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

Luca Marzari*, Gabriele Roncolato and Alessandro Farinelli

Department of Computer Science, Univerisity of Verona, Italy

Abstract

Deep Neural Networks (DNNs) are powerful tools that have shown extraordinary results in many scenarios, ranging from pattern recognition to complex robotic problems. However, their intricate designs and lack of transparency raise safety concerns when applied in real-world applications. In this context, Formal Verification (FV) of DNNs has emerged as a valuable solution to provide provable guarantees on the safety aspect. Nonetheless, the binary answer (i.e., `safe` or `unsafe`) could be not informative enough for direct safety interventions such as safety model ranking or selection. To address this limitation, the FV problem has recently been extended to the counting version, called *#DNN-Verification*, for the computation of the size of the unsafe regions in a given safety property's domain. Still, due to the complexity of the problem, existing solutions struggle to scale on real-world robotic scenarios, where the DNN can be large and complex. To address this limitation, inspired by advances in FV, in this work, we propose a novel strategy based on reachability analysis combined with Symbolic Linear Relaxation and parallel computing to enhance the efficiency of existing exact and approximate FV for DNN counters. The empirical evaluation on standard FV benchmarks and realistic robotic scenarios shows a remarkable improvement in scalability and efficiency, enabling the use of such techniques even for complex robotic applications.

Keywords

Formal Verification of Deep Neural Network, Safety for Robotics, Parallel Computing

1. Introduction

In recent years, the use of Deep Neural Networks (DNNs) has increased due to their ability to learn complex patterns from vast amounts of data. In detail, DNNs have emerged as a novel technology revolutionizing various fields ranging from image classification [1], robotic manipulation [2, 3], robotics for medical applications [4, 5], and even for autonomous navigation [6, 7]. As the DNNs become more powerful and pervasive in our applications, the safety aspect has become increasingly prominent. The typical non-linear and non-convex nature of these approximator functions raises two critical concerns: (i) the behavior of these networks is often not comprehensible, leading them to be called "black boxes" (ii) slight human-imperceptible changes in the domain of these functions can cause drastic mispredictions that can endanger both the safety aspect of the intelligent agents and the human life in the real-world applications. More specifically, DNNs are vulnerable to the so-called "adversarial attacks"[8]. To provide the reader with an intuition of what these adversarial inputs are in practice in a robotic scenario, suppose to

10th Italian Workshop on Artificial Artificial Intelligence and Robotics (AIRO 2023)

*Corresponding author.

luca.marzari@univr.it (L. Marzari); gabriele.roncolato@studenti.univr.it (G. Roncolato); alessandro.farinelli@univr.it (A. Farinelli)

consider a context where we have a mobile robot trained to navigate in a particular environment. The agent's goal is to navigate towards a random target placed in the environment without having access to a map, using only Lidar values to sense the surroundings. In this context, if we train a neural network, for instance, with some Deep Reinforcement Learning (DRL) techniques, it has been shown in [9] how, despite empirically achieving a high level of success rate (measured in terms of how many times the agent reaches the goal) and safety (measured in terms of collision rate with obstacles), it is possible to find particular input configurations, that when propagated through the network, lead to suboptimal behaviors, such as the one shown in Fig. 1.

