

UNIVERSITY OF VERONA

DEPARTMENT OF COMPUTER SCIENCE

GRADUATE SCHOOL OF NATURAL SCIENCES AND ENGINEERING

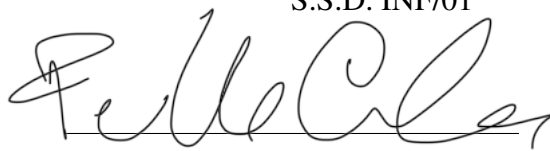
DOCTORAL PROGRAM IN COMPUTER SCIENCE

CYCLE 37

On the construction of useful quantum kernels

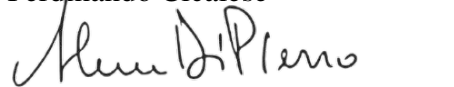
S.S.D. INF/01

Coordinator:



Ferdinando Cicalese

Tutor:



Alessandra Di Pierro

Co-Tutor:



David Windridge

Doctoral Student:



Massimiliano Incudini

© 2025 Massimiliano Incudini. All Rights Reserved.

On the construction of useful quantum kernels - Massimiliano Incudini

Tesi di Dottorato

Verona, April 24, 2025

Sommario

Il crescente interesse nella comunità scientifica e industriale verso l'informatica quantistica ha portato a progressi significativi, rendendo questa tecnologia una scelta promettente per risolvere alcuni problemi computazionali al di là delle capacità dei computer classici. Tuttavia, nonostante questi avanzamenti, la piena comprensione del potenziale dei dispositivi quantistici, sia dal punto di vista teorico che applicativo, rimane ancora lontana. Questa lacuna diventa ancora più evidente quando consideriamo l'applicazione del calcolo quantistico per ottimizzare o accelerare i modelli di *machine learning*.

Numerose proposte sono state avanzate per sfruttare il calcolo quantistico in specifici contesti, come il processamento di matrici sparse, le equazioni differenziali, l'analisi dei dati topologici e l'ottimizzazione combinatoria. Questa tesi, invece, si concentra su un'altra direzione, esaminando le potenzialità delle *kernel machines* implementate su computer quantistici.

Le *kernel machines* sono modelli di apprendimento automatico che generalizzano i modelli lineari, consentendo loro di catturare relazioni tra i dati di grande complessità. Inizialmente proposte per la loro facilità di implementazione sui primi computer quantistici, noti come 'NISQ', queste tecniche sono state oggetto di molteplici esperimenti volti a mostrare i progressi tecnologici dell'hardware stesso, mancando tuttavia di chiaro vantaggio funzionale rispetto alle tecniche classiche. Questa tesi si propone di colmare tale divario.

La prima parte della tesi si concentra sul miglioramento dei kernel per le macchine NISQ. Inizialmente, proponiamo approcci automatici per la definizione dei kernel quantistici tramite procedure di ottimizzazione, eliminando la necessità di definire manualmente trasformazioni per ciascun caso d'uso. Inoltre, esploriamo l'introduzione di non linearità nei modelli e il miglioramento delle prestazioni mediante tecniche di ensemble, rendendo questi primi più simili ai performanti modelli classici di *deep learning*.

La seconda parte della tesi esplora il potenziale del calcolo quantistico fault-tolerant nei metodi kernel. E' noto come sia possibile sfruttare algoritmi quantistici per estrarre proprietà algebriche dei dati, che quindi possono essere usate anche all'interno delle kernel machines. Noi ci concentriamo in particolare sui kernel per l'analisi di grafi e dati topologici. Infine, presentiamo *quask*, il software open source di quantum machine learning sviluppato specificatamente per semplificare il lavoro dei ricercatori nel campo dei quantum kernel.

Abstract

The significant interest from the scientific and industrial community in quantum computing has led to unprecedented progress, making this technology a realistic candidate for addressing problems considered beyond the possibility of any computational device today. Despite these advancements, a comprehensive understanding of the potential of quantum devices remains elusive, both from theoretical and practical standpoints. This gap becomes particularly apparent when considering using quantum computing to enhance or accelerate machine learning models.

Numerous proposals have emerged in recent years, spanning from algorithms for sparse linear algebra and solvers for dissipative partial differential equations to tools for topological data analysis and combinatorial optimization. This thesis departs from these paths by investigating kernel machines whose kernel functions are computed through quantum circuits.

Kernel models expand upon linear models, enabling them to capture intricate relationships within data. Initially proposed in the context of quantum computing as a potential implementation on the first generation of quantum computers, known as ‘NISQ’, numerous experiments have showcased technological advancements without yielding clear applications or performance improvements compared to classical techniques. This thesis seeks to bridge this gap.

The thesis first focuses on improving kernels for NISQ machines. Firstly, we delve into automatic approaches for defining quantum kernels through optimization procedures, thereby circumventing the need to customize transformations for each use case manually. Secondly, we explore the introduction of non-linearity into these models and its limitations. Thirdly, we boost performance using ensemble techniques, which share significant connections with the over-parametrization employed in deep learning models.

The second part of the thesis delves into the potential of fault-tolerant quantum

computing in kernel methods. We can leverage quantum algorithms to extract the algebraic properties of data for our machine learning models. Among these, we concentrate on kernels tailored for graph analysis and topological data. Finally, we discuss open-source machine learning software in quantum sciences, elucidating how our software *quask* accelerates researchers' work in kernels.

