0.35	9.2s

SBA

	# Parameters	Mean Accuracy	Mean δ_{max}	Mean Comp. Time
MLP	199	99%	0.53	0.02s
TabNet	30992	100%	0.16	10.3s

Credit

	# Parameters	Mean Accuracy	Mean δ_{max}	Mean Comp. Time
MLP	371	74%	0.34	0.01s
TabNet	40946	74%	1.8	11.3s

Table 5.2: Scalability experiments of APΔS .

Before considering the robustness property, we analyse the accuracy in the training and testing phases of *TabNet*. To this end, we employ a supervised training approach, splitting the datasets employed in the previous evaluation, namely *Diabetes*, *No2*, *SBA* and *Credit*, into training and testing datasets, and we first compare the accuracy obtained using this architecture with standard MLPs employed in Sub-Sec. 5.1.4. To ensure statistical significance of our results, we consider, for each dataset tested, the mean of the accuracies obtained using ten random initializations of the transformer architecture. As highlighted in the first two columns of Tab. 5.2,

with *TabNet*, we have an increased number of parameters in the model but similar or even higher accuracy with respect to the classical MLP, confirming the potential of this novel architecture in selecting important features to get more precise final accuracy in the prediction.

As our results show a similar level of accuracy between MLP and *TabNet*, we move on to how to generate robust CEs for this transformer architecture. Given the significantly higher number of parameters in *TabNet*, we replace the MILP-based procedure used in Sec. 5.1.4, Algorithm 18 with a Nearest Neighbors Counterfactual Explainer (NNCE) [89] to ensure scalability of our generation procedure. More specifically, line 5 in Algorithm 18 now implements the following strategy. Given an input x for which a robust CE is sought, we identify the nearest data point belonging to the dataset for which *TabNet* produces a different classification outcome. Our implementation uses k-d trees to improve the efficiency of this nearest-neighbor search. Given the significantly higher number of parameters in *TabNet*, we replace the MILP-based procedure used in Sec. 5.1.4, Algorithm 18 with a Nearest Neighbors Counterfactual Explainer (NNCE) [89] to ensure scalability of our generation procedure. More specifically, line 5 in Algorithm 18 now implements the following strategy.

Algorithm 19: *Nearest Neighbors Counterfactual Explanation*

1: **Input:** Dataset **d**, a k-d tree built from dataset features, **x** set of original inputs, **y** set of original outcomes
2: **Output: x'** set of nearest counterfactual explanation.
3: **x'** $\leftarrow \emptyset$
4: **for** i in $len(\mathbf{x})$ **do**
5: $x \leftarrow \mathbf{x}[i]$ ▷ original input
6: $y \leftarrow \mathbf{y}[i]$ ▷ original output
7: $y' \leftarrow 1 - y$ ▷ desired outcome
8: $idx, distance \leftarrow$ `k-d tree.query`$(x, len(features))$ ▷ already sorted per distance
9: **if** $\mathbf{d}[idx]['outcome'] == y'$ **then**
10: **x'** $\leftarrow \mathbf{d}[idx]$
11: **else**
12: **x'** $\leftarrow None$
13: **return x'**

This function iteratively searches for a neighboring data point with the opposite outcome by evaluating distances between features and selecting the nearest as possible. Clearly, this approach and the one of [116] can produce different explanations. We perform a further experiment to understand the difference between the two CE-generation approaches. Hence, we consider the same datasets used in Sec. 5.1.4 and the same 50 original inputs employed in those experiments. In the NNCE approach, once a valid counterfactual is found, the mean feature-wise distance between the identified counterfactual and the MILP-generated counterfactual is calculated. This distance serves as a measure of similarity between the counterfactuals identified by the TabNet-based approach and those obtained via MILP in an MLP setting. Our

results are reported in Fig. 5.19. On the x-axis, we report the index of CE, while on the y-axis, the mean and standard deviation distance between our CE and the one generated with MILP in each dataset.

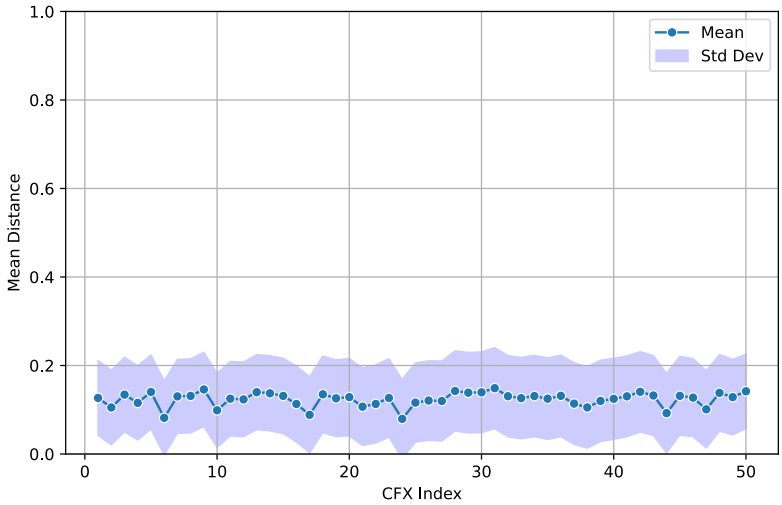


Fig. 5.19: Mean and standard deviation distance between CEs generated with NNCE and MILP in *Diabetes, Credit, SBA, no2* datasets.

As we can notice, since the values in the datasets are typically normalized in a range $[0, 1]$, the CEs generated with the two approaches are consistently close. In fact, there is a mean feature distance between the CE generated with NNCE and MILP of ~ 0.12 for the 50 inputs selected. This result shows the correctness and efficiency of the NNCE generation approach in the transformers-based setting.

For the following experiments, we consider the same datasets used in Sec. 5.1.4 and the same 50 original inputs employed in those experiments. The last two columns of Table 5.2 report the results we obtained when generating CE with robustness guarantees for *TabNet*. As we can observe, APΔS is still able to compute robust CEs within tens of seconds, even when employed in transformer-based architecture with $\sim 400x$ times parameters, showing a linear growth time computation. Similar runtimes are observed across all four datasets, thus confirming the high scalability of our approach. To further confirm this aspect, we ran an in-depth scalability study by training a set of 4 *TabNet* models with an increasing number of parameters. Using the diabetes dataset, we trained models containing $[30834, 40946, 62578, 126066]$ respectively and generated 50 robust CEs for each. We stored the runtimes for each robust CE and report the mean computation time for each model in Fig. 5.20. As we can observe, the runtime increase follows a linear trend, thus highlighting the effectiveness and applicability of our proposed solution even when targetting complex architectures.

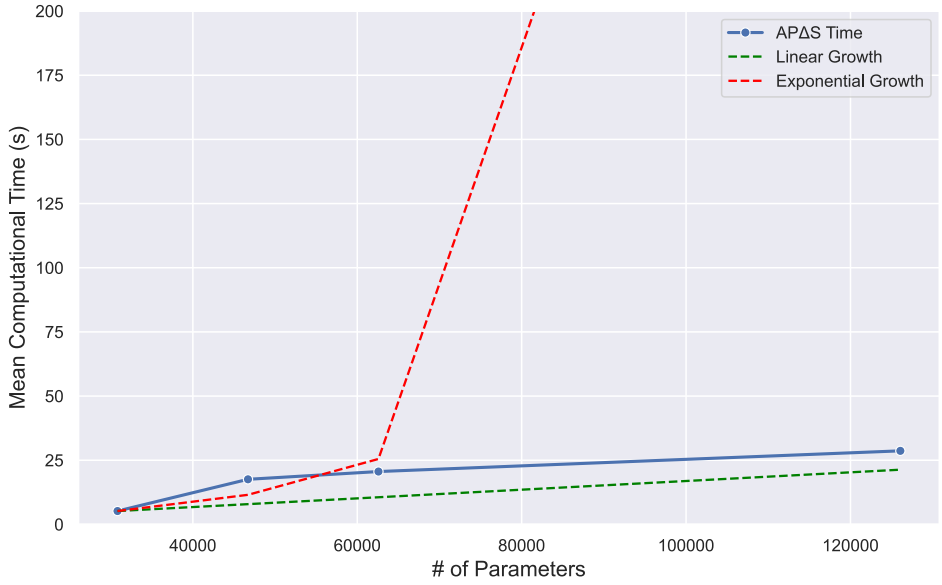


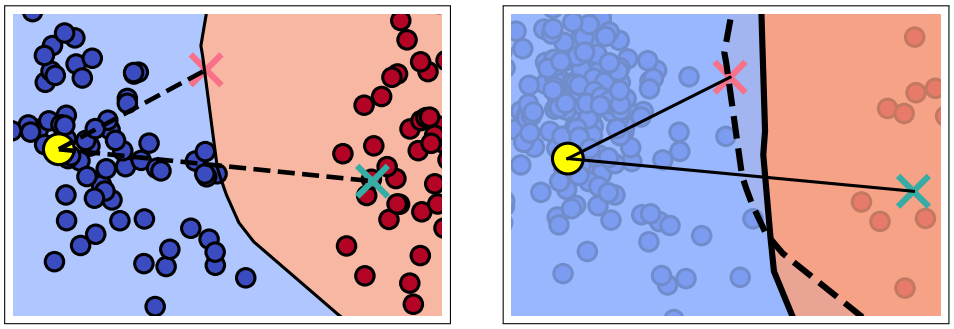
Fig. 5.20: Mean computation time of APΔS applied to *TabNet* architectures with increasing number of parameters.

Summary. In this section, we studied the problem of generating robust CEs with respect to plausible model changes. We proved for the first time that certifying the robustness of CE with respect to this notion of robustness is an NP-hard problem, and also extended this result to show that the same complexity results apply to naturally-occurring model changes. These results motivate the quest for new scalable algorithms to certify robustness under plausible model changes. To this end, we investigated existing methods to generate robust CEs with probabilistic guarantees and showed that these approaches may not be directly applicable to our setting. We then introduced APΔS , a novel scalable approach for probabilistic robustness certification, and used it to generate robust CEs under plausible model changes. We carried out an extensive experimental analysis, demonstrating the advantages of APΔS and outperforming SOTA methods on a range of metrics, including validity, plausibility, and cost. Crucially, we also applied our method to certify CEs' robustness for tabular transformers containing thousands of parameters. To the best of our knowledge, we are the first to consider models of this size within the robust CE literature [118], further demonstrating the scalability and wide applicability of our approach. We see these outcomes as important contributions towards complementing existing formal approaches for Explainable AI and making them applicable in practice.

5.2 RobustX: Robust Counterfactual Explanations Made Easy

With the increasing use of Machine Learning (ML) models to aid decision-making in high-stakes fields such as healthcare [233] and finance [40], there is a growing need for better explainability of these models. Counterfactual Explanations [89] are often leveraged in Explainable AI (XAI) to this end due to their intelligibility and alignment with human reasoning [193, 39]. In particular, CEs can offer insights into the predictions produced by an ML model by showing how small changes in its input may lead to different (often more desirable) outcomes. To see what benefits CEs can bring, consider an illustration of a loan application with features *27* years of age, *low* credit rating, and *15K* loan amount. Assume a bank's ML model classifies the application as not creditworthy. A CE for this outcome could be an altered input where a *medium* credit rating (with the other features unchanged) would result in the application being classified as creditworthy, thus giving the applicant an idea of what is required to have their loan approved.