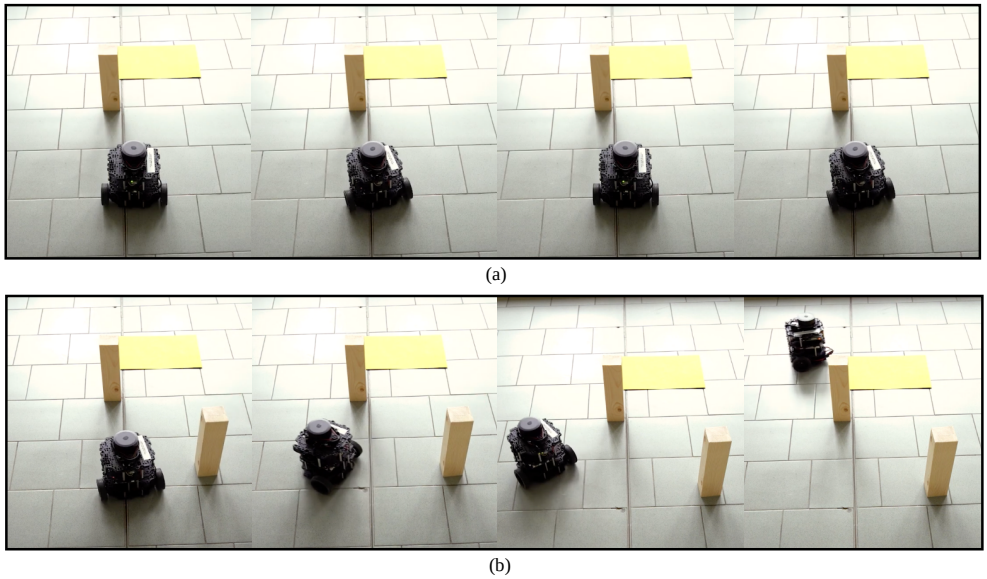


Figure 1: Explanatory image of adversarial input in a DRL setup. The robot is trained and is generally able to navigate and reach the yellow target in the environment. FV detects this unsafe input configuration where the agent shows a suboptimal behavior as it is stuck in an infinite alternating loop (a). When an obstacle is added, changing the agent's observation, the robot is able to escape from the loop and reach the target (b).

To address such an issue, the research field of Formal Verification (FV) of DNNs [10], has emerged as a valuable solution to provide formal assurances on the safety aspect of these functions before the actual deployment in real scenarios. Broadly speaking, the goal of FV is to prove (or falsify) a desired input-output relationship–aka safety property– for a specific DNN. This decision problem is known as *DNN-Verification* and is typically solved either using interval analysis [11] to propagate the input bound through the network and perform a *reachability analysis* in the output layer [12, 13, 14] or by encoding the linear combinations and the non-linear activation functions of a DNN as constrained for an optimization problem [15, 16].

Despite the considerable advancements made by DNN-verifiers over the years [10], the binary nature (`safe` or `unsafe`) of the result provided by these tools could hide additional helpful information to gain a deeper understanding of these functions. In particular, with the

DNN-Verification, we are able to find a counterexample in case a particular safety property does not hold, however, we do not have any information on how many unsafe configurations violate the safety property for the given DNN. This is a crucial element to rank different trained models or to re-train the model so to avoid such unsafe states. To this end, in [17], the authors proposed a quantitative variant of FV, aiming to provide the total number of violations for given safety property and a specific category of DNNs called Binarized Neural Networks (BNNs). Nonetheless, a potential binarization of a DNN to a BNN typically does not preserve violation points [18], hence, [19] formalized the problem for standard DNNs called *#DNN-Verification*. This novel type of FV allows us to estimate the total number of violations in the domain of the safety property and the probability that a DNN violates a given property. However, given the NP-Completness of the standard *DNN-Verification*, the *#DNN-Verification* has been proven to be #P-Complete [19]. Hence, to solve this challenging problem, both studies in [17, 19], focus on efficient approximate solutions providing provable (probabilistic) guarantees regarding the computed count. The main solutions of FV tools at the state-of-the-art for both *DNN-Verification* and *#DNN-Verification* are strongly based on the Branch-And-Bound (BaB) approach [20]. 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* [14, 21], in this work, we first investigate possible optimizations to solve the *#DNN-Verification* in an exact fashion [22]. Moreover, we employ these optimizations to the recent approximate solution [19] 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) [13, 21], 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 [23] and on realistic robotic DRL mapless navigation domain confirm our hypothesis. In particular, we empirically confirm 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 work 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 [14, 21, 22] 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.

Crucially, this work paves the way for the application of different types of FV even for complex DNNs typically employed in challenging robotic scenarios.