Table of Contents

1	Introduction	27
2	Preliminaries	31
2.1	Notation	31
2.2	Classical machine learning	32
2.2.1	Linear regression	33
2.2.2	Kernel regression	36
2.2.3	Support vector machine	38
2.2.4	Neural networks	41
2.3	Quantum machine learning	42
2.3.1	Quantum mechanics	42
2.3.2	Quantum computing	44
2.3.3	Machine learning algorithms for fault-tolerant quantum devices	47
2.3.4	Machine learning algorithms for NISQ devices	49
2.3.5	Quantum advantage in supervised learning	53
3	Automatic and effective discovery of quantum kernels	55
3.1	The Kernel Compatibility	55
3.1.1	Kernel-target alignment	56
3.1.2	Task-model alignment	56
3.2	Challenges in the design of a quantum kernel	58
3.2.1	Expressible quantum kernel cannot learn nearly all functions . .	58
3.2.2	Expressible quantum kernel cannot be calculated precisely . . .	59
3.2.3	Simple quantum kernels can be efficiently simulated	60
3.2.4	Choosing the quantum kernel requires insight on the task	61
3.3	How to control the expressivity of a quantum kernel	61

3.3.1	Projected quantum kernels	62
3.3.2	Bandwidth	63
3.3.3	Symmetries in the data	64
3.4	Automatic procedure to build a quantum kernel	65
3.4.1	Modeling quantum kernels	67
3.4.2	Criteria	69
3.4.3	Optimization	73
3.5	Detection of beyond-Standard Model events	75
3.5.1	Dataset	76
3.5.2	Experimental setup	77
3.5.3	Results	79
4	Toward “deep” quantum machine learning models	83
4.1	Characteristics of deep machine learning models	84
4.1.1	Feature learning	84
4.1.2	Hierarchical feature learning	86
4.1.3	Over-parameterization	86
4.2	Neural Tangent Kernel	89
4.3	Path Kernel	91
4.3.1	Path Kernel as a tool to investigate the role of feature learning .	92
4.3.2	Path Kernel as a tool to investigate the role of hierarchical feature learning	92
4.3.3	The Path Kernel is a Mercer kernel	93
4.3.4	A mathematical caveat in the proof of [90, Theorem 1]	94
4.4	Non-linearity in quantum neural networks	95
4.5	Quantum Path Kernel	96
4.5.1	Experimental evaluation	99
4.5.2	Results	102
4.5.3	Further directions	105
4.6	From over-parameterization to ensembles in NISQ machine learning models	106
4.6.1	Background on ensemble learning	109
4.6.2	Experimental evaluation	111
4.6.3	Results	115
4.6.4	Quantification of the quantum resources for ensemble learning .	119

5	Quantum kernels on graph data	121
5.1	Machine learning on graph data	121
5.2	Graph (dis)similarity measures	123
5.3	Quantum algorithm for the Graph Edit Distance	125
5.3.1	Rephrasing the GED	125
5.3.2	Reduction to GED to QUBO	126
5.3.3	Quantum and classical device for tackling QUBO optimization .	129
5.3.4	Experimental setup	131
5.3.5	Results	134
6	Quantum kernels on topological data	141
6.1	Machine learning on topological data	142
6.2	Topological data analysis on a quantum computer	146
6.2.1	Task	146
6.2.2	Overview of the quantum algorithms	147
6.2.3	Encoding of the p -simplices	148
6.2.4	Preparation of the uniform mixtures of cliques	149
6.2.5	Encoding of the operator	149
6.2.6	Runtime	150
6.2.7	Practical applicability	151
6.3	Topological feature-based quantum kernels	152
6.3.1	Betti curves	152
6.3.2	Betti curves with normalized quasi-Betti numbers	153
6.4	A toy example of topological kernel over normalized quasi-Betti numbers	154
6.4.1	Setup	155
6.4.2	Results	157
6.5	Potential advantage of quantum kernels on combinatorial structures . .	157
7	Application of quantum techniques for combinatorial optimization to epis-	
	tasis detection	161
7.1	Tackling epistasis detection	162
7.2	Functioning of NeEDL	163
7.2.1	Epistasis models	163
7.2.2	Construction of the SNP-SNP interaction network	165
7.2.3	First phase: seeding	165

7.2.4	Second phase: searching	167
7.3	Experimental evaluation	168
7.4	Results	171
8	Open source software in quantum kernels	175
8.1	The jungle of open source software in quantum machine learning	176
8.1.1	PennyLane	176
8.1.2	Qiskit Machine Learning	177
8.1.3	QiBO/Qibolab	177
8.1.4	TensorFlow Quantum	177
8.2	“Make your code count” guidelines	178
8.3	Overview of <i>quask</i>	179
8.3.1	A short example	180
8.4	Functioning of <i>quask</i> : API and architecture	182
8.4.1	The <code>quask.core</code> module	182
8.4.2	The <code>quask.evaluator</code> module	185
8.4.3	The <code>quask.optimizer</code> module	187
8.5	Resources of <i>quask</i> for the community	189
8.5.1	GitHub	190
8.5.2	Documentation and tutorials	192
8.5.3	Homepage and visibility	192
9	Conclusion	197
10	Bibliography	201

List of Figures

2.1	Different kinds of machine learning tasks.	33
2.2	Conjectured view of the complexity classes of classical and quantum computation, with some notable examples of problems belonging to each region.	45
3.1	Overview of the automatic process for constructing a quantum kernel. The process starts with an initial guess, which can be an empty quantum circuit (identity unitary) or an informed choice based on prior knowledge. Next, a cost function is defined based on the training data and established criteria from the literature that provide insights into the statistical properties of the kernel. This cost function becomes the objective of an iterative optimization process; the specific optimization technique is selected by the researcher from the available options. The same criteria used in the cost function can also be employed to evaluate and debug the candidate solution at the end of the process.	67
3.2	The ‘overlap test’ quantum circuit is employed to estimate the inner product of two vectors, $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, which are encoded using the unitary transformation U . The likelihood of observing the string 0^n as the result of the quantum circuit with unlimited accuracy (infinite shots) is evaluated using the formula in Equation 2.73.	68