Despite their potential, current approaches to generating CEs often fall short in producing *robust explanations*. Consequently, these methods may produce explanations whose validity is compromised by slight changes in the scenario being explained. For instance, recent work [262, 116, 94] has highlighted that even small alterations in the parameters of an ML model, e.g., following fine-tuning, may invalidate previously generated CEs. An example of this scenario is captured in Fig. 5.21, where a lack of robustness is demonstrated on a model trained for binary classification tasks. In Fig. 5.21a, an input (yellow circle) receives an initial classification (blue class), and two counterfactuals are generated for it: one (red cross) laying exactly on the decision boundary (full black line) and one deeper inside the counterfactual class (green cross). In Fig. 5.21b, we observe that the decision boundary of the model undergoes slight changes, induced by fine-tuning on a slightly shifted input distribution. As a result, previously generated CEs may cease to be valid if no precautions are taken to ensure robustness. For instance, we observe that the



(a) (b)

Fig. 5.21: A lack of robustness may invalidate CEs, here demonstrated on a neural network classifier trained to solve a binary classification task. An input (yellow circle) receives an initial classification, and two counterfactuals (red and green crosses) are generated for it (Fig. 5.21a). After a fine-tuning step occurs (Fig. 5.21b), the decision boundary slightly changes (from dashed to full black line), and previously generated CEs may be invalidated (red cross).

CE corresponding to the red cross is now classified as belonging to the blue class and is thus invalid. Now consider the consequences of these behaviors in our loan example: after fine-tuning, the applicant changing their credit rating to medium no longer ensures the success of the loan application, as the CE was not robust. When this happens, the CE previously generated by the bank is invalidated, and the bank may be liable for inconsistent statements made to customers regarding loan terms.

This and many other forms of robustness of CEs have recently been the subject of intense research efforts, and numerous algorithms to evaluate the robustness of CEs have been proposed (a recent survey identified about 40 methods [120]). However, the current state of robust CE research is fragmented, with various methods developed independently and implemented in different, often incompatible, ways. This lack of standardization has resulted in challenges for the broader research community, as comparing the effectiveness of robust CE generation methods is impractical.

We fill this gap in this section and introduce `RobustX`, an open-source Python library to standardise and streamline the generation, evaluation, and benchmarking of robust CEs. `RobustX` provides flexible, extensible, and customisable tools to implement custom CE methods. Differently from existing frameworks, e.g. [208, 3], our library focuses on providing a consistent framework for testing robustness and systematically comparing various methods, ensuring fair and reliable evaluations. `RobustX` addresses key limitations on library tools in the current landscape ([133, 120]) by offering a standardised approach to robust CE development while also promoting extensibility, allowing users to integrate new datasets and explanation algorithms as needed. The library, including documentation and tutorials, is publicly available at the following link:

https://github.com/RobustCounterfactualX/RobustX

The reminder of this section is organised as follows. Sec. 5.2.1 presents the main components of the library, providing details about their functionalities. Sec. 5.2.2 demonstrates how easy it is to use `RobustX` to benchmark existing CE generation algorithms and compare them using different metrics. Finally, we offer some concluding remarks and pointers for future work.

5.2.1 Overview

RobustX implements a complete pipeline for robust CE generation and evaluation (Fig. 5.22) with three major components: **Task**, **CE generator**, and **CE evaluator**. Each has an abstract class template for easy customization. Users start by creating a task, which defines the model and inputs to be explained. Then, the user can choose whether to use `RobustX` to generate CEs, or directly evaluate the robustness of previously (externally) generated CEs.

Task objects, providing functionality for interactions between models and datasets, are the basic class passed into the CE generation and evaluation pipelines. Our current implementation assumes a `ClassificationTask` by default, as this is the most commonly considered use case in the literature; however, users can also implement customised `Task` objects for learning problems other than classification. `RobustX` natively supports models trained using sklearn [210], Keras [46]

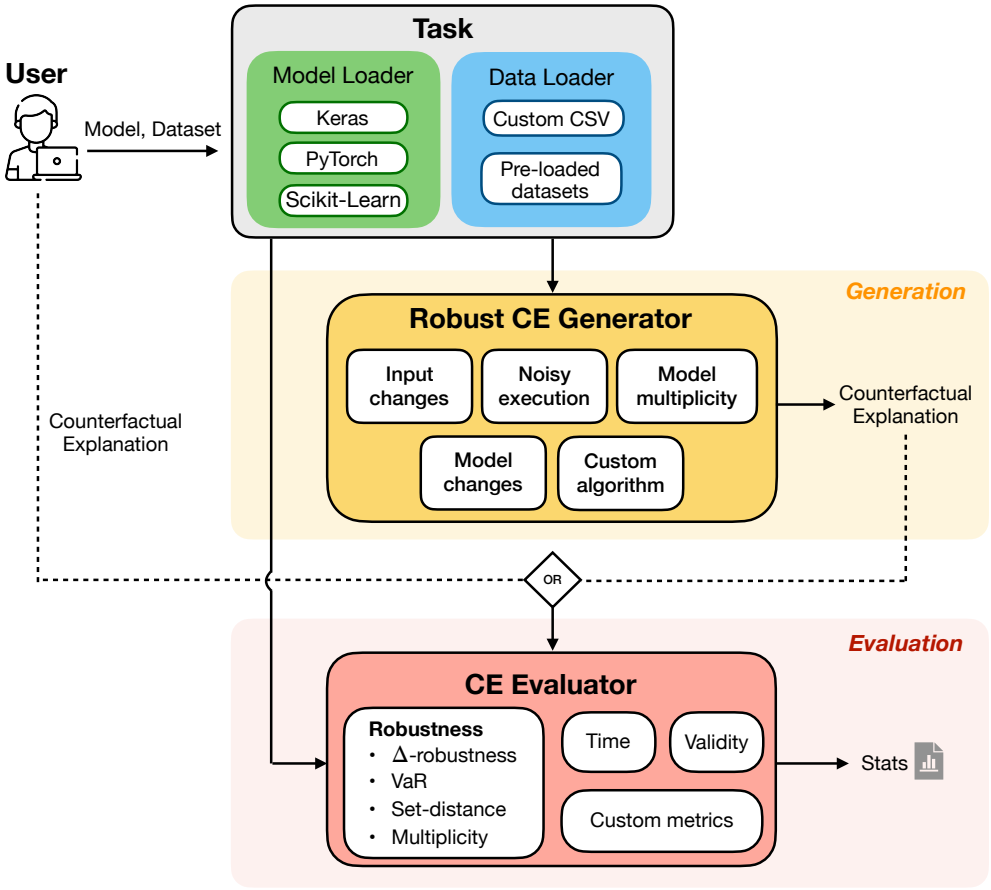


Fig. 5.22: Internal work-flow of `RobustX`. Users can choose whether to generate robust CEs using our library or evaluate the robustness of externally generated CEs in our `RobustX`'s evaluation facilities.

and PyTorch [204]. Models trained using other frameworks can also be used by instantiating a `BaseModel` wrapper class. As far as datasets are concerned, `RobustX` offers a selection of pre-loaded example datasets that can be readily loaded using the `DatasetLoader` class. Additionally, this class also allows the uploading of custom datasets if needed via *.csv* files.

`RobustX` currently implements nine **robust CE generation methods** across different robustness use cases reported in the yellow box in Fig. 5.22: `APΔS` [177], `ArgEnsembling` [117], `DiverseRobustCE` [143], `MCER` [116], `ModelMultiplicityMILP` [144], `PROPLACE` [115], `RNCE` [119], `ROAR` [262], `STCE` [62, 94]. It also provides four popular non-robust methods that can be used as baselines to crease new generation methods: `BLS` [143], `MCE` [195], `KDTreeNNCE` [36], and the seminal work by [271]. All methods inherit from the abstract class `CEGenerator` and implement the `_generation_method()` function, providing an easy interface to other components in the pipeline.

CE evaluation methods can take in CEs, either generated within or outside `RobustX`, and benchmark their robustness along with other common properties identified in the literature. Currently, `RobustX` provides a `CEEvaluator` class to

```python
# first prepare a task
from robustx.datasets.ExampleDatasets import get_example_dataset
from robustx.lib.models.pytorch_models.SimpleNNModel import SimpleNNModel
from robustx.lib.tasks.ClassificationTask import ClassificationTask

data = get_example_dataset("ionosphere")
data.default_preprocess()
model = SimpleNNModel(34, [8], 1)
model.train(data.X, data.y)
task = ClassificationTask(model, data)

# specify the names of the methods and evaluations we want to use, run benchmarking
# This will find CEs for all instances predicted with the undesirable class (0) and compare
from robustx.lib.DefaultBenchmark import default_benchmark

methods = ["KDTreeNNCE", "MCE", "MCER", "RNCE", "STCE", "PROPLACE"]
evaluations = ["Validity", "Distance", "Delta-robustness"]
default_benchmark(task, methods, evaluations, neg_value=0, column_name="target", delta
    =0.005)
```

Fig. 5.23: Code snippet exemplifying how `RobustX` can be used to easily benchmark CE methods. Table 3.25 lists the benchmarking results.

evaluate the validity and proximity of CEs [271], as well as five classes specifically focusing on robustness evaluation metrics: `VaRRobustnessEvaluator` to assess the validity of CEs after retraining [62], `DeltaRobustnessEvaluator` [116] to assess the robustness of CEs under plausible model changes, `ApproximateDeltaRobustnessEvaluator` [177] assessing probabilistic robustness to plausible model changes, `SetDistanceRobustnessEvaluator` [143] for assessing stability of CEs when the input is perturbed, and `MultiplicityValidityRobustnessEvaluator` [144] for checking CE robustness under model multiplicity. Additional robustness evaluators can be easily added through the extensible interface provided by `RobustX`.

5.2.2 RobustX in Action

In this subsection we provide an example on how to use `RobustX` in practice; additional examples are available online. Fig. 5.23 shows how to run and compare six methods supported by `RobustX`. In this example we focus on robustness against model changes [262] and perform a comparison between four robust methods and two non-robust baselines. To this end, we first import the (pre-loaded) `ionosphere` dataset for binary classification and apply standard pre-processing. We then create and train a simple three-layer neural network model. A task object is then created from the dataset and model. Then, we specify the CE generation and evaluation methods of interest and run the benchmarking procedure. The default benchmark function in this example runs each method with its default hyperparameters, although customised hyperparameters can be configured. It then generates CEs for all instances in the dataset which are predicted with an undesirable class (here 102 points with neg_value=0), and runs the specified evaluation methods. In this example, we evaluate CEs along three metrics: validity, proximity [271] and Δ-robustness [116]. The results along the selected evaluation metrics are then printed in a structured table, so that we can easily compare how each method performs.

Table 5.3 shows the results obtained, and reports the computation time, the percentage of valid CEs, average proximity (L2 distance to the input), and the percentage of CEs that are Δ-robust. Observing this table, users of `RobustX` will

be able to identify that robustness performance improvements from the non-robust baselines (NNCE, MCE) to the robust methods (MCER, RNCE, STCE, PROPLACE) are notable, although MCER fails to achieve 100% robustness. Based on these results, users might conclude that `RNCE` is the optimal CE generator for this task, balancing between computation time, proximity, and robustness.