2. Background

Our work 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 [13, 21, 14]. In the following sections, we briefly introduce the *DNN-Verification* and the strategies to solve the problem efficiently. Finally, we provide the formulation on the *#DNN-Verification* and the existing solutions to this quantitative variant of standard FV. We refer interested readers to [13, 21, 14, 22, 19] for more details.

2.1. DNN-Verification

The *DNN-Verification* can be defined in two main ways, either using the satisfiability formulation or the reachability one. In general, the *DNN-Verification* problem takes as input a tuple $\mathscr{T} = \langle \mathscr{N}, \mathscr{P}, \mathscr{Q} \rangle$, where $\mathscr{N}$ is a trained DNN, while $\langle \mathscr{P}, \mathscr{Q} \rangle$ encodes respectively the safety property as input-output relationship. More specifically, $\mathscr{P}$ is a precondition on the input that defines the possible input configuration we are interested in (typically expressed using the cartesian product of hyperrectangles), while $\mathscr{Q}$ encompasses the postcondition that represents the output results we aim to guarantee formally. Hence, the problem consists of verifying that for each input that satisfies $\mathscr{P}$ when fed to the DNN $\mathscr{N}$, the resulting output also satisfies $\mathscr{Q}$ [15, 10, 19].

Referring to Fig. 1, assuming an observation space composed of normalized laser scan values in $[0, 1]$ (where 0 represents the closest distance to an obstacle), a possible high-level "behavioral" safety property in this specific example could be:

$$\mathscr{P} : \text{frontal lidar scans } \in [0.01, 0.05]$$
$$\mathscr{Q} : \text{action } \neq \uparrow \text{ (forward movement)}$$

Typically, what is done by the FV tools is to search for a single counterexample that falsifies the property. Hence, the FV tool searches for the existence of at least a single input configuration that satisfies $\mathscr{P}$ and the negation of the postcondition $\mathscr{Q}$. Referring to the safety property reported above, a FV tool searches for the existence of and input $x \in [0.01, 0.05]$ for which $\mathscr{N}(x) = \uparrow$. If this particular input configuration exists, then the FV returns a `SAT` answer, meaning that there exists at least one violation point, and therefore, the original property does not hold, i.e., the DNN is `unsafe`. Otherwise, the answer is `UNSAT`, so the original property holds for each possible input selected from the intervals encoded in $\mathscr{P}$, and the DNN is provable `safe`.

As previously mentioned, our approach is based on interval analysis. In this context, FV methods propagate directly the intervals that encode the precondition and perform layer-by-layer reachability analysis to compute the output reachable set $\mathscr{R}(\mathscr{X}, \mathscr{N})$, with $\mathscr{X}$ the intervals that encode the property precondition $\mathscr{P}$, and $\mathscr{N}$, our DNN. At the end of this propagation, the

tool checks whether $\mathscr{R}(\mathscr{X}, \mathscr{N}) \subseteq \mathscr{Y}$, where $\mathscr{Y}$ is the desired reachable set. However, given the non-linear and non-convex nature of DNNs, estimating the exact $\mathscr{R}(\mathscr{X}, \mathscr{N})$ is infeasible, and thus, the output reachable set is typically over-approximated. Nonetheless, the naive interval analysis produces large overestimation errors as it ignores the input dependencies during interval propagation [11]. To address this issue, [12] proposes (i) the iterative refinement process and (ii) symbolic interval propagation. Usually, FV methods that exploit these solutions are combined with branching [20] methods and are called *search-reachability* methods [10]. In the next section, we briefly introduce the intuition behind these optimizations.

2.2. Efficient Bound Estimation via Symbolic Interval Propagation and Linear Relaxation

In Fig. 2, 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 [12].