3.3	The ‘swap test’ quantum circuit is employed to estimate the inner product of two vectors $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}$ that are encoded using the unitary transformation U . The overlap test utilizes a total of $2n + 1$ qubits, as opposed to n qubits, but nevertheless yielding circuits with reduced depth. To calculate the inner product, we can use the method in Equation 2.74 by estimating the chance of obtaining a measurement result of 0 on the topmost qubit. In addition, by utilizing the controlled operation of the swap test on a subset of the n target qubits in each data-related state, we may compute partial inner products that can be understood as a projected quantum kernel (see to Equation 2.73).	69
3.4	Example of a quantum kernel implemented following the proposed approach. The quantum kernel is defined by a feature map that acts on a Hilbert space with a total of 3 qubits. It is composed of 3 operations. Each sample is denoted as $\mathbf{x} \in \mathbb{R}^5$ (just the last two characteristics are utilized in this example). Although the specific values for bandwidth are not mentioned, a typical approach is to assign $\beta_i = i/10$ for $i \in 1, \dots, 10$. This is a projected quantum kernel, where the measurement is conducted on a subset of the three qubits.	70
3.5	Comparison of the performance of the automatically discovered quantum kernels (solid line), the best-performing classical kernel (dashed line, plot on the left), and the best-performing quantum kernel (dashed line, plot on the right) in terms of the ROC-AUC curve over the evolving parameter ν characterizing the one-class SVM. The data points belong to the latent space with $\ell = 8$, and the automatically discovered kernels have quantum circuits with $m = 8$ gates.	80
3.6	Comparison of the performance of the automatically discovered quantum kernels (solid line), the best-performing classical kernel (dashed line, plot on the left), and the best-performing quantum kernel (dashed line, plot on the right) in terms of the ROC-AUC curve over the evolving parameter ν characterizing the one-class SVM. The data points belong to the latent space with $\ell = 8$, and the automatically discovered kernels have quantum circuits with $m = 12$ gates.	80

3.7	Comparison of the performance of the automatically discovered quantum kernels over different configurations of the optimization process on the Narrow G dataset. The performance is stated in terms of AUC-ROC. The left column shows results for $m = 8$, and the right column shows results for $m = 12$. Experiments with RL (SARSA) have been conducted for 25 and 100 iterations (episodes).	82
3.8	Comparison of the performance of the automatically discovered quantum kernels over different configurations of the optimization process on the A→HZ dataset. The performance is stated in terms of AUC-ROC. The left column shows results for $m = 8$, and the right column shows results for $m = 12$. Experiments with RL (SARSA) have been conducted for 25 and 100 iterations (episodes).	82
3.9	Comparison of the performance of the automatically discovered quantum kernels over different configurations of the optimization process on the Broad G dataset. The performance is stated in terms of AUC-ROC. The left column shows results for $m = 8$, and the right column shows results for $m = 12$. Experiments with RL (SARSA) have been conducted for 25 and 100 iterations (episodes).	82
4.1	Layerwise inspection of the patterns activating the neurons of a convolutional neural network. The model has been trained for image recognition tasks. From top to bottom, the features learned by neurons from layers 2 to 5 are shown on the left, and examples of images activating the neurons are shown on the right. Adapted from [81, Figure 2], reused with permission © 2014, Springer International Publishing Switzerland.	87

4.2	Pictorial representation of the double descent phenomenon. In the classical, under-parameterized regime, the test error follows the bias-variance tradeoff rule, which has a sweet spot in model capacity (the model is neither too small nor too big) that exhibits the best performance. At the interpolation threshold, the model can perfectly interpolate the data (the training loss is zero), but in doing so, it overfits. A further increase in model capacity will enter the over-parameterized regime, where the bias-variance tradeoff no longer holds. Any increase in model size will help the optimization process lead, among all the solutions having zero loss, to the one with the smallest norm, thus the simplest function.	89
4.3	Computation of the Path Kernel. <i>Bottom left:</i> A typical parameter trajectory γ is depicted, representing parametric evolution during the training phase. <i>Top left:</i> As θ evolves, it gives rise to differing NTK matrices corresponding to distinct data representations. Such a sequence of matrices thus gives rise to a hierarchical stack of representations in the feature learning regime. <i>Middle:</i> As the training approaches convergence, subsequent matrices become similar, and thus their corresponding representations are correlated. <i>Right:</i> The Path Kernel constitutes the average over these representations.	98
4.4	Instance of variational quantum classifier ($d = 3$ qubits and $L = 2$ layers) contributing to the definition of the machine learning model leading to the QPK machine.	99
4.5	Experimental workflow overview. A random labeled dataset is generated for the Gaussian XOR Mixture classification problem. Subsequently, a variational quantum classifier is trained to compute both the Quantum Path kernel and the baseline kernel (QNTK). Finally, the performance of the Quantum Path kernel is compared with that of the QNTK.	100