Method	**Time (s)**	**Validity (%)**	**Proximity**	**Rob. (%)**
KDTreeNNCE	0.2	100	5.76	51.6
MCE	3.4	100	2.95	0
MCER	137.6	100	4.84	64.8
RNCE	3.9	100	6.03	100
STCE	39.7	100	7.30	100
PROPLACE	12.9	100	6.02	100

Table 5.3: Example benchmarking of six CE generation methods.

Summary. In this last section, we introduced `RobustX`, a Python framework to generate, evaluate, and compare robust CE for ML models. Our library fills a major gap in the existing literature on robust CEs, providing an easy-to-use and extensible platform to benchmark existing algorithms for robust CEs. Building upon extensive research in the area, `RobustX` provides a unified platform to run and compare existing approaches, as well as implement new ones, reducing the need to re-implement software from scratch. Work is underway to further expand the list of available generation algorithms, evaluation methods, and additional software facilities for testing and validation to ensure the correctness of the implementations. We believe `RobustX` will streamline research efforts in robust CEs, accelerating the development of innovative solutions and fostering collaboration within this rapidly growing field.

Chapter 6

Conclusions and Open Challenges

"Science is not only a disciple of reason but also one of romance and passion."

– Stephen Hawking

In this final chapter, we draw the conclusions, summarizing the contribution of this thesis, and discuss open challenges and possible future directions.

In this work, we started addressing the following open research questions:
(RQ.1) *How can we enhance scalability and expressivity of formal verification techniques to larger and more realistic neural networks while maintaining safety guarantees?*
(RQ.2) *How can verification be incorporated into the DRL training loop, and how can we move beyond simple, hand-designed safety properties to automatically define meaningful behavioral preferences for complex robotic tasks?*
(RQ.3) *What role can verification methods play in enabling explainability, e.g., via counterfactual reasoning, in complex decision-making systems?*

Specifically, in Chapter 3, we presented a set of methods to address **RQ.1** and part of **RQ.3**. We introduced the Abstract DNN-Verification, extending the standard formal verification of neural networks to include a hierarchical structure of safety and robustness properties. By allowing multiple levels of output abstraction, our approach addresses limitations in traditional verification methods, which rely on binary classifications of safe or unsafe outputs. This enhanced framework enables a scalable and more expressive analysis of deep neural networks, especially in complex scenarios where traditional safety properties are hard to write or to verify. Nonetheless, the proposed approach is still based on provable over-approximations, which can inherently suffer from scalability limitations, as evidenced by the NP-hardness of the problem. Thus, we introduce a novel probabilistic perspective on the robustness verification of deep neural networks, aiming to balance tractability and reliability in high-dimensional settings. Although this novel solution provides promising results in terms of scalability, it focuses on standard verification, which may not be enough to fully understand the safety level of a model. Hence, we extended the robustness verification from a Boolean decision problem to a quantitative

measure of the unsafe input space, introducing the #DNN-VERIFICATION problem. We analyzed the complexity of this problem, proving the #P-hardness and highlighting why it is relevant for the community. Furthermore, we proposed an exact count approach that, however, inherits the limitations of the formal verification tool exploited as a backend and struggles to scale on real-world problems. Crucially, we presented an alternative probabilistic approach, `CountingProVe`, which provides an approximated solution with formal guarantees on the confidence interval. Furthermore, to provide information on the actual input configurations' locations that are safe or violate the property of interest, we introduced the ALLDNN-VERIFICATION problem. Due to the #P-hardness of the problem, we presented a novel approximation approach, ϵ-`ProVe`, which is able to efficiently approximate the safe regions with strong probabilistic guarantees on the tightness of the solution returned. However, the reliance on a single decision tree to provide the solution often results in the generation of a large number of regions, which in complex scenarios can even exhaust memory resources, producing highly fragmented representations that are difficult to interpret and impractical for downstream tasks such as safe recovery or explanation. To address this issue, we introduced `RF-ProVe`, exploiting the potential of bootstrap-based and randomized approaches capable of capturing complex patterns in high-dimensional spaces, including input regions where a given output property holds. Finally, we presented `ModelVerification.jl`, a comprehensive toolbox for verifying deep neural networks that incorporates several of the solutions discussed in this thesis.

To address **RQ.2**, in Chapter 4, we first presented a way to exploit formal verification of deep neural networks in challenges associated with the deployment of DRL approaches in safety-critical scenarios such as autonomous colonoscopy navigation in a virtual simulation. Specifically, we proposed a novel model selection strategy that exploits FV of deep neural networks to evaluate the safety of a vast pool of policies trained using a constrained reinforcement learning approach that ensures soft safety constraints through a cost function of safety violations. However, this solution is actionable only after the training of the DRL policies, as the FV of DNNs is computationally demanding to perform online during training. To address this issue, we introduced an unconstrained DRL framework that leverages local violations of input-output conditions to foster safety. In detail, we discussed the limitations of using cost functions as in safe DRL, presenting the CROP framework that allows the collection and refinement of safety properties during training, thus overcoming the limitations of hand-designed approaches and a sample-based approach to approximate a *violation* metric (i.e., the result of the #DNN-VERIFICATION) and use it as a penalty in DRL algorithms. We provided provable probabilistic guarantees on the correctness of our approximation, showing that such a violation introduces task-level safety specifications into the optimization, addressing the potential lack of safety information of cost-based approaches. We empirically demonstrated that our solution allows us to obtain a more robust approach with respect to other safe DRL methodologies, promoting safer behaviors while maintaining similar or better returns. Additionally, instead of solely using these unsafe regions to collect properties

and compute penalties, we investigated whether, motivated by the way humans improve through repeated practice, retraining the agent specifically on these challenging regions can enhance both its sample efficiency and its safety. Hence, we introduced ε-retrain, a novel exploration strategy with monotonic improvement guarantees that optimizes policies while encouraging specific behavioral preferences. Our empirical and formal evaluation over hundreds of seeds considering various tasks and behavioral preferences, demonstrated the effectiveness in terms of higher sample efficiency and superior performance of ε-retrain when integrated with existing policy optimization methods.

Nonetheless, both ε-retrain and the previously introduced methods still require the use of a formal verification tool after training, as their safety guarantees hold only in expectation. To address this limitation, we presented a novel combination of probabilistic enumeration of unsafe regions (ε-`ProVe` of Chapter 3) to construct a safe set for a control barrier function. Such a mechanism, deployed at run time, enforces provably safe behaviors throughout the agent's operation. Unlike traditional methods that rely heavily on domain-specific knowledge or incur high computational costs, our framework is agnostic to the deployment environment and operates independently of the DRL training process, enhancing its general applicability. Through extensive evaluations, we demonstrated that our approach achieves robust safety guarantees with zero violations of safety constraints. At the same time, it preserves effective navigation behaviors across different robot platforms, including an indoor robot ground and an aquatic drone.

Finally, to address **RQ.3**, in Chapter 5, we studied the problem of generating robust CFXs with respect to plausible model changes. We proved for the first time that certifying the robustness of CFX with respect to this notion of robustness is an NP-hard problem, and also extended this result to show that the same complexity results apply to *naturally-occurring* model changes. These results motivate the quest for new scalable algorithms to certify robustness under plausible model changes. To this end, we investigated existing methods to generate robust CFXs with probabilistic guarantees and showed that these approaches may not be directly applicable to our setting. We then introduced `AP`Δ`S` a novel scalable approach for probabilistic robustness certification, and used it to generate robust CFXs under plausible model changes. We carried out an extensive experimental analysis, even considering tabular transformers containing thousands of parameters, demonstrating the advantages of `AP`Δ`S` and outperforming SOTA methods on a range of metrics, including validity, plausibility, and cost. We then concluded the chapter presenting `RobustX`, a Python framework to generate, evaluate, and compare robust CE for ML models. Our library fills a major gap in the existing literature on robust CEs, providing an easy-to-use and extensible platform to benchmark existing algorithms for robust CEs. Building upon extensive research in the area, `RobustX` provides a unified platform to run and compare existing approaches, as well as implement new ones, reducing the need to re-implement software from scratch. These outcomes are an important contribution towards complementing existing formal approaches for Explainable AI and making them applicable in practice.

Open Challenges

Despite the progress achieved in this thesis, several open challenges remain, pointing towards promising avenues for future research direction, summarized in Fig. 6.1.

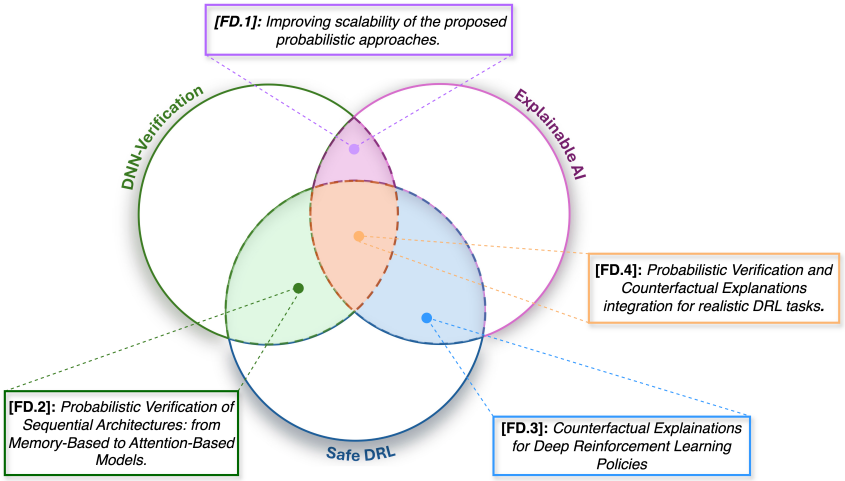


Fig. 6.1: Schematic overview of possible future directions for each intersection of the three main areas discussed in this thesis.

- **[FD.1]: Investigating the scalability of Probabilistic Verification**. Formal verification is inherently hard and struggles with high-dimensional models and real-world tasks. While the proposed frameworks address important limitations of traditional verification techniques, scalability remains a central challenge due to the curse of dimensionality. As the dimensionality of the input space and the complexity of the network increase, achieving high confidence and low error bounds may require a significant number of samples. This can still make the verification process through the proposed techniques, even if probabilistic, computationally demanding, particularly for deep or high-dimensional models. Exploring alternative probabilistic frameworks, such as those inspired by probably approximately correct (PAC) [266] learning theory, to reduce the sample complexity while maintaining meaningful statistical guarantees, is an interesting future direction.
- **[FD.2]: Extending Probabilistic Verification Beyond Feedforward Models.** Current probabilistic verification techniques presented in this thesis for DRL tasks primarily focus on memoryless, i.e., feedforward architectures. However, sequential memory-based models such as recurrent neural networks [188], long short-term memories (LSTMs) [102], or attention-based mechanisms Transformers [269], are increasingly used in robotics and decision-making used in robotics and decision-making. Extending verification techniques for these architectures, such as model unrolling [5], policy extraction [42], invariant inference [111], (backward) reachability analysis [67, 255] mainly target robustness verification. Those that address RL control face the problem of defining meaningful safety specifications, as well as an exponential scalability barrier for even simpler behavioral preferences certification. On the other hand, Transformer verification

is still almost unexplored, focusing only on robustness verification for toy models [234]. New methods that combine probabilistic reasoning with memory- or attention-based architectures represent a critical direction. Our results on tabular transformers presented in Chapter 5, represent a promising direction and pave the way for interesting future research.