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, [12] 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, the authors 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. Nonetheless, ReLU nodes can demonstrate non-linear behavior for a given input interval. More specifically, if the interval in input to the ReLU node produces

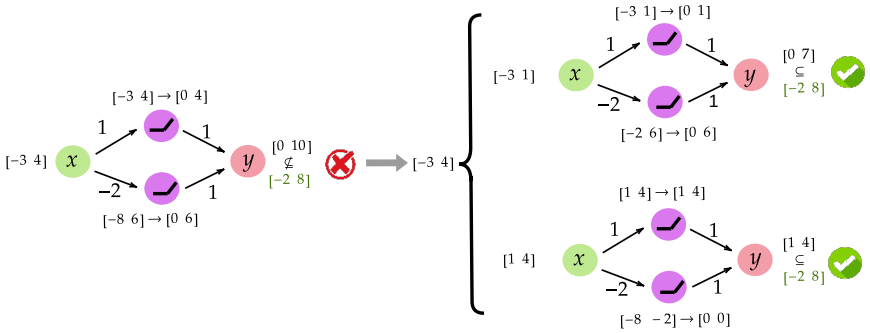


Figure 2: Explanatory image of reachability analysis combined with branching method called *iterative refinement*[12]. Exploiting the interval algebra [11], the lower and the upper bounds of each interval are propagated through the DNN layer-by-layer. In the left part, naive interval analysis produces 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.

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, [13] 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 in the form:

$$z = \left[\frac{u}{u-\ell}Eq_\ell,\ \frac{u}{u-\ell}(Eq_u - \ell)\right] \tag{1}$$

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, 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. 1 on the same ReLU node, we are able to partially preserve the information and thus obtain a tighter output reachable set.

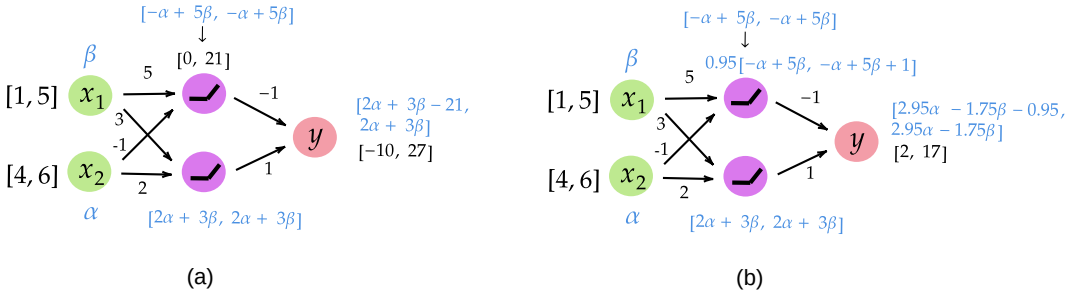


Figure 3: Explanatory image of (a) Symbolic interval propagation proposed in [12] and (b) Symbolic linear relaxation [13].

2.3. #DNN-Verification

The combination of the optimizations seen in the previous sections allows formal verification tools to achieve impressive performance, enabling the ability to verify several safety properties in a fraction of a second [24]. Nonetheless, one of the main limitations of FV lies in the binary nature of the result returned. To address this issue, in [22], the authors propose `ProVe` a novel FV method to compute the so-called *Violation Rate* (VR), a metric to infer the percentage of the property's input area that causes a violation of a given safety specification. Moreover, in a recent work, [19] formalized the *#DNN-Verification*, the counting version of the *DNN-Verification* and proposed, due the #P-completness of the problem, an efficient approximate solution called `CountingProVe`. More specifically, this type of problem seeks to count the provable number of points (or regions) that satisfy (or not) a given safety property allowing more safety-impactful operations such as estimating the probability of incurring unsafe behaviors, ranking the model for its safety level and subsequent selection of the safest one, or even guiding the training of a

DRL agent in a more informed fashion, for instance, minimizing this number over the training process [25, 26].