4.6	Behavior of quantum machine learning models $f(\mathbf{x}; \boldsymbol{\theta})$ during the training phase. Figure (4.6a) displays the training dataset for the parameters $d = 4, \epsilon = 0.1$. The training loss progression for each of the 20 depthwise models ($L = 1, \dots, 20$) is illustrated in (4.6b). Figure (4.6c) quantifies how much the parameter vector deviates from its initial state. For $d = 4$ and $\epsilon = 0.4$, the corresponding data is shown in (4.6d)-(4.6e)-(4.6f). Similarly, for $d = 4$ and $\epsilon = 1.0$, the relevant information is provided in (4.6g)-(4.6h)-(4.6i).	104
4.7	Test accuracy scores for the Quantum Path Kernel model, the Quantum NTK, and the oracle.. Error bars indicate the standard deviation across three identical experiments with the following parameters: $d = 4, \epsilon = 0.1$ (Figure 4.7a); $d = 4, \epsilon = 0.4$ (Figure 4.7b); $d = 4, \epsilon = 1.0$ (Figure 4.7c).	105
4.8	Training accuracies of the Quantum Path Kernel model, the Quantum NTK, and the oracle. Error bars represent the standard deviation across three identical experiments with specifications $d = 4, \epsilon = 0.1$ (Figure 4.8a); $d = 4, \epsilon = 0.4$ (Figure 4.8b); $d = 4, \epsilon = 1.0$ (Figure 4.8c).	106
4.9	Comparison of testing error as a function of the number of model parameters between a single deep neural network applied to MNIST classification and an ensemble of few (20) deep neural networks trained on different datasets. As expected, the single neural networks exhibit double descent behavior, with their worst performance occurring at the interpolation threshold. Acting as a form of regularization itself, the ensemble can mitigate peaks in loss at the interpolation threshold, leading to a monotonically decreasing error with respect to the number of parameters. Remarkably, the ensemble achieves performance comparable to the largest DNN at the interpolation threshold. This is a schematized and simplified adaptation of the results shown in [24, Figure 2].	108
4.10	Taxonomy of ensemble learning models.	109

4.11	Comparison of bagging (left) and 'standard' boosting (right) techniques. In bagging ensembles, models are trained in parallel on dataset subsets drawn uniformly, with predictions combined using an averaging function. In contrast, boosting ensembles train models sequentially: the first predictor draws samples uniformly, while subsequent models draw samples from a probability distribution biased towards previously misclassified items.	110
4.12	Example of the quantum circuit used for the predictor solving the linear regression dataset; in this case, the number of features is 5, corresponding to the number of qubits. The number of layers is fixed at 1. The input feature vector (x_i) is encoded in the free parameters of the rotational gates in the feature map, while the optimization parameters (θ_i) are encoded in the ansatz.	114
4.13	Performance of ensembles of QNN compared to the performance of their base instances on a noiseless simulator, with respect to the number of layers. (4.13a) refers to the linear regression dataset. (4.13b) refers to the concrete strength dataset. (4.13c) refers to the diabetes dataset. (4.13d) refers to the wine dataset. For regression tasks (4.13a-4.13b-4.13c) the performance is measured in terms of MSE, lower is better. For classification tasks (4.13d), the performance is measured in terms of accuracy, higher is better.	116
4.14	Performance of ensembles of QNN compared to the performance of their base instances on a noisy simulator (using the noise model of the QPU IBM Lima), with respect to the number of layers and over the linear regression dataset only. <i>Left</i> : MSE difference over 5 executions with all Bagging techniques. <i>Right</i> : FM and Bagging models' performance with 80% features in ideal and noisy settings.	118
4.15	Performance of $BAG - 0.8 - 0.2$ ensemble of QNN compared to the performance of their base instances on a QPU IBM Lagos, with respect to the number of layers and the linear regression dataset only. <i>Left</i> : Performance of the bagging model relative to its estimators. <i>Right</i> : Difference in terms of MSE over 10 executions.	119

4.16	Number of qubits and parameters used by the various instances of QNN for different experiments. It is evident how the random subspace technique has been beneficial in reducing the width and length of the circuits across all configurations of 'bagging' ensemble learning tested.	119
5.1	Calculation of a graph edit distance on a pair of small graphs. (a) shows the input and (b) highlights the steps of the smallest graph edit path, in green the additions and red the deletions.	125
5.2	Equivalence in the calculation of the GED between a pair of graphs (a) by calculating the length of a single graph edit path P , made of addition and deletion operations, applied to the first graph (b) or by calculating the sum of lengths of two graph edit path, made of addition only, applied to both graphs (c).	127
5.3	Performance of D-Wave 2000Q quantum annealer (plots a-b-c) and D-Wave Advantage 1.1 (plots d-e-f) in terms of relative difference (a-d), success probability (b-e), high-quality probability (c-f) under different configurations. The standard deviation is pictured as error bars. Legend: SA is simulated annealing, D1 is annealing ($1\mu s \times 10^4$ runs), D20 is annealing ($20\mu s \times 10^4$ runs), D21 is annealing ($20\mu s \times 10^3$ runs), D500 is annealing ($500\mu s \times 10^3$ runs), D100 is annealing ($100\mu s$ paused in the middle and $\times 10^3$ runs).	135
5.4	Performance of simulated annealing (classical baseline) compared to the D-Wave devices in terms of relative difference (a), success probability (b) and high quality probability (c). The standard deviation is pictured as error bars. Legend: SA is simulated annealing, D2000 is D-Wave 2000Q ($20\mu s \times 10^4$ runs), ADV is D-Wave Advantage 1.1 ($20\mu s \times 10^4$ runs), LEAP D-Wave Leap.	136
5.5	Performance of variational algorithms in terms of relative difference (a), success probability (b) and high-quality probability (c). The standard deviation is pictured as error bars. Legend: V1 is VQE ($p = 1$), V3 is VQE ($p = 3$), Q1 is QAOA ($p = 1$), Q3 is QAOA ($p = 3$).	139