- **[FD.3]: Counterfactual Explanations for DRL policies.** In Chapter 5 we have shown that counterfactual reasoning is effective in tabular and classification settings, yet its use for explaining complex sequential decision-making — such as DRL policies — is still largely unexplored. Recent work [75] discusses the use of CEs in reinforcement learning settings, however, future work should investigate how counterfactuals can be used not only to explain policies but also to guide safer exploration and retraining. Extending robust counterfactual methods to sequential domains is a challenging research direction.
- **[FD.4]: Probabilistic verification and counterfactual explanations integration for realistic DRL tasks.** While verification can certify that a model behaves safely under specific conditions, it does not provide insights into why a decision was taken, nor how it could have been different under slightly altered circumstances. In this thesis, we have shown how CEs provide this human-centric reasoning by highlighting minimal changes to inputs that would lead to alternative outcomes. However, CE methods suffer from two critical limitations: (i) they are often heuristic and lack guarantees, and (ii) they struggle with scalability in high-dimensional sequential models. Future directions should investigate novel integration of verification and counterfactual reasoning, similar to what we proposed in Chapter 5. For example, probabilistic verification methods can be used to identify critical decision boundaries in sequential models, while counterfactual explanations will be generated to communicate these boundaries in an interpretable manner. This integration will produce counterfactuals that are not only meaningful to end-users but also backed by probabilistic guarantees of robustness. For instance, in reinforcement learning applied to challeging power grid operations such as the ANM task of Sec. 4.3, the combination of verification and CE reasoning will enable explanations such as: “If the energy demand had been 5% higher at this timestep, with a 99% confidence, the policy would have opted for storage discharge rather than grid draw”.

In conclusion, the solutions related to probabilistic verification and counterfactual explanations developed in this thesis represent a first step for the trustworthy deployment of intelligent systems in real-world applications. However, the growing demand for autonomous systems operating in uncertain and dynamic environments, and in close collaboration with humans, introduces new problems in terms of safety, reliability, and interpretability. We believe, addressing these challenges requires a novel unified probabilistic framework that rigorously quantifies the safety aspect of intelligent systems while providing human-understandable explanations of their decision-making processes. Advancing in this direction will be crucial to bridge the gap between theory and practice, fostering the development of intelligent systems that are not only effective but also transparent, reliable, and aligned with human values.

List of Figures

List of Tables

References

[1] D. Abel, W. Dabney, A. Harutyunyan, M. K. Ho, M. L. Littman, D. Precup, and S. Singh. On the expressivity of markov reward. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[2] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017.

[3] C. Agarwal, S. Krishna, E. Saxena, M. Pawelczyk, N. Johnson, I. Puri, M. Zitnik, and H. Lakkaraju. Openxai: Towards a transparent evaluation of model explanations. In *NeurIPS*, 2022.

[4] F. Airaldi, B. D. Schutter, and A. Dabiri. Learning safety in model-based reinforcement learning using mpc and gaussian processes. *IFAC*, 56(2):5759–5764, 2023.

[5] M. E. Akintunde, A. Kevorchian, A. Lomuscio, and E. Pirovano. Verification of rnn-based neural agent-environment systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6006–6013, 2019.

[6] E. Altman. Constrained markov decision processes. In *CRC Press*, 1999.

[7] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2016.

[8] G. Amir, D. Corsi, R. Yerushalmi, L. Marzari, D. Harel, A. Farinelli, and G. Katz. Verifying learning-based robotic navigation systems. In *29th International Conference, TACAS 2023*, pages 607–627. Springer, 2023.

[9] D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano, J. Schulman, and D. Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

[10] F. Archetti and F. Schoen. A survey on the global optimization problem: general theory and computational approaches. *Annals of Operations Research*, 1:87–110, 1984.

[11] S. Ö. Arik and T. Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6679–6687, 2021.

[12] A. Artelt, V. Vaquet, R. Velioglu, F. Hinder, J. Brinkrolf, M. Schilling, and B. Hammer. Evaluating robustness of counterfactual explanations. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–09. IEEE, 2021.

[13] A. Athavale, E. Bartocci, M. Christakis, M. Maffei, D. Nickovic, and G. Weissenbacher. Verifying global two-safety properties in neural networks with con-

fidence. In A. Gurfinkel and V. Ganesh, editors, *Proc. of CAV 2024: the 36th International Conference on Computer Aided Verification*, volume 14682 of *LNCS*, pages 329–351. Springer, 2024. doi: 10.1007/978-3-031-65630-9_17.

[14] A. A. Aydeniz, E. Marchesini, C. Amato, and K. Tumer. Entropy seeking constrained multiagent reinforcement learning. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, page 2141–2143, 2024.

[15] A. A. Aydeniz, E. Marchesini, R. Loftin, C. Amato, and K. Tumer. Safe multiagent coordination via entropic exploration, 2024. URL `https://arxiv.org/abs/2412.20361`.

[16] M. Balunovic, M. Baader, G. Singh, T. Gehr, and M. Vechev. Certifying geometric robustness of neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[17] T. Baluta, S. Shen, S. Shinde, K. S. Meel, and P. Saxena. Quantitative verification of neural networks and its security applications. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[18] C. Barrett and C. Tinelli. Satisfiability modulo theories. In *Handbook of model checking*, pages 305–343. Springer, 2018.

[19] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi. Measuring neural net robustness with constraints. *Advances in neural information processing systems*, 29, 2016.

[20] L. Berrada, S. Dathathri, K. Dvijotham, R. Stanforth, R. R. Bunel, J. Uesato, S. Gowal, and M. P. Kumar. Make sure you're unsure: A framework for verifying probabilistic specifications. *Advances in Neural Information Processing Systems*, 34:11136–11147, 2021.

[21] G. Best, O. M. Cliff, T. Patten, R. R. Mettu, and R. Fitch. Dec-mcts: Decentralized planning for multi-robot active perception. *The International Journal of Robotics Research*, 38(2-3):316–337, 2019.

[22] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[23] F. Bianchi, D. Corsi, L. Marzari, D. Meli, F. Trotti, M. Zuccotto, A. Castellini, and A. Farinelli. Safe and efficient reinforcement learning for environmental monitoring. In *Proceedings of the Italia Intelligenza Artificiale - Thematic Workshops co-located with the 3rd CINI National Lab AIIS Conference on Artificial Intelligence (Ital IA 2023), Pisa, Italy, May 29-30, 2023*, volume 3486 of *CEUR Workshop Proceedings*, pages 610–615. CEUR-WS.org, 2023. URL `https://ceur-ws.org/Vol-3486/18.pdf`.

[24] F. Bianchi, A. Castellini, A. Farinelli, L. Marzari, D. Meli, F. Trotti, and C. Veronese. Developing safe and explainable autonomous agents: from simulation to the real world. In *Proceedings of the Ital-IA Intelligenza Artificiale - Thematic Workshops co-located with the 4th CINI National Lab AIIS Conference on Artificial Intelligence (Ital-IA 2024), Naples, Italy, May 29-30, 2024*, volume 3762 of *CEUR Workshop Proceedings*, pages 129–134. CEUR-WS.org, 2024. URL `https://ceur-ws.org/Vol-3762/547.pdf`.

[25] Y. Biktairov and J. Deshmukh. Sol: Sampling-based optimal linear bounding of arbitrary scalar functions. *Advances in Neural Information Processing Systems*, 36:33161–33173, 2023.

[26] A. Björklund, M. Zaitsev, and M. Kwiatkowska. Efficient preimage approximation for neural network certification. *arXiv preprint arXiv:2505.22798*, 2025.

[27] E. Black, Z. Wang, and M. Fredrikson. Consistent counterfactuals for deep models. In *Proceedings of the 10th International Conference on Learning Representations (ICLR22)*. OpenReview.net, 2022.

[28] A. Blum and R. L. Rivest. Training a 3-node neural network is np-complete. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1, [NIPS Conference, Denver, Colorado, USA, 1988]*, pages 494–501. Morgan Kaufmann, 1988. URL `http://papers.nips.cc/paper/125-training-a-3-node-neural-network-is-np-complete`.

[29] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[30] D. Boetius, S. Leue, and T. Sutter. Probabilistic verification of neural networks using branch and bound. *International Conference on Machine Learning (ICML)*, 2024.

[31] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener. Efficient verification of relu-based neural networks via dependency analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3291–3299, 2020.

[32] C. E. Boyd, L. R. D'Abramo, B. D. Glencross, D. C. Huyben, L. M. Juarez, G. S. Lockwood, A. A. McNevin, A. G. Tacon, F. Teletchea, J. R. Tomasso Jr, et al. Achieving sustainable aquaculture: Historical and current perspectives and future needs and challenges. *Journal of the world aquaculture society*, 51(3):578–633, 2020.

[33] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[34] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 93–104. ACM, 2000.

[35] C. Brix, S. Bak, C. Liu, and T. T. Johnson. The fourth international verification of neural networks competition (vnn-comp 2023): Summary and results. *arXiv preprint arXiv:2312.16760*, 2023.

[36] D. Brughmans, P. Leyman, and D. Martens. NICE: an algorithm for nearest instance counterfactual explanations. *Data Min. Knowl. Discov.*, 38(5):2665–2703, 2024. doi: 10.1007/S10618-023-00930-Y. URL `https://doi.org/10.1007/s10618-023-00930-y`.

[37] R. Bunel, J. Lu, I. Turkaslan, P. H. Torr, P. Kohli, and M. P. Kumar. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42):1–39, 2020.

[38] R. R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. K. Mudigonda. A unified view of piecewise linear neural network verification. *Advances in Neural Information Processing Systems*, 31, 2018.

[39] R. M. J. Byrne. Counterfactuals in explainable artificial intelligence (XAI): evidence from human reasoning. In *IJCAI*, pages 6276–6282, 2019. doi: 10.24963/IJCAI.2019/876. URL `https://doi.org/10.24963/ijcai.2019/876`.

[40] L. Cao. AI in finance: challenges, techniques, and opportunities. *ACM Computing Surveys*, 55(3):1–38, 2022.

[41] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

[42] S. Carr, N. Jansen, and U. Topcu. Verifiable rnn-based policies for pomdps under temporal logic constraints. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 4121–4127, 2021.

[43] S. Carr, N. Jansen, S. Junges, and U. Topcu. Safe reinforcement learning via shielding under partial observability. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 14748–14756, 2023.

[44] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE international conference on computer vision*, pages 2722–2730, 2015.

[45] H. Cheng, H. Hu, and C. Liu. Robust tracking control with neural network dynamic models under input perturbations. *arXiv preprint arXiv:2410.10387*, 2024.

[46] F. Chollet et al. Keras. `https://keras.io`, 2015.

[47] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[48] Y. Chow, O. Nachum, A. Faust, M. Ghavamzadeh, and E. A. Duéñez-Guzmán. Lyapunov-based safe policy optimization for continuous control. In *International Conference on Machine Learning (ICML)*, 2019.