To solve this problem in a sound and complete fashion, the idea is to formally verify each node of the Branch-and-Bound [20] tree iteratively. Each node of this BaB represents a partition of the property's domain where, for some part of the domain, the property holds and for another one, it does not. Hence, in the leaves of the BaB tree, we have either a situation of a completely safe or unsafe region. We are then able to obtain the total count of violations, computing the total volume of the subinput spaces in the leaves that present complete violations. However, to state whether each node of the BaB is safe or not, there is the necessity of solving a *DNN-Verification* instance, and thus, this approach struggles to scale on real-world scenarios. To address this issue, [19] proposes a novel approximate solution called `CountingProVe` providing provable (probabilistic) guarantees on the count returned. The underlying idea of this approximation is based on maintaining a well-balanced distribution of violation points using a sampling approach during each splitting operation. At the conclusion of this process, the goal is to evaluate only the count within a single leaf, multiplying this count by a factor of 2^s, where s denotes the number of splits performed. However, achieving a perfect balance among these violation points within the BaB tree is unattainable, prompting the authors to propose a probabilistic solution in expectation and calling an exact count only at the end of this process. This approach is more efficient compared to an exact method, as it requires a single invocation of the exact count method on a reduced problem instance.

Nevertheless, the probabilistic nature of point balancing and the requirement for an exact counting method may still present challenges in contrast to recent approximate solutions [25, 27], which provide slightly less precise bounds on the estimated violation rate, but significantly reduce the computation time by orders of magnitude. In particular, in order to estimate the number of states where a binary property holds (i.e., whether the agent commits a violation or not), in these approaches, a sampling of a small subset of states from a potentially infinite one is used, showing that the result obtained from this subset is representative of the original set we are interested in. The theoretical guarantees of these approaches lie either directly from the Chernoff bound [28] or on the statistical prediction of tolerance limits of [29]. Even though these approaches allow us to obtain a result in significantly less computation time, with respect to the methods proposed in this work, they could fail to capture minimal percentage values of the violation rate (as shown in our experiments) and thus could not always be applicable in realistic safety-critical robotic domains.

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

In this section, we present our main contributions. Considering the fact that the #DNN-Verification problem is #P-complete, 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

Algorithm 1 `Exact Count`

Input: $\mathcal{T} = \langle \mathcal{N}, \mathcal{P}, \mathcal{Q} \rangle$
Output: Violation Rate (VR)

$VR \leftarrow 0$, unknown $\leftarrow \mathcal{P}$
while unknown $\neq \emptyset$ **do**
  $\mathcal{X} \leftarrow$ `Pop`(unknown)
  $\mathcal{R} \leftarrow$ `ComputeReachableSet`($\mathcal{X}$, $\mathcal{N}$)
  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`(Area)
  end if
end while
return *VR*

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 [13] 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, even though our approach is easily extendable to different DNNs such as Tanh or Sigmoid, using similar solutions proposed in [21] to deal with different non-linear activation functions.

We report in Alg. 1 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 $\mathcal{Q}$ (as discussed in Sec.2.1). Broadly speaking, we check if the corresponding output reachable set of $\mathcal{X}$ lies in the undesired reachable set $\mathcal{Y}$. 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. 1, described in Sec. 2, to preserve the interdependence with the previous layer, obtaining tight output bounds and thus speeding up the verification process. In the output layer, we check whether the output reachable set $\mathcal{R}(\mathcal{X}, \mathcal{N})$ is contained or not in $\mathcal{Y}$–the output reachable set that encodes the negated postcondition $\mathcal{Q}$– using Moore's interval algebra [11] 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

$\mathscr{R}(\mathscr{X}, \mathscr{N})$ and $\mathscr{Y}$, respectively.