5.6	Average Success probability (higher is better), Legend: V: number of vertices, B1: Branch, B2: Branch Fast, B3: Branch Tight, B4: Branch Uniform, B5: Branch Compact, P1: Partition, H1: Hybrid, R1: Ring, A1: ANCHOR_AWARE_GED, W1: Walks, I1: IPFP, B6: BIPARTITE, S1: SUBGRAPH, N1: NODE, R2: RING_ML, B7: BIPARTITE_ML, R3: REFINE, B8: BP_BEAM, SA: Simulated Annealing (GEDLib implementation, not D-Wave's) H2: HED, S2: STAR. Average success probability (higher is better). Legend: V: number of vertices, SA: Simulated Annealing (D-Wave's implementation), D1: annealing $1\mu s \times 10^4$ runs, D20: annealing $20\mu s \times 10^4$ runs, D21: annealing $20\mu s \times 10^3$ runs, D500: annealing $500\mu s \times 10^3$ runs, D100: annealing $100\mu s$ paused in the middle, $\times 10^3$ runs, p : number of repetition of the variational form.	140
6.1	Quantum circuit implementing the LGZ algorithm.	148
6.2	Characteristics of Betti number estimation on a quantum device and its relevance to our experimental setup. (a) Quantum speedup is anticipated for high Betti numbers, potentially depending on n (e.g., $p \in O(\log n)$), and for structures large enough to pose classical computational challenges. (b) Vietoris-Rips Filtration at different threshold values. (c) Experimental setup of our toy example: geometric shapes are generated and sampled to form point clouds. The associated Vietoris-Rips (VR) filtration is analyzed using Betti curves to define the topological kernel. Subsequently, the kernel is evaluated using an SVM against the defined dataset.	155
6.3	Results of the toy example. (a) Accuracy of the various kernels (y) with respect to the number of points sampled from the geometric shapes (x). (b) Root Mean Square Error (RMSE) between the exactly calculated topological kernel and its approximation using weaker Hamiltonian simulation techniques and quantum circuit shots.	158

7.1	Schematization of <i>quepistasis</i> module. The module receives a matrix that contains pairwise relations between SNPs. This matrix is then processed by a preliminary software layer to create a QUBO formulation using the input data. Subsequently, this QUBO instance can be resolved either through the utilization of a thermal annealing algorithm on a classical device or by employing one of the existing quantum devices such as D-Wave, IBM Quantum devices, or the devices accessible via the Azure Quantum cloud, which encompasses IonQ devices. The result consists of one or more sets of Single Nucleotide Polymorphisms (SNPs) that will serve as starting points for the local search process.	168
7.2	Timing performance comparison between Linear NeEDL (randomly chosen seeds) and informed seeding NeEDL with various classical and quantum-based optimization techniques. The negative time represents the duration of the seeding process, while the positive time corresponds to the searching process. Quantum computing enables the generation of a reduced number of high-quality seeds, thereby accelerating the local search phase.	172
7.3	(Projected) scaling of the quantum hardware. The various quantum and classical devices utilized for optimization exhibit distinct scaling patterns, with quantum devices (such as Quantum Annealing and QAOA) demonstrating a steeper slope compared to classical hardware (such as thermal annealing and parallel tempering). This suggests a potential advantage in terms of scaling for quantum hardware.	173
8.1	Structure of <i>quask</i> : The software is written in Python 3 and is built on top of existing quantum frameworks such as Qiskit, PennyLane, and Qibo. The core module of <i>quask</i> implements main objects, such as the ansatz and the kernel, which are directly mapped onto quantum circuits and explicitly embed all the features necessary for expressivity control, such as bandwidth. An even higher-level API is also designed to evaluate and optimize these kernels.	182

8.2	Core module of <i>quask</i> . Its classes map directly to the components a quantum circuit is made of. The <code>Operation</code> class describes a quantum gate, the <code>Ansatz</code> class describes a parametric quantum circuit, and the <code>Kernel</code> class describes the kernel object, which consists of an ansatz and the type of measurement involved. The <code>Kernel</code> class is abstract, with concrete classes being <code>[PennyLane, Qiskit, Qibo, Braket]Kernel</code> , each implementing the object on a different backend. The <code>KernelFactory</code> class determines the actual backend used by providing a <code>create_kernel</code> method that relies on the configured backend.	185
8.3	Evaluator module of <i>quask</i> . The <code>KernelEvaluator</code> interface provides a simple API to quantify the fitness of the kernel for the task at hand. The other classes extend this interface. Notably, it is possible to define ad-hoc criteria by extending this interface and possibly making them depend on the already defined criteria.	187
8.4	Optimizer module of <i>quask</i> . Each optimizer provides a simple API with a single method, as they inherit from the <code>BaseKernelOptimizer</code> interface. This interface's single instantiation can be much more complex and is often based on other Python3 libraries.	189
8.5	Main page of the code repository hosted on GitHub. It shows the stable branch of the code and the latest contributions with dates and authors. A brief summary regarding the management of the repository, including how to install, how to contribute, and how to access other resources, is listed below the code.	191
8.6	Front page of the documentation hosted on ReadTheDocs. It consists of a brief presentation of the library, the motivation behind its design, and its features. On the side, the Getting Started and Installation tutorials are shown along with the pedagogical material.	193
8.7	Home page of <i>quask</i> . The top section of the home page introduces the tool, the motivation behind its implementation, and its features. The bottom section presents tutorials, showcases works based on <i>quask</i> , and credits the contributors.	194