[49] J. Cohen, E. Rosenfeld, and Z. Kolter. Certified adversarial robustness via randomized smoothing. In *international conference on machine learning*, pages 1310–1320. PMLR, 2019.

[50] C. Colas, O. Sigaud, and P.-Y. Oudeyer. A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms. *arXiv preprint arXiv:1904.06979*, 2019.

[51] D. Corsi, E. Marchesini, and A. Farinelli. Formal verification of neural networks for safety-critical tasks in deep reinforcement learning. In *Uncertainty in Artificial Intelligence*, pages 333–343. PMLR, 2021.

[52] D. Corsi, L. Marzari, A. Pore, A. Farinelli, A. Casals, P. Fiorini, and D. Dall'Alba. Constrained reinforcement learning and formal verification for safe colonoscopy navigation. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10289–10294. IEEE, 2023.

[53] N. Couellan. Probabilistic robustness estimates for feed-forward neural networks. *Neural networks*, 142:138–147, 2021.

[54] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In

Conference Record of the 4th ACM Symposium on Principles of Programming Languages (*POPL '77*), pages 238–252. ACM Press, 1977.

[55] S. Dathathri, S. Gao, and R. M. Murray. Inverse abstraction of neural networks using symbolic interpolation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3437–3444, 2019.

[56] S. Dathathri, K. Dvijotham, A. Kurakin, A. Raghunathan, J. Uesato, R. R. Bunel, S. Shankar, J. Steinhardt, I. Goodfellow, P. S. Liang, et al. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *Advances in Neural Information Processing Systems*, 33: 5318–5331, 2020.

[57] M. Dawood, A. Shokry, and M. Bennewitz. A dynamic safety shield for safe and efficient reinforcement learning of navigation tasks. In *7th Annual Conference on Learning for Dynamics and Control*, pages 686–697, 2025.

[58] L. De Haan. Estimation of the minimum of a function using order statistics. *Journal of the American Statistical Association*, 76(374):467–469, 1981.

[59] R. Dominguez-Olmedo, A. Karimi, and B. Schölkopf. On the adversarial robustness of causal algorithmic recourse. In *Proceedings of the International Conference on Machine Learning (ICML22)*, pages 5324–5342, 2022.

[60] D. Dua and C. Graff. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2017. Accessed: 2022-08-30.

[61] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pages 121–138. Springer, 2018.

[62] S. Dutta, J. Long, S. Mishra, C. Tilli, and D. Magazzeni. Robust counterfactual explanations for tree-based ensembles. In *Proceedings of the International Conference on Machine Learning (ICML22)*, pages 5742–5756, 2022.

[63] K. Dvijotham, M. Garnelo, A. Fawzi, and P. Kohli. Verification of deep probabilistic models, 2018.

[64] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. First return, then explore. *Nature*, 590:580–586, 2021. ISSN 1476-4687.

[65] R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*, pages 269–286. Springer, 2017.

[66] I. Elsayed-Aly, S. Bharadwaj, C. Amato, R. Ehlers, U. Topcu, and L. Feng. Safe multi-agent reinforcement learning via shielding. In *AAMAS*, 2021.

[67] M. Everett, G. Habibi, C. Sun, and J. P. How. Reachability analysis of neural feedback loops. *IEEE Access*, 9:163938–163953, 2021.

[68] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. In *International Conference on Learning Representations*, 2018.

[69] J. Fan, C. Huang, X. Chen, W. Li, and Q. Zhu. Reachnn*: A tool for reachability analysis of neural-network controlled systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 537–542. Springer, 2020.

[70] K. Fernandes, P. Vinagre, P. Cortez, and P. Sernadela. Online News Popularity. UCI Machine Learning Repository, 2015. DOI: https://doi.org/10.24432/C5NS3V.

[71] C. Ferrari, M. N. Muller, N. Jovanovic, and M. Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. *ICRL*, 2022.

[72] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[73] M. Forets and C. Schilling. Lazysets. jl: Scalable symbolic-numeric set computations. *arXiv preprint arXiv:2110.01711*, 2021.

[74] V. Gabillon, M. Ghavamzadeh, and B. Scherrer. Approximate dynamic programming finally performs well in the game of tetris. *Advances in neural information processing systems*, 26, 2013.

[75] J. Gajcin and I. Dusparic. Redefining counterfactual explanations for reinforcement learning: Overview, challenges and opportunities. *ACM Computing Surveys*, 56(9):1–33, 2024.

[76] S. Gao, S. Kong, and E. M. Clarke. dreal: An smt solver for nonlinear theories over the reals. In *International conference on automated deduction*, pages 208–214. Springer, 2013.

[77] J. Garcıa and F. Fernández. A comprehensive survey on safe reinforcement learning. In *Journal of Machine Learning Research (JMLR)*, 2015.

[78] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[79] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2018.

[80] B. Ghosh, D. Basu, and K. S. Meel. Justicia: A stochastic sat approach to formally verify fairness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7554–7563, 2021.

[81] R. Giacobazzi, I. Mastroeni, and E. Perantoni. How fitting is your abstract domain? In M. V. Hermenegildo and J. F. Morales, editors, *Static Analysis*, pages 286–309, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-44245-2.

[82] R. Giacobazzi, I. Mastroeni, and E. Perantoni. Adversities in abstract interpretation-accommodating robustness by abstract interpretation. *ACM Transactions on Programming Languages and Systems*, 46(2):1–31, 2024.

[83] W. Gierusz. Modelling the dynamics of ships with different propulsion systems for control purpose. *Polish Maritime Research*, (1):31–36, 2016.

[84] C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. From sampling to model counting. In *IJCAI*, volume 2007, pages 2293–2299, 2007.

[85] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting. In *Handbook of satisfiability*, pages 993–1014. IOS press, 2021.

[86] I. J. Goodfellow, Y. Bengio, and A. C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 978-0-262-03561-3. URL `http://www.deeplearningbook.org/`.

[87] E. Goubault and S. Putot. Rino: robust inner and outer approximated reachability of neural networks controlled systems. In *International Conference on Computer Aided Verification*, pages 511–523. Springer, 2022.

[88] L. Grüne, J. Pannek, L. Grüne, and J. Pannek. *Nonlinear model predictive control*. Springer, 2017.

[89] R. Guidotti. Counterfactual explanations and how to find them: literature review and benchmarking. *Data Mining and Knowledge Discovery*, pages 1–55, 2022.

[90] D. Guihen. The challenges and opportunities for the use of robotic autonomous robotic systems in support of the blue economy. In *International Conference on Offshore Mechanics and Arctic Engineering*, volume 86922. American Society of Mechanical Engineers, 2023.

[91] V. Guyomard, F. Fessant, T. Guyet, T. Bouadi, and A. Termier. Generating robust counterfactual explanations. In *Machine Learning and Knowledge Discovery in Databases: Research Track - European Conference, ECML PKDD 2023*, volume 14171 of *Lecture Notes in Computer Science*, pages 394–409. Springer, 2023.

[92] L. Haan and A. Ferreira. *Extreme value theory: an introduction*, volume 3. Springer, 2006.

[93] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.

[94] F. Hamman, E. Noorani, S. Mishra, D. Magazzeni, and S. Dutta. Robust counterfactual explanations for neural networks with probabilistic guarantees. In *Proceedings of the International Conference on Machine Learning (ICML23)*, pages 12351–12367, 2023.

[95] Y. Han, S. Yang, W. Wang, and J. Liu. From design draft to real attire: Unaligned fashion image translation. In *Proceedings of the 28th ACM international conference on multimedia*, pages 1533–1541, 2020.

[96] M. Harms, M. Kulkarni, N. Khedekar, M. Jacquet, and K. Alexis. Neural control barrier functions for safe navigation. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10415–10422. IEEE, 2024.

[97] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[98] S. He, W. Zhao, C. Hu, Y. Zhu, and C. Liu. A hierarchical long short term safety framework for efficient robot manipulation under uncertainty. *Robotics and Computer-Integrated Manufacturing*, 82:102522, 2023.

[99] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus), 2016.

[100] P. Henriksen and A. Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*, pages 2513–2520. IOS Press, 2020.

[101] R. Henry and D. Ernst. Gym-anm: Reinforcement learning environments for active network management tasks in electricity distribution systems. *Energy and AI*, 5:100092, 2021. ISSN 2666-5468.

[102] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[103] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[104] Z. Hong, T. Shann, S. Su, Y. Chang, and C. Lee. Diversity-driven exploration strategy for deep reinforcement learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[105] M. Hopkins, E. Reeber, G. Forman, and J. Suermondt. Spambase. UCI Machine Learning Repository, 1999. DOI: https://doi.org/10.24432/C53G6X.

[106] T.-F. Hsiao, B.-L. Huang, Z.-X. Ni, Y.-T. Lin, H.-H. Shuai, Y.-H. Li, and W.-H. Cheng. Natural light can also be dangerous: Traffic sign misinterpretation under adversarial natural light attacks. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3915–3924, 2024.

[107] H. Hu, Y. Yang, T. Wei, and C. Liu. Verification of neural control barrier functions with symbolic derivative bounds propagation. In *Conference on Robot Learning*, pages 1797–1814. PMLR, 2025.

[108] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274): 1–18, 2022. URL `http://jmlr.org/papers/v23/21-1342.html`.

[109] A. Ilyas, L. Engstrom, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. A closer look at deep policy gradients. In *ICLR*, 2018.

[110] K. İncetan et al. Vr-caps: A virtual environment for capsule endoscopy. *Med. Imag. Anal.*, 70:101990, 2021.

[111] Y. Jacoby, C. Barrett, and G. Katz. Verifying recurrent neural networks using invariant inference. In *International Symposium on Automated Technology for Verification and Analysis*, pages 57–74. Springer, 2020.

[112] N. Jansen, B. Könighofer, S. Junges, A. Serban, and R. Bloem. Safe reinforcement learning using probabilistic shields. In *31st International Conference on Concurrency Theory (CONCUR 2020)*, pages 3–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

[113] J. Ji, B. Zhang, J. Zhou, X. Pan, W. Huang, R. Sun, Y. Geng, Y. Zhong, J. Dai, and Y. Yang. Safety gymnasium: A unified safe reinforcement learning benchmark. *Advances in Neural Information Processing Systems*, 36, 2024.

[114] J. Ji, J. Zhou, B. Zhang, J. Dai, X. Pan, R. Sun, W. Huang, Y. Geng, M. Liu, and Y. Yang. Omnisafe: An infrastructure for accelerating safe reinforcement learning research. *Journal of Machine Learning Research*, 25(285):1–6, 2024. URL `http://jmlr.org/papers/v25/23-0681.html`.

[115] J. Jiang, J. Lan, F. Leofante, A. Rago, and F. Toni. Provably robust and plausible counterfactual explanations for neural networks via robust optimisation. In *ACML*, volume 222 of *Proceedings of Machine Learning Research*, pages

582–597. PMLR, 2023. URL https://proceedings.mlr.press/v222/jiang24a.html.

[116] J. Jiang, F. Leofante, A. Rago, and F. Toni. Formalising the robustness of counterfactual explanations for neural networks. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI23)*, pages 14901–14909, 2023.

[117] J. Jiang, F. Leofante, A. Rago, and F. Toni. Recourse under model multiplicity via argumentative ensembling. In *AAMAS*, pages 954–963. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2024. doi: 10.5555/3635637.3662950. URL https://dl.acm.org/doi/10.5555/3635637.3662950.