Since we start verifying the negation of the postcondition, if the $\mathscr{R}(\mathscr{X}, \mathscr{N}) \subseteq \mathscr{Y}$ this implies that all $\mathscr{X}$ is unsafe, that is, there exists at least one and possibly an infinite number of violation points[1]. On the other hand, if $\mathscr{R}(\mathscr{X}, \mathscr{N}) \cap \mathscr{Y} = \emptyset$, we can state that there are no violation points in $\mathscr{X}$, 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 $(\mathscr{R}_\ell < \mathscr{Y}_\ell \wedge \mathscr{R}_u \leq \mathscr{Y}_u)$ or the symmetric case, $(\mathscr{R}_\ell \geq \mathscr{Y}_\ell \wedge \mathscr{R}_u > \mathscr{Y}_u)$, we cannot state whether the property is respected or not, and we have to proceed with the iterative refinement process [12]. This process is repeated until, in all the partitions of the input space created with the iterative refinement, we have complete safety or violation.

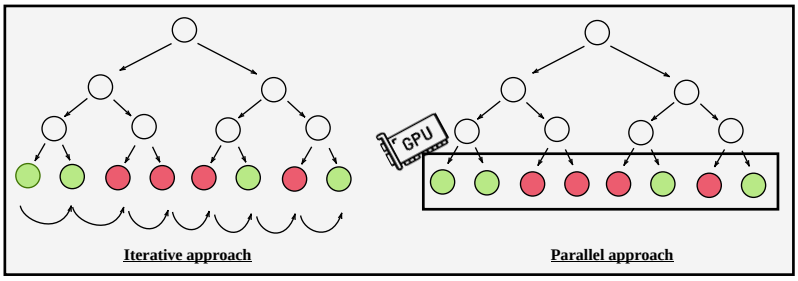


Figure 4: Left: the original approach proposed in [19] for obtain the exact count for the *#DNN-Verification* problem. Each node of the BaB is explored iteratively, thus resulting in poor sample efficiency and scalability as the size of the tree grows. Right: part of the optimization proposed in this work. As shown in [22, 14], we can exploit GPUs to 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. 4, 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., the GPUs, to compute in a parallel fashion the result of the FV verification of each layer of the BaB tree, as shown in [22, 14] and reported in the right Fig. 4. 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.4 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

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

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.

In the next section, we see how the combination of a fast, tight parallel propagation of bounds, and thus, formal verification process, allows us to speed up the existing *#DNN-Verification* tools [22, 19].

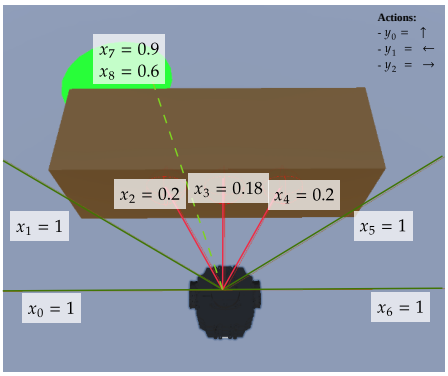
4. Empirical Evaluation

In this section, we empirically analyze the improvement in scalability and efficiency of the methods proposed in this work. 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 in Sec. 3 on `ProVe`[22] and we refer to this new method as `ProVe_SLR` and `CountingProVe`[19] referred as `CountingProVe_SLR`. Moreover, we compare our methods against the `Exact Count` presented in Alg. 1, which use 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. 1, we analyze ACAS Xu[23, 15], 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[2].



$$\mathcal{P} : \forall x_0, x_1, x_5, x_6 \in [0.95, 1], x_2, x_4 \in [0.15, 0.2], x_3 \in [0.15, 0.18], x_7, x_8 \in [0.2, 1]$$
$$\mathcal{Q} : max(y_1, y_2) > y_0$$

Figure 5: Explanatory image of the safety property tested on different models in the second block of Tab 1. Each input $x_0, \dots, x_6$ 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.

We then analyze a set of DRL Mapless Navigation models collected in recent works [9, 25]. In particular, in the second block of Tab.1, 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. We chose discrete actions over continuous ones as discrete controllers can achieve excellent performance in robotic navigation even in multi-agent settings [30, 31].