List of Tables

3.1	Comparison of kernel machines using classical kernels, quantum kernels fixed a priori, and quantum kernels chosen via the discovery process in terms of accuracy over a testing set (best results only). Here, the dataset has a latent dimension ℓ . The best-performing quantum kernel in Woźniak et al. [72] uses 2ℓ rotational gates and $\ell - 1$ CNOT gates. The best-performing discovered kernel uses m two-qubit rotational gates, $m \in \{8, 12\}$, and is found after 5 or 10 iterations of BO. Best-performing classifier in bold.	79
4.1	Characteristics of the datasets used to benchmark ensemble models. Note that: 'UCI' refers to the open-source <i>UCI Repository</i> available at archive.ics.uci.edu ; <i>Scikit-Learn</i> is an open-source software library for Python3. The number of features excludes the target variable.	112
4.2	Summary of the ensemble configuration tested.	113
5.1	Average relative difference for the experiments with $\lambda_h = 1, \lambda_s \in \{0.01, 0.05\} \cup \{i/10 \mid i = 1, \dots, 10\}$ with respect to the number of vertices n	133
5.2	Time To Solution for the two hardware configurations, measured in seconds. Legend: n : number of vertices, A: $1\mu s$ per run $\times 10k$ runs, B: $20\mu s$ per run $\times 10k$ runs, C: $20\mu s$ per run $\times 1k$ runs, D: $500\mu s$ per run $\times 1k$ runs, E: $100\mu s$ per run $\times 1k$ runs with paused annealing.	137
5.3	Resource consumption in quantum annealers. Legend: n : number of vertices, L: logical qubits, P: physical qubits, CL: maximum chain length, CS: chain strength.	138
5.4	Resource consumption in variational algorithms. Legend: n : number of vertices, Q: number of qubits, P: parameters, D: depth, S: size.	139

8.1	Some popular quantum computing frameworks and their characteristics. Legend: S: Supports simulation on CPU, E: Supports execution on quantum devices, QML: Includes quantum machine learning capabilities. Domain-specific languages are embedded in the language indicated in parentheses.	195
-----	---	-----

List of publications and preprints

This document builds upon the following works:

- [1] *Computing graph edit distance with algorithms on quantum devices*. M Incudini, F Tarocco, R Mengoni, A Di Pierro, A Mandarino. Quantum Machine Intelligence 4.2 (2022).
- [2] *Resource saving via ensemble techniques for quantum neural networks*. M Incudini, M Grossi, A Ceschini, A Mandarino, M Panella, S Vallecorsa, D Windridge. Quantum Machine Intelligence 5.39 (2023).
- [3] *Quantum Advantage Seeker with Kernels (QuASK): a software framework to speed up the research in quantum machine learning*. F Di Marcantonio, M Incudini, D Tezza, M Grossi. Quantum Machine Intelligence 5.1 (2023).
- [4] *The Quantum Path Kernel: a Generalized Neural Tangent Kernel for Deep Quantum Machine Learning*. M Incudini, M Grossi, A Mandarino, S Vallecorsa, A Di Pierro, D Windridge. IEEE Transactions on Quantum Engineering 4 (2023).
- [5] *Higher-order topological kernels via quantum computation*. M Incudini, F Martini, A Di Pierro. Proceeding of the 2023 IEEE International Conference on Quantum Computing and Engineering (QCE).
- [6] *Toward useful quantum kernels*. M Incudini, F Martini, A Di Pierro. Advanced Quantum Technologies (2024).
- [7] *Automatic and effective discovery of quantum kernels*. M Incudini, D Lizzio Bosco, F Martini, M Grossi, G Serra, A Di Pierro. arXiv preprint arXiv:2209.11144. Accepted into IEEE Emerging Topics in Computational Intelligence (2024).

- [8] *Network medicine-based epistasis detection in complex diseases: ready for quantum computing.* M Hoffmann, JM Poschenrieder, M Incudini, ..., and DB Blumenthal. Nucleic Acids Research 52.17 (2024).

Additional publications not covered in this thesis include:

- [9] *Facial expression recognition on a quantum computer.* R Mengoni, M Incudini, A Di Pierro. Quantum Machine Intelligence 3.8 (2021).
- [10] *Quantum machine learning and fraud detection.* A Di Pierro, M Incudini. Proceeding of Springer “Protocols, Strands, and Logic”, 139-155 (2022).
- [11] *Amplitude-assisted tagging of longitudinally polarised bosons using wide neural networks.* M Grossi, M Incudini, M Pellen, G Pelliccioli. Eur. Phys. J. C 83, 759 (2023).
- [12] *The maximum-likelihood quantum amplitude estimation algorithm provides the best tradeoff between accuracy and circuit depth among quantum solutions for integral estimation.* M Maronese, M Incudini, L Asproni and E Prati. Proceeding of the 2023 IEEE International Conference on Quantum Computing and Engineering (QCE).
- [13] *The Quantum Amplitude Estimation Algorithms on Near-Term Devices: A Practical Guide.* Maronese M, Incudini M, Asproni L, Prati E. Quantum Reports (2024).

Acknowledgment

First and foremost, I would like to thank my wonderful family and friends at home. To my mom, for cheering me up and providing countless lunches, my brother, for the many rides to the airport, and my sister-in-law, for that unforgettable apricot donut. And, of course, a special acknowledgment to my cat, who patiently listened to every practice talk – though I do wish you would stop waking me at 6 a.m. I could not have done this without your unwavering support.

I am deeply grateful to my friends Mike, Sara, Fabio, and Giulio, who have been a source of comfort, laughter, and steadfast encouragement throughout these years – despite not fully understanding what my job is. I'm still not entirely sure either.

I owe great gratitude to my advisors, Alessandra and David, for their patience, guidance, and faith. Your mentorship has been invaluable as I navigated the many challenges of this journey.

I also wish to express my deep appreciation to the scientists who opened the doors to their laboratories to me, offering remarkable opportunities and guidance: Michele Grossi at CERN, Jan Baumbach at the University of Hamburg, and Vedran Dunjko at the University of Leiden. I hold a great debt of gratitude for the trust and support they showed me at pivotal moments.

Finally, to the friends who joined me on this adventure, ranked according to the number of beers we have shared together: Oriel Kiss, Alice Barthe, Paulin de Schoulepnikoff, Sacha Lerch, Enver Savaş, Matteo Robbiati, Nina Hefele, Casper Gyurik, Riccardo Molteni, Riccardo Mengoni, Adrian Solymos, Alessandro Berti, Michele Vischi, Riccardo Romanello, Giacomo Franceschetto, Alessandro Falco, and Nunzia Cerrato.