[118] J. Jiang, F. Leofante, A. Rago, and F. Toni. Robust counterfactual explanations in machine learning: A survey. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pages 8086–8094. ijcai.org, 2024. URL https://www.ijcai.org/proceedings/2024/894.

[119] J. Jiang, F. Leofante, A. Rago, and F. Toni. Interval abstractions for robust counterfactual explanations. *Artif. Intell.*, 336:104218, 2024. doi: 10.1016/J.ARTINT.2024.104218. URL https://doi.org/10.1016/j.artint.2024.104218.

[120] J. Jiang, F. Leofante, A. Rago, and F. Toni. Robust counterfactual explanations in machine learning: A survey. In *IJCAI*, pages 8086–8094, 2024. Survey Track.

[121] J. Jiang, A. Rago, F. Leofante, and F. Toni. Recourse under model multiplicity via argumentative ensembling. In *Proceedings of the 2024 International Conference on Autonomous Agents and Multiagent Systems (AAMAS24)*, 2024.

[122] J. Jiang, L. Marzari, A. Purohit, and F. Leofante. Robustx: Robust counterfactual explanations made easy. In J. Kwok, editor, *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pages 11067–11071. International Joint Conferences on Artificial Intelligence Organization, 8 2025. doi: 10.24963/ijcai.2025/1264. URL https://doi.org/10.24963/ijcai.2025/1264.

[123] Y. Jiang, J. Z. Kolter, and R. Raileanu. On the importance of exploration for generalization in reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=y5duN2j9s6.

[124] K. D. Julian and M. J. Kochenderfer. A reachability method for verifying dynamical systems with deep neural network controllers. *arXiv preprint arXiv:1903.00520*, 2019.

[125] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE, 2016.

[126] A. Juliani, V. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange. Unity: A platform for intelligent agents. In *CoRR*, 2018.

[127] S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on*

Machine Learning, pages 267–274, 2002.

[128] P. Kapoor, A. Balakrishnan, and J. V. Deshmukh. Model-based reinforcement learning from signal temporal logic specifications. In *arXiv*, 2020.

[129] A. Karimi, G. Barthe, B. Schölkopf, and I. Valera. A survey of algorithmic recourse: Contrastive explanations and consequential recommendations. *ACM Comput. Surv.*, 55(5):95:1–95:29, 2023.

[130] R. M. Karp and M. Luby. Monte-carlo algorithms for the planar multiterminal network reliability problem. *Journal of Complexity*, 1(1):45–64, 1985.

[131] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International conference on computer aided verification*, pages 97–117. Springer, 2017.

[132] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: a calculus for reasoning about deep neural networks. *Formal Methods in System Design*, pages 1–30, 2021.

[133] M. T. Keane, E. M. Kenny, E. Delaney, and B. Smyth. If only we had better counterfactual explanations: Five key deficits to rectify in the evaluation of counterfactual XAI techniques. In *IJCAI*, pages 4466–4474, 2021. doi: 10.24963/IJCAI.2021/609. URL `https://doi.org/10.24963/ijcai.2021/609`.

[134] A. Kofnov, D. Kapla, E. Bartocci, and E. Bura. Exact upper and lower bounds for the output distribution of neural networks with random inputs. In *Proc. of ICML 2025: Forty-Second International Conference on Machine Learning*, 2025.

[135] B. Könighofer, F. Lorber, N. Jansen, and R. Bloem. Shield synthesis for reinforcement learning. In *International symposium on leveraging applications of formal methods*, pages 290–306. Springer, 2020.

[136] S. Kotha, C. Brix, J. Z. Kolter, K. Dvijotham, and H. Zhang. Provably bounding neural network preimages. *Advances in Neural Information Processing Systems*, 36:80270–80290, 2023.

[137] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[138] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[139] M. G. Lagoudakis and R. Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 424–431, 2003.

[140] J. F. Lazo et al. Autonomous intraluminal navigation of a soft robot using deep-learning-based visual servoing. In *Proc. IEEE Int. Conf. Intell. Robot. Syst.*, pages 6952–6959. IEEE, 2022.

[141] C. Lemaréchal. Cauchy and the gradient method. *Doc Math Extra*, 251(254): 10, 2012.

[142] F. Leofante and A. Lomuscio. Towards robust contrastive explanations for human-neural multi-agent systems. In N. Agmon, B. An, A. Ricci, and W. Yeoh, editors, *Proceedings of the 2023 International Conference on Au-*

tonomous Agents and Multiagent Systems (AAMAS23), pages 2343–2345. ACM, 2023.

[143] F. Leofante and N. Potyka. Promoting counterfactual robustness through diversity. In *AAAI*, pages 21322–21330, 2024. doi: 10.1609/AAAI.V38I19.30127. URL https://doi.org/10.1609/aaai.v38i19.30127.

[144] F. Leofante, E. Botoeva, and V. Rajani. Counterfactual explanations and model multiplicity: a relational verification view. In *KR*, pages 763–768, 2023. doi: 10.24963/KR.2023/78. URL https://doi.org/10.24963/kr.2023/78.

[145] F. Leofante, E. Botoeva, and V. Rajani. Counterfactual explanations and model multiplicity: a relational verification view. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning (KR23)*, pages 763–768, 2023.

[146] G. Li, Y. Xie, T. Wei, K. Wang, and L. Lin. Flow guided recurrent neural encoder for video salient object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3243–3252, 2018.

[147] M. Li, A. Mickel, and S. Taylor. "should this loan be approved or denied?": A large dataset with class assignment guidelines. *Journal of Statistics Education*, 26(1):55–66, 2018.

[148] Y. Lin, E. D. Sontag, and Y. Wang. A smooth converse lyapunov theorem for robust stability. *SIAM Journal on Control and Optimization*, 34(1):124–160, 1996.

[149] Z. C. Lipton, J. Gao, L. Li, J. Chen, and L. Deng. Combating reinforcement learning's sisyphean curse with intrinsic fear. In *arXiv*, 2016.

[150] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, M. J. Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021.

[151] S. Liu, C. Liu, and J. Dolan. Safe control under input limits with neural control barrier functions. In *Conference on Robot Learning*, pages 1970–1980. PMLR, 2023.

[152] Y. Liu, J. Ding, and X. Liu. IPO: interior-point policy optimization under constraints. In *AAAI Conference on Artificial Intelligence*, 2020.

[153] R. Livni, S. Shalev-Shwartz, and O. Shamir. On the computational efficiency of training neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 855–863, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/3a0772443a0739141292a5429b952fe6-Abstract.html.

[154] A. Lomuscio and L. Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

[155] D. M. Lopez, S. W. Choi, H.-D. Tran, and T. T. Johnson. Nnv 2.0: the neural network verification tool. In *International Conference on Computer Aided Verification*, pages 397–412. Springer, 2023.

[156] I. D. Lopez-Miguel, S. Adam, E. Bartocci, T. Eiter, and M. Tappler. Rule-guided reinforcement learning policy evaluation and improvement. In *Proc. of ECAI 2025*, 2025.

[157] X. Luo, T. Wei, S. Liu, Z. Wang, L. Mattei-Mendez, T. Loper, J. Neighbor, C. Hutchison, and C. Liu. Certifying robustness of learning-based keypoint detection and pose estimation methods. *arXiv preprint arXiv:2408.00117*, 2024.

[158] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *In International Conference on Learning Representations (ICLR)*, 2018.

[159] L. Manfredi. Endorobots for colonoscopy: design challenges and available technologies. *Front. Robot. AI*, 8:705454, 2021.

[160] R. Mangal, A. V. Nori, and A. Orso. Robustness of neural networks: A probabilistic and practical approach. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 93–96. IEEE, 2019.

[161] D. Manzanas Lopez, P. Musau, N. P. Hamilton, and T. T. Johnson. Reachability analysis of a general class of neural ordinary differential equations. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 258–277. Springer, 2022.

[162] E. Marchesini and C. Amato. Safety-informed mutations for evolutionary deep reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, page 1966–1970, 2022. ISBN 9781450392686.

[163] E. Marchesini and C. Amato. Improving deep policy gradients with value function search. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=6qZC7pfenQm`.

[164] E. Marchesini and A. Farinelli. Discrete deep reinforcement learning for mapless navigation. In *ICRA*, 2020.

[165] E. Marchesini and A. Farinelli. Centralizing state-values in dueling networks for multi-robot reinforcement learning mapless navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4583–4588, 2021. doi: 10.1109/IROS51168.2021.9636349.

[166] E. Marchesini and A. Farinelli. Centralizing state-values in dueling networks for multi-robot reinforcement learning mapless navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4583–4588, 2021. doi: 10.1109/IROS51168.2021.9636349.

[167] E. Marchesini and A. Farinelli. Enhancing deep reinforcement learning approaches for multi-robot navigation via single-robot evolutionary policy search. In *International Conference on Robotics and Automation (ICRA)*, pages 5525–5531, 2022. doi: 10.1109/ICRA46639.2022.9812341.

[168] E. Marchesini, L. Marzari, A. Farinelli, and C. Amato. Safe deep reinforcement learning by verifying task-level properties. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pages 1466–1475, 2023.

[169] E. Marchesini, B. Donnot, C. Crozier, I. Dytham, C. Merz, L. Schewe, N. Westerbeck, C. Wu, A. Marot, and P. L. Donti. Rl2grid: Benchmarking reinforcement learning in power grid operations. *arXiv preprint arXiv:2503.23101*, 2025.

[170] J. W. Martin et al. Enabling the future of colonoscopy with intelligent and autonomous magnetic manipulation. *Nature Mach. Intell.*, 2(10):595–606, 2020.

[171] L. Marzari, A. Pore, D. Dall'Alba, G. Aragon-Camarasa, A. Farinelli, and P. Fiorini. Towards hierarchical task decomposition using deep reinforcement learning for pick and place subtasks. In *2021 20th International Conference on Advanced Robotics (ICAR)*, pages 640–645. IEEE, 2021.

[172] L. Marzari, D. Corsi, E. Marchesini, and A. Farinelli. Curriculum learning for safe mapless navigation. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, pages 766–769, 2022.

[173] L. Marzari, D. Corsi, F. Cicalese, and A. Farinelli. The #dnn-verification problem: Counting unsafe inputs for deep neural networks. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023*, pages 217–224. ijcai.org, 2023. doi: 10.24963/IJCAI.2023/25. URL https://doi.org/10.24963/ijcai.2023/25.

[174] L. Marzari, E. Marchesini, and A. Farinelli. Online safety property collection and refinement for safe deep reinforcement learning in mapless navigation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7133–7139. IEEE, 2023.

[175] L. Marzari, G. Roncolato, and A. Farinelli. Scaling #dnn-verification tools with efficient bound propagation and parallel computing. In *AIRO 2023 Artificial Intelligence and Robotics 2023*, pages 92–106. 2023.

[176] L. Marzari, D. Corsi, E. Marchesini, A. Farinelli, and F. Cicalese. Enumerating safe regions in deep neural networks with provable probabilistic guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 21387–21394, 2024.