Finally, in order to test the scalability, in the last block of Tab. 1, 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 these mapless navigation scenarios are behavioral properties, as the one described in Sec.2.1. We provide the reader with a pictorial representation of the property tested in Fig. 5.

Focusing on the first block in Tab. 1, due to the scalability issues discussed in Sec.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.

[2]We refer the interested reader to the original paper [23, 15] for more details on this property.

Instance	Exact methods				Approximations (confidence >99%)			
	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 1
Comparison on different case studies of `Exact Count` without parallelization, `ProVe_SLR` that combines parallelization and SLR, CountingProVe of [19] 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.

A slight minor improvement is also obtained when we use the `CountingProVe_SLR` approximation that uses `ProVe_SLR` as backend over the original `CountingProVe`[19]. Recalling the intuitions of this latter approximation provided in Sec. 2.3, given the 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`.

4.1. Limitation of Sampling Strategies

As discussed in Sec. 2.3, if we are interested in an estimation of the violation rate, there are sampling-based techniques with probabilistic guarantees that allow us to obtain a reasonably accurate result, significantly reducing the computation time by orders of magnitude. However, we argue that in safety-critical contexts such as robotic DRL, where expensive hardware and human lives are potentially at risk, having an estimate of the number of unsafe input configurations may not be sufficient. To this end, we performed an additional test considering a simple DNN with three input nodes, two hidden layers of 32 ReLU nodes, and a single output. We tested the following property:

$$\mathscr{P}: \forall x_0, x_1, x_2 \in [0, 1]$$
$$\mathscr{Q}: y \geq 0$$

Hence, we collected an estimate of the violation rate for this network and this property by employing a Monte Carlo sampling approach using $1M$ samples. The result obtained with this strategy reported a VR = 0%, i.e., no violation points in the input domain. We then perform the formal verification on the same DNN and property, employing `ProVe_SLR`, obtaining instead a VR = 0.00012%. As previously discussed, this result shows that sampling-based approaches are computationally more efficient; still, in the presence of low VR rates, they can fail even with considerable sample sizes. Hence, it is crucial to develop efficient and very accurate approximation methods capable of scaling on larger instances of the problem, capturing even the minute fraction of the violation rate potentially present in the property's domain.

5. Discussion

In this paper, we present 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 work 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`.

Future directions involve developing novel approximation algorithms capable of scaling to considerably larger instances of the problem, being able to provide the violation rate even in complex contexts such as robotic DRL. Moreover, we plan to extend the optimizations proposed in this work to different robotic tasks, even with different input types, such as images.

References

[1] K. O'Shea, R. Nash, An introduction to convolutional neural networks, arXiv preprint arXiv:1511.08458 (2015).

[2] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, S. Levine, Learning complex dexterous manipulation with deep reinforcement learning and demonstrations, arXiv preprint arXiv:1709.10087 (2017).

[3] L. Marzari, A. Pore, D. Dall'Alba, G. Aragon-Camarasa, A. Farinelli, P. Fiorini, Towards hierarchical task decomposition using deep reinforcement learning for pick and place subtasks, in: 2021 20th International Conference on Advanced Robotics (ICAR), IEEE, 2021, pp. 640–645.

[4] D. Corsi, L. Marzari, A. Pore, A. Farinelli, A. Casals, P. Fiorini, 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), IEEE, 2023, pp. 10289–10294.