Thank you all for being part of this experience. I could not have reached this milestone without each and every one of you.

This page has been intentionally left blank.

Chapter 1

Introduction

The mass production and availability of semiconductor-based integrated circuits have welcomed mankind into the third industrial revolution, pushing automatic computation devices into most aspects of our society. However, a paradigm shift is imperative. The cost of enhancing these devices is rising due to technological constraints, such as the physical limitations of transistor miniaturization processes, and political factors, such as the restricted access to advanced processes and materials currently held by a small number of entities worldwide. Alternative approaches to building computational devices are needed, and quantum computing has lately been the most popular one [14].

Originally introduced to simulate quantum processes [15], a task widely believed to be unfeasible for classical computers, the field has undergone a significant revival with the introduction of the first noisy and imperfect prototypes of quantum computers. This renewed interest has led to the discovery of numerous new applications for these devices. These applications extend beyond simulation purposes and indicate potential advantages in terms of speed and energy consumption across various tasks. One of the most important applications is the enhancement of machine learning algorithms, in both runtime and accuracy. This family of approaches is known as quantum machine learning [16, 17] with various applications in chemistry [18], nuclear [19], and high energy physics Incudini et al. [7].

A popular approach to quantum machine learning involves the use of quantum kernels. In classical machine learning, a kernel function generalizes the notion of similarity defined by the Euclidean inner product, allowing for the capture of complex data properties. This task is often performed by defining a feature map, a function mapping the data in the original feature space to another, more complex vector space.

On the other hand, quantum kernels leverage the Hilbert space description of a quantum system’s state to calculate such a function. This technique was developed theoretically by Schuld and Killoran [20], building upon the similarities between the mathematical frameworks of kernel methods and quantum mechanics. However, it has gained even more attention due to experimental work conducted by Havlíček et al. [21] on a 5-qubit superconducting-based quantum computer. To exploit these devices, hybrid techniques are often employed. Implicit methods involve calculating the kernel on a quantum computer and constructing the kernel machine using a classical algorithm, such as the Support Vector Machine. Explicit methods, such as Variational Quantum Classifiers (VQC), implement a quantum program that encodes both the data and a parametric model. We choose the parameter values for these models using a classical optimization procedure, such as stochastic gradient descent. They have proven that classical computation cannot efficiently simulate these quantum programs, paving the way for a potential advantage.

Despite the wide interest in these techniques, little benefit has been found in real-world tasks [22, 23]. This is due to several reasons. From a theoretical standpoint, using quantum transformations as data encoding methods may lead to classifiers that are extremely complicated and unrealistic functions. For these functions, we have both intuitive reasons and analytical proof of their lack of success in generalizing a ‘typical’ target function. The theory of *expressivity* and *expressivity control* aims to study this aspect. From an application perspective, we often lack knowledge about individual problems, thus making it difficult to assess whether any quantum kernel benefits a specific task.

Within this conflicting context, the purpose of this thesis is to identify criteria, structures, and problems for the evaluation, construction, and proficient application of quantum kernels.

Firstly, since each kernel, whether classical or quantum, gives rise to a different class of functions, the choice of the kernel itself is crucial for the model’s success. This choice should be based on the problem’s inductive bias, which comprises a set of assumptions and relationships between the data that are believed to hold within the problem’s dataset. Many decisions regarding quantum kernels are made through educated or random guesses, depending on whether insights into the problem’s structure are available. In contrast, our focus has been on cases where we lack such insights into the problem. We have formulated a set of criteria that determine the usefulness of a quantum kernel in real-world tasks. These criteria include: designing quantum kernels with underlying encodings that are not efficiently simulable using classical computation,

as the lack of this condition renders the use of quantum devices superfluous; defining transformations that are not overly expressible to avoid obtaining a predictor that is a complex and unrealistic function that cannot generalize effectively; and considering and modeling the kernel function to adapt to the dataset’s structure as closely as possible. We have developed an optimization technique that automatically and heuristically identifies a potentially good quantum kernel in cases where we lack prior knowledge about the problem. The algorithm guides the optimization toward kernels satisfying the criteria we have introduced. We tested this technique on one of the few problems for which there is evidence of the benefits of using quantum kernels: anomaly detection in LHCb proton events. The details of this work are presented in Chapter 3.

Secondly, we have tackled certain limitations associated with implementing more powerful models within a quantum framework. Motivated by deep learning’s successes, our goal is to create deep models on quantum devices. However, quantum dynamics are inherently linear, and composing multiple unitary transformations still results in a unitary transformation. This can lead to VQC lacking the hierarchical structure found in deep neural networks. Nevertheless, the theory of Neural Tangent Kernel provides a tool to analyze these quantum models. Such a theory concludes that the VQC is indeed a nonlinear model in parameter space (not in the space of the quantum system’s amplitudes), allowing for mild feature learning. We exploit this characteristic to define the Quantum Path Kernel, which is obtained by integrating the Neural Tangent Kernel at specific parameter instances during the gradient descent training. Each contribution corresponds to a certain data representation. Our motivation stems from the study by Geiger et al. [24], which establishes a connection between the concept of overparameterization, where models have an abundance of parameters that result in a configuration after training that perfectly interpolates the data while being regularized, thus achieving great performance, and ensemble techniques. We suggest using these techniques to investigate the potential for performance improvement. Furthermore, dividing a single large classifier into several smaller ones allows the execution of such models on less demanding hardware, while shorter circuits help minimize quantum noise. We have presented these latter points in Chapter 4.