[177] L. Marzari, F. Leofante, F. Cicalese, and A. Farinelli. Rigorous probabilistic guarantees for robust counterfactual explanations. In *ECAI*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pages 1059–1066. IOS Press, 2024. doi: 10.3233/FAIA240597. URL https://doi.org/10.3233/FAIA240597.

[178] L. Marzari, F. Cicalese, and A. Farinelli. Probabilistically tightened linear relaxation-based perturbation analysis for neural network verification. *arXiv preprint arXiv:2507.05405. Currently under review at Journal of Artificial Intelligence Research*, 2025.

[179] L. Marzari, F. Cicalese, A. Farinelli, C. Amato, and E. Marchesini. Verifying online safety properties for safe deep reinforcement learning. *ACM Trans. Intell. Syst. Technol.*, Sept. 2025. ISSN 2157-6904. doi: 10.1145/3770068. URL https://doi.org/10.1145/3770068.

[180] L. Marzari, P. L. Donti, C. Liu, and E. Marchesini. Improving policy optimization via ϵ-retrain. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2025*,

pages 1464–1472. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2025. doi: 10.5555/3709347.3743780. URL https://dl.acm.org/doi/10.5555/3709347.3743780.

[181] L. Marzari, F. Leofante, and E. Marchesini. Explaining reinforcement learning policies for power grid operations. In *Joint Proceedings of the Thematic Workshops at Ital-IA 2025 colocated with the 5th National Conference on Artificial Intelligence, organized by CINI (Ital-IA 2025), Trieste, Italy, June 23-24, 2025*, volume 4121 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2025. URL https://ceur-ws.org/Vol-4121/Ital-IA_2025_paper_7.pdf.

[182] L. Marzari, I. Mastroeni, and A. Farinelli. Advancing neural network verification through hierarchical safety abstract interpretation. In *ECAI 2025*, Frontiers in Artificial Intelligence and Applications, pages 1736–1743. IOS Press, 2025. doi: 10.3233/FAIA251002. URL https://ebooks.iospress.nl/doi/10.3233/FAIA251002.

[183] L. Marzari, F. Trotti, E. Marchesini, and A. Farinelli. Designing control barrier function via probabilistic enumeration for safe reinforcement learning navigation. *IEEE Robotics and Automation Letters*, 10(10):9630–9637, 2025. doi: 10.1109/LRA.2025.3596431.

[184] L. Marzari, F. Trotti, F. D. Santo, A. Zhalehmehrabi, C. Veronese, D. Villaboni, F. Bianchi, D. Meli, A. Castellini, and A. Farinelli. Enhancing safety and explainability of reinforcement learning agents for environmental monitoring tasks. In *Joint Proceedings of the Thematic Workshops at Ital-IA 2025 colocated with the 5th National Conference on Artificial Intelligence, organized by CINI (Ital-IA 2025), Trieste, Italy, June 23-24, 2025*, volume 4121 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2025. URL https://ceur-ws.org/Vol-4121/Ital-IA_2025_paper_10.pdf.

[185] I. Mastroeni. Abstract domain adequacy. *Int. J. Softw. Tools Technol. Transf.*, 26(6):747–765, 2024. doi: 10.1007/S10009-024-00774-X. URL https://doi.org/10.1007/s10009-024-00774-x.

[186] K. Matoba and F. Fleuret. Exact preimages of neural network aircraft collision avoidance systems. In *Proceedings of the Machine Learning for Engineering Modeling, Simulation, and Design Workshop at Neural Information Processing Systems*, pages 1–9, 2020.

[187] G. Mazzi, D. Meli, A. Castellini, and A. Farinelli. Learning logic specifications for soft policy guidance in pomcp. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pages 373–381, 2023.

[188] L. Medsker and L. C. Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[189] D. Meli, A. Castellini, and A. Farinelli. Learning logic specifications for policy guidance in pomdps: an inductive logic programming approach. *Journal of Artificial Intelligence Research*, 79:725–776, 2024.

[190] M. H. Meng, G. Bai, S. G. Teo, Z. Hou, Y. Xiao, Y. Lin, and J. S. Dong. Adversarial robustness of deep neural networks: A survey from a formal verification perspective. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[191] N. Messikommer, Y. Song, and D. Scaramuzza. Contrastive initial state buffer for reinforcement learning. *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[192] Z. Mhammedi, D. J. Foster, and A. Rakhlin. The power of resets in online reinforcement learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[193] T. Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267:1–38, 2019. doi: 10.1016/j.artint.2018.07.007. URL `https://doi.org/10.1016/j.artint.2018.07.007`.

[194] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, 2017. ISBN 978-1-107-15488-9.

[195] K. Mohammadi, A. Karimi, G. Barthe, and I. Valera. Scaling guarantees for nearest counterfactual explanations. In *AIES*, pages 177–187, 2021. doi: 10.1145/3461702.3462514. URL `https://doi.org/10.1145/3461702.3462514`.

[196] R. E. Moore, R. B. Kearfott, and M. J. Cloud. Introduction to interval analysis. In *SIAM*, 2009.

[197] P. Morettin, A. Passerini, and R. Sebastiani. A unified framework for probabilistic verification of ai systems via weighted model integration, 2024.

[198] M. N. Müller, C. Brix, S. Bak, C. Liu, and T. T. Johnson. The third international verification of neural networks competition (vnn-comp 2022): Summary and results, 2022.

[199] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

[200] T.-D. H. Nguyen, N. Bui, D. Nguyen, M.-C. Yue, and V. A. Nguyen. Robust Bayesian recourse. In *Proceedings of the 38hth Conference on Uncertainty in AI (UAI22)*, pages 1498–1508, 2022.

[201] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2 edition, 2006. ISBN 9780387400655.

[202] Observatory (2021), Global Cancer Observatory, 2021. available at `https://gco.iarc.fr/`.

[203] K. O'Shea and R. Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[204] Paszke. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035. Curran Associates, Inc., 2019.

[205] B. Paulsen and C. Wang. Linsyn: Synthesizing tight linear bounds for arbitrary neural network activation functions. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 357–376. Springer, 2022.

[206] M. Pautov, N. Tursynbek, M. Munkhoeva, N. Muravev, A. Petiushko, and I. Oseledets. Cc-cert: A probabilistic approach to certify general robustness of neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7975–7983, 2022.

[207] M. Pawelczyk, K. Broelemann, and G. Kasneci. Learning model-agnostic counterfactual explanations for tabular data. In *Proceedings of the Web Conference (WWW20)*, pages 3126–3132. ACM / IW3C2, 2020.

[208] M. Pawelczyk, S. Bielawski, J. van den Heuvel, T. Richter, and G. Kasneci. CARLA: A python library to benchmark algorithmic recourse and counterfactual explanation algorithms. In *NeurIPS Datasets and Benchmarks*, 2021.

[209] M. Pawelczyk, T. Datta, J. van den Heuvel, G. Kasneci, and H. Lakkaraju. Probabilistically robust recourse: Navigating the trade-offs between costs and robustness in algorithmic recourse. In *Proceedings of the 11th International Conference on Learning Representations (ICLR23)*. OpenReview.net, 2023.

[210] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[211] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, July 2018.

[212] F. J. Pierce and P. Nowak. Aspects of precision agriculture. *Advances in agronomy*, 67:1–85, 1999.

[213] D. Pollard. Asymptopia: an exposition of statistical asymptotic theory. In *Asymptopia: an exposition of statistical asymp-totic theory*, 2000.

[214] A. Pore, D. Corsi, E. Marchesini, D. Dall'Alba, A. Casals, A. Farinelli, and P. Fiorini. Safe reinforcement learning using formal verification for tissue retraction in autonomous robotic-assisted surgery. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4025–4031. IEEE, 2021.

[215] A. Pore et al. Colonoscopy navigation using end-to-end deep visuomotor control: A user study. In *Proc. IEEE Int. Conf. Intell. Robot. Syst.*, pages 9582–9588. IEEE, 2022.

[216] N. Porter. Wilks' formula applied to computational tools: A practical discussion and verification. *Annals of Nuclear Energy*, 133:129–137, 2019.

[217] R. Poyiadzi, K. Sokol, R. Santos-Rodríguez, T. D. Bie, and P. A. Flach. FACE: feasible and actionable counterfactual explanations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society (AIES20)*, pages 344–350, 2020.

[218] P. Prabhakar and Z. R. Afzal. Abstraction based output range analysis for neural networks. In *Advances in Neural Information Processing Systems 32 (NeurIPS19)*, pages 15762–15772, 2019.

[219] J. M. Prendergast, G. A. Formosa, et al. A real-time state dependent region estimator for autonomous endoscope navigation. *IEEE Trans. Robot.*, 37(3): 918–934, 2020.

[220] J. M. Prendergast et al. Autonomous localization, navigation and haustral fold detection for robotic endoscopy. In *Proc. IEEE Int. Conf. Intell. Robot. Syst.*, pages 783–790. IEEE, 2018.

[221] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions, 2017.

[222] A. Ray, J. Achiam, and D. Amodei. Benchmarking safe exploration in deep reinforcement learning. In *arXiv*, 2019.

[223] A. Ray, J. Achiam, and D. Amodei. Benchmarking safe exploration in deep reinforcement learning. In *OpenAI Blog*, 2019.

[224] R. Reilink, S. Stramigioli, and S. Misra. Image-based flexible endoscope steering. In *Proc. IEEE Int. Conf. Intell. Robot. Syst.*, pages 2339–2344. IEEE, 2010.

[225] D. B. Rokhlin. Robbins–monro conditions for persistent exploration learning strategies. In *International Scientific Conference (on) Modern Methods, Problems and Applications of Operator Theory and Harmonic Analysis*, pages 237–247. Springer, 2018.

[226] J. Roy, R. Girgis, J. Romoff, P.-L. Bacon, and C. Pal. Direct behavior specification via constrained reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2022.

[227] J. Roy, R. Girgis, J. Romoff, P.-L. Bacon, and C. J. Pal. Direct behavior specification via constrained reinforcement learning. In *ICML*, volume 162, pages 18828–18843, 2022.

[228] W. Rudin et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1964.

[229] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[230] C. Schilling, M. Forets, and S. Guadalupe. Verification of neural-network control systems by integrating taylor models and zonotopes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8169–8177, 2022.

[231] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[232] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[233] M. Y. Shaheen. Applications of artificial intelligence (ai) in healthcare: A review. *ScienceOpen Preprints*, 2021.

[234] Z. Shi, H. Zhang, K.-W. Chang, M. Huang, and C.-J. Hsieh. Robustness verification for transformers. In *ICLR*, 2020.

[235] Z. Shi, H. Zhang, K.-W. Chang, M. Huang, and C.-J. Hsieh. Robustness verification for transformers. In *International Conference on Learning Representations*, 2020.

[236] D. Silver, S. Singh, D. Precup, and R. S. Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, 2021.

[237] G. Singh, T. Gehr, M. Püschel, and M. Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

[238] V. Sivaramakrishnan, K. C. Kalagarla, R. Devonport, J. Pilipovsky, P. Tsiotras, and M. Oishi. Saver: A toolbox for sampling-based, probabilistic verification of neural networks. *arXiv preprint arXiv:2412.02940*, 2024.

[239] D. Slack, A. Hilgard, H. Lakkaraju, and S. Singh. Counterfactual explanations can be manipulated. In *Advances in Neural Information Processing Systems 34 (NeurIPS21)*, pages 62–75, 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/009c434cab57de48a31f6b669e7ba266-Abstract.html`.