[5] A. Pore, D. Corsi, E. Marchesini, D. Dall'Alba, A. Casals, A. Farinelli, 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), IEEE, 2021, pp. 4025–4031.
[6] L. Marzari, D. Corsi, E. Marchesini, A. Farinelli, Curriculum learning for safe mapless navigation, in: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, 2022, pp. 766–769.
[7] L. Tai, G. Paolo, M. Liu, Virtual-to-real drl: Continuous control of mobile robots for mapless navigation, in: IROS, 2017.
[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, arXiv preprint arXiv:1312.6199 (2013).
[9] G. Amir, D. Corsi, R. Yerushalmi, L. Marzari, D. Harel, A. Farinelli, G. Katz, Verifying learning-based robotic navigation systems, in: 29th International Conference, TACAS 2023, Springer, 2023, pp. 607–627.
[10] 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 (2021) 244–404.
[11] R. E. Moore, R. B. Kearfott, M. J. Cloud, Introduction to interval analysis, SIAM, 2009.
[12] S. Wang, K. Pei, J. Whitehouse, J. Yang, S. Jana, Formal security analysis of neural networks using symbolic intervals, in: 27th USENIX Security Symposium (USENIX Security 18), 2018, pp. 1599–1614.
[13] S. Wang, K. Pei, J. Whitehouse, J. Yang, S. Jana, Efficient formal safety analysis of neural networks, Advances in Neural Information Processing Systems 31 (2018).
[14] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, 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 (2021) 29909–29921.
[15] G. Katz, C. Barrett, D. L. Dill, K. Julian, M. J. Kochenderfer, Reluplex: An efficient smt solver for verifying deep neural networks, in: International conference on computer aided verification, Springer, 2017, pp. 97–117.
[16] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, et al., The marabou framework for verification and analysis of deep neural networks, in: International Conference on Computer Aided Verification, 2019.
[17] T. Baluta, S. Shen, S. Shinde, K. S. Meel, 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] Y. Zhang, Z. Zhao, G. Chen, F. Song, 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, Springer, 2021, pp. 175–200.
[19] L. Marzari, D. Corsi, F. Cicalese, A. Farinelli, The #DNN-Verification Problem: Counting Unsafe Inputs for Deep Neural Networks, in: International Joint Conference on Artificial Intelligence (IJCAI), 2023, pp. 217–224.
[20] R. R. Bunel, I. Turkaslan, P. Torr, P. Kohli, P. K. Mudigonda, A unified view of piecewise linear neural network verification, Advances in Neural Information Processing Systems 31 (2018).
[21] P. Henriksen, et al., Deepsplit: An efficient splitting method for neural network verification

via indirect effect analysis., in: IJCAI, 2021, pp. 2549–2555.
[22] D. Corsi, E. Marchesini, A. Farinelli, Formal verification of neural networks for safety-critical tasks in deep reinforcement learning, in: Uncertainty in Artificial Intelligence, PMLR, 2021, pp. 333–343.
[23] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, M. J. Kochenderfer, Policy compression for aircraft collision avoidance systems, in: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), IEEE, 2016, pp. 1–10.
[24] S. Bak, C. Liu, T. Johnson, The second international verification of neural networks competition (vnn-comp 2021): Summary and results, arXiv:2109.00498 (2021).
[25] E. Marchesini, L. Marzari, A. Farinelli, C. Amato, Safe deep reinforcement learning by verifying task-level properties, AAMAS '23, International Foundation for Autonomous Agents and Multiagent Systems, 2023, p. 1466–1475.
[26] L. Marzari, E. Marchesini, 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), 2023, pp. 7133–7139.
[27] L. Marzari, D. Corsi, E. Marchesini, A. Farinelli, F. Cicalese, Enumerating safe regions in deep neural networks with provable probabilistic guarantees, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, 2024, pp. 21387–21394.
[28] M. Mitzenmacher, E. Upfal, Probability and Computing: Randomized and Probabilistic Techniques in Algorithms and Data Analysis, Cambridge University Press, 2017.
[29] S. S. Wilks, Statistical prediction with special reference to the problem of tolerance limits, The annals of mathematical statistics 13 (1942) 400–409.
[30] E. Marchesini, A. Farinelli, Enhancing deep reinforcement learning approaches for multi-robot navigation via single-robot evolutionary policy search, in: 2022 International Conference on Robotics and Automation (ICRA), 2022, pp. 5525–5531.
[31] E. Marchesini, A. Farinelli, Centralizing state-values in dueling networks for multi-robot reinforcement learning mapless navigation, in: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, pp. 4583–4588.