Building on the evidence we have uncovered, we observed that satisfying these conditions can be quite challenging for NISQ-based quantum kernels and has, so far, been achieved for only a few physics-related problems. However, we have more opportunities when employing fault-tolerant algorithms to construct such mappings. Chapter 5 explores this direction by proposing a quantum algorithm for calculating the

Graph Edit Distance (GED), a metric between graphs that can also be used to derive a kernel function. Calculating the GED is NP-hard, and in general, we can only expect a polynomial speedup compared to classical algorithms. Nonetheless, this task allows us to immediately evaluate our work on NISQ devices and conjecture about the potential advantages of fault-tolerant ones as *heuristics*. We achieved this by formulating the GED as a quadratic optimization problem, which is feasible to run on the D-Wave quantum annealer and, in general, on any NISQ device using the Quantum Approximate Optimization Algorithm. Such a test does not lead to any particular speedup due to the lack of guarantees of NISQ devices; however, the same algorithm executed on fault-tolerant may lead to some speedup. These topics are explored in Chapter 5. Furthermore, our formulation has inspired a quantum algorithm for epistasis detection, a paramount problem in system biology and genetics. Although it doesn't directly relate to kernel machines, it shares intimate connections with the topic of graph edit distances, making it a potential application for our approach. We focus on overcoming the difficulties of dealing with such a large amount of data involved in solving this task and prove it is feasible to tackle such a task on a quantum computer, leaving open the possibility of a quantum advantage whenever a more solid platform becomes available. This topic is explored in Chapter 7.

Then, we explored the definition of kernel methods based on concepts borrowed from topological data analysis, specifically the Betti numbers. The literature has already proposed similar techniques. However, we have modified these approaches to work with the LGZ algorithm, a quantum method used to estimate the low-lying part of the spectrum of the combinatorial Laplacian associated with a certain abstract simplicial complex (a graph and its cliques). Such a technique has implications for data analysis tools such as time series analysis and anomaly detection. The details of this work are presented in Chapter 6.

Finally, a significant amount of effort has gone into designing and implementing a software solution specifically for quantum kernels. This effort is motivated by the need to streamline the large number of experiments that involve repeating the same steps, which can lead to time waste and a higher risk of errors. Our software, named *quask*, is available under an open-source license and is presented in Chapter 8.

Chapter 2

Preliminaries

This chapter briefly reviews some notions in classical machine learning and quantum computation that will be discussed throughout the document. Section 2.1 presents the notation used. Section 2.2 introduces machine learning, with a particular focus on kernel methods. Section 2.3 introduces the formalism of quantum computing and its applications in machine learning.

2.1 Notation

This work uses lowercase alphabetical and Greek characters to indicate single-valued functions ($f, g, h, \dots$) and scalar values ($x, y, w, \alpha, \dots$). Whenever possible, f will be used for target functions, h for predictor functions, and κ will denote a kernel function.

We will also use lowercase boldfaced characters ($\mathbf{x}, \boldsymbol{\theta}, \dots$) to indicate vectors. Furthermore, x_i will denote the i -th component of the vector, and $\mathbf{x}^{(i)}$ will denote the i -th vector of an indexed sequence $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(i)}, \dots$. Uppercase characters will denote matrices and operators. Calligraphic letters, $\mathcal{X}, \mathcal{Y}, \mathcal{H}, \dots$, will denote Hilbert spaces.

Furthermore, we will use $\mathbb{I}$ and $\mathbb{1}$ for the identity matrix and the column vector of 1's, respectively (their dimensions will be clear from the context).

2.2 Classical machine learning

Machine learning develops statistical algorithms aiming to solve certain tasks by means of given data instead of being explicitly programmed¹. These algorithms, named *machine learning models*, can be dichotomized into various categories based on their learning approach; the main categories are supervised learning (e.g., linear regression, support vector machines), unsupervised learning (e.g., PCA, k-means, one-class support vector machine), and reinforcement learning, although more finely grained dichotomies have been proposed in the literature (cf. Figure 2.1). In this thesis, we focus on supervised and unsupervised learning tasks only.

Supervised learning models aim to reproduce the behavior of a target function $\hat{f} : \mathcal{X} \rightarrow \mathcal{Y}$ by selecting an appropriate candidate $\hat{h} \in H$ from a hypothesis class H , based on a set of labeled data $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y}$. Regression and classification are example of supervised learning tasks. We want to select the candidate minimizing the *true risk*,

$$\mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \left[\left(\hat{h}(\mathbf{x}) - \hat{f}(\mathbf{x}) \right)^2 \right] \quad (2.1)$$

Such a quantity is unfeasible to compute in most real-world scenario; however, it can be approximated without bias by the *empirical risk*,

$$\sum_{i=1}^n \left(\hat{h}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2, \quad (2.2)$$

defined the set of given data, denoted as *training set*, with $\mathbf{x}^{(i)}$ is a feature vector sampled i.i.d. from an unknown distribution, $y^{(i)}$ is the corresponding label possibly subject to some noise ϵ , $y^{(i)} = \hat{f}(\mathbf{x}^{(i)}) + \epsilon^{(i)}$. We often consider $\mathcal{X} = \mathbb{R}^d$, although they can correspond to less structured data such as graphs. Regression problems have $\mathcal{Y} = \mathbb{R}$ while classification problems have $\mathcal{Y} = \{1, \dots, n\}$. The training set can be arranged to a matricial representation,

$$\mathbf{X} = \begin{bmatrix} -\mathbf{x}^{(1)} - \\ \vdots \\ -\mathbf{x}^{(n)} - \end{bmatrix} \in \mathbb{R}^{n \times d} \text{ and } \mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \in \mathbb{R}^{n \times 1}. \quad (2.3)$$

¹The definition is commonly attributed to Arthur L. Samuel