[240] J. W. Smith, J. E. Everhart, W. Dickson, W. C. Knowler, and R. S. Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the annual symposium on computer application in medical care*, page 261. American Medical Informatics Association, 1988.

[241] O. So, Z. Serlin, M. Mann, J. Gonzales, K. Rutledge, N. Roy, and C. Fan. How to train your neural control barrier function: Learning safety filters for complex input-constrained systems. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11532–11539. IEEE, 2024.

[242] I. Stepin, J. M. Alonso, A. Catalá, and M. Pereira-Fariña. A survey of contrastive and counterfactual explanation generation methods for explainable artificial intelligence. *IEEE Access*, 9:11974–12001, 2021.

[243] A. Stooke, J. Achiam, and P. Abbeel. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning (ICML)*, 2020.

[244] A. L. Strehl, L. Li, and M. L. Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(11), 2009.

[245] H. Sung, J. Ferlay, R. L. Siegel, M. Laversanne, I. Soerjomataram, A. Jemal, and F. Bray. Global cancer statistics 2020: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA: a cancer journal for clinicians*, 71(3):209–249, 2021.

[246] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[247] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. Chaplot, O. Maksymets, A. Gokaslan, V. Vondrus, S. Dharur, F. Meier, W. Galuba, A. Chang, Z. Kira, V. Koltun, J. Malik, M. Savva, and D. Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[248] E. Tagliabue, A. Pore, D. Dall'Alba, E. Magnabosco, M. Piccinelli, and P. Fiorini. Soft tissue simulation environment to learn manipulation tasks in autonomous robotic surgery. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3261–3266. IEEE, 2020.

[249] L. Tai, G. Paolo, and M. Liu. Virtual-to-real drl: Continuous control of mobile robots for mapless navigation. In *IROS*, 2017.

[250] M. Tappler, I. D. Lopez-Miguel, S. Tschiatschek, and E. Bartocci. Rule-guided reinforcement learning policy evaluation and improvement. In *Proc. of IJCAI 2025*, 2025.

[251] C. Tessler, D. J. Mankowitz, and S. Mannor. Reward constrained policy optimization. In *International Conference on Learning Representations*, 2019.

[252] V. Tjeng, K. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2017.

[253] V. Tjeng, K. Y. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2018.

[254] H.-D. Tran, S. Bak, W. Xiang, and T. T. Johnson. Verification of deep convolutional neural networks using imagestars. In *International conference on computer aided verification*, pages 18–42. Springer, 2020.

[255] H.-D. Tran, T. T. Johnson, et al. Reachability analysis of recurrent neural networks using star sets. In *HSCC*, 2023.

[256] F. Trotti, A. Farinelli, and R. Muradore. An online path planner based on pomdp for uavs. In *2023 European Control Conference (ECC)*, pages 1–6. IEEE, 2023.

[257] F. Trotti, A. Farinelli, and R. Muradore. A markov decision process approach for decentralized uav formation path planning. In *2024 European Control Conference (ECC)*, pages 436–441. IEEE, 2024.

[258] F. Trotti, A. Farinelli, and R. Muradore. Path re-planning with stochastic obstacle modeling: A monte carlo tree search approach. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8017–8022. IEEE, 2024.

[259] F. Trotti, A. Farinelli, and R. Muradore. Towards aircraft autonomy using a pomdp-based planner. In *2024 American Control Conference (ACC)*, pages 2399–2404. IEEE, 2024.

[260] C. E. Tuncali, J. Kapinski, H. Ito, and J. V. Deshmukh. Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In *Design Automation Conference (DAC)*, 2018.

[261] E. Union. Regulation (eu) 2021/0106 on a european approach for artificial intelligence, 2024. URL `https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng`. Accessed: 2024-06-12.

[262] S. Upadhyay, S. Joshi, and H. Lakkaraju. Towards robust and reliable algorithmic recourse. In *NeurIPS*, pages 16926–16937, 2021.

[263] B. Ustun, A. Spangher, and Y. Liu. Actionable recourse in linear classification. In *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT*19)*, pages 10–19, 2019. doi: 10.1145/3287560.3287566. URL `https://doi.org/10.1145/3287560.3287566`.

[264] L. G. Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

[265] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[266] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27 (11):1134–1142, 1984.

[267] F. Van Erning, L. Van Steenbergen, V. Lemmens, H. Rutten, H. Martijn, D. J. van Spronsen, and M. Janssen-Heijnen. Conditional survival for long-term

colorectal cancer survivors in the netherlands: who do best? *European Journal of Cancer*, 50(10):1731–1739, 2014.

[268] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. Openml: networked science in machine learning. *SIGKDD Explor.*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL https://doi.org/10.1145/2641190.2641198.

[269] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[270] M. Virgolin and S. Fracaros. On the robustness of sparse counterfactual expl. to adverse perturbations. *Artif. Intell.*, 316, 2023. doi: 10.1016/j.artint.2022.103840. URL https://doi.org/10.1016/j.artint.2022.103840.

[271] S. Wachter, B. D. Mittelstadt, and C. Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.*, 31:841, 2017.

[272] S. Wachter, B. D. Mittelstadt, and C. Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *CoRR*, abs/1711.00399, 2017. URL http://arxiv.org/abs/1711.00399.

[273] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Efficient formal safety analysis of neural networks. *Advances in Neural Information Processing Systems*, 31, 2018.

[274] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1599–1614, 2018.

[275] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems*, 34:29909–29921, 2021.

[276] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning (ICML)*, pages 1995–2003, 2016.

[277] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[278] S. Webb, T. Rainforth, Y. W. Teh, and M. P. Kumar. A statistical approach to assessing neural network robustness, 2018.

[279] T. Wei and C. Liu. Safe control algorithms using energy functions: A uni ed framework, benchmark, and new directions. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 238–243. IEEE, 2019.

[280] T. Wei and C. Liu. Safe control with neural network dynamic models. In *Learning for Dynamics and Control Conference*, pages 739–750. PMLR, 2022.

[281] T. Wei, L. Ma, R. Chen, W. Zhao, and C. Liu. Meta-control: Automatic model-based control synthesis for heterogeneous robot skills. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=cvVEkS5yij.

[282] T. Wei, H. Hu, L. Marzari, K. S. Yun, P. Niu, X. Luo, and C. Liu. Modelverification.jl: A comprehensive toolbox for formally verifying deep neural networks. In *Computer Aided Verification*, pages 395–408. Springer Nature Switzerland, 2025. ISBN 978-3-031-98679-6.

[283] L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*, pages 5276–5285. PMLR, 2018.

[284] L. Weng, P.-Y. Chen, L. Nguyen, M. Squillante, A. Boopathy, I. Oseledets, and L. Daniel. Proven: Verifying robustness of neural networks with a probabilistic approach. In *International Conference on Machine Learning*, pages 6727–6736. PMLR, 2019.

[285] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. 2018.

[286] S. S. Wilks. Statistical prediction with special reference to the problem of tolerance limits. *The annals of mathematical statistics*, 13(4):400–409, 1942.

[287] H. Wu, O. Isac, A. Zeljić, T. Tagomori, M. Daggitt, W. Kokke, I. Refaeli, G. Amir, K. Julian, S. Bassan, et al. Marabou 2.0: a versatile formal analyzer of neural networks. In *International Conference on Computer Aided Verification*, pages 249–264. Springer, 2024.

[288] W. Xiang and T. T. Johnson. Reachability analysis and safety verification for neural network control systems. *arXiv preprint arXiv:1805.09944*, 2018.

[289] W. Xiang, H.-D. Tran, and T. T. Johnson. Reachable set computation and safety verification for neural networks with relu activations. *arXiv preprint arXiv:1712.08163*, 2017.

[290] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020.

[291] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C.-J. Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *arXiv preprint arXiv:2011.13824*, 2020.

[292] Z. Xue, S. Liu, Z. Zhang, Y. Wu, and M. Zhang. A tale of two approximations: Tightening over-approximation for dnn robustness verification via under-approximation. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2023, page 1182–1194, 2023.

[293] T.-Y. Yang, J. Rosca, K. Narasimhan, and P. J. Ramadge. Projection-based constrained policy optimization. In *International Conference on Learning Representations (ICLR)*, 2020.

[294] X. Yang, T. Yamaguchi, H.-D. Tran, B. Hoxha, T. T. Johnson, and D. Prokhorov. Neural network repair with reachability analysis. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 221–236. Springer, 2022.

[295] Y. Yang, H. Hu, T. Wei, S. E. Li, and C. Liu. Scalable synthesis of formally verified neural value function for hamilton-jacobi reachability analysis. *arXiv preprint arXiv:2407.20532*, 2024.

[296] M. Zarei, Y. Wang, and M. Pajic. Statistical verification of learning-based cyber-physical systems. In *International Conference on Hybrid Systems: Computation and Control*, 2020.

[297] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.

[298] H. Zhang, S. Wang, K. Xu, L. Li, B. Li, S. Jana, C.-J. Hsieh, and J. Z. Kolter. General cutting planes for bound-propagation-based neural network verification. *Advances in neural information processing systems*, 35:1656–1670, 2022.

[299] H. Zhang, S. Wang, K. Xu, Y. Wang, S. Jana, C.-J. Hsieh, and Z. Kolter. A branch and bound framework for stronger adversarial attacks of relu networks. In *International Conference on Machine Learning*, pages 26591–26604. PMLR, 2022.

[300] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *IROS*, 2017.

[301] L. Zhang, L. Shen, L. Yang, S. Chen, B. Yuan, X. Wang, and D. Tao. Penalized proximal policy optimization for safe reinforcement learning. In *IJCAI*, 2022.

[302] S. Zhang, X. Chen, S. Wen, and Z. Li. Density-based reliable and robust explainer for counterfactual explanation. *Expert Syst. Appl.*, 226:120214, 2023.

[303] X. Zhang, Y. Peng, B. Luo, W. Pan, X. Xu, and H. Xie. Model-based safe reinforcement learning with time-varying state and control constraints: An application to intelligent vehicles. In *arXiv*, 2021.

[304] X. Zhang, B. Wang, and M. Kwiatkowska. Provable preimage under-approximation for neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 3–23. Springer, 2024.

[305] X. Zhang, B. Wang, M. Kwiatkowska, and H. Zhang. Premap: A unifying preimage approximation framework for neural networks. *arXiv preprint arXiv:2408.09262*, 2025.

[306] Y. Zhang, Q. Vuong, and K. W. Ross. First order constrained optimization in policy space. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[307] Y. Zhang, Z. Zhao, G. Chen, F. Song, and T. Chen. Bdd4bnn: a bdd-based quantitative analysis framework for binarized neural networks. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33*, pages 175–200. Springer, 2021.

[308] Y. Zhang, Y. Tang, W. Ruan, X. Huang, S. Khastgir, P. Jennings, and X. Zhao. Protip: Probabilistic robustness verification on text-to-image diffusion models against stochastic perturbation. In *European Conference on Computer Vision*, pages 455–472. Springer, 2025.

[309] W. Zhao, T. He, and C. Liu. Probabilistic safeguard for reinforcement learning using safety index guided gaussian process models. In *L4DC*, 2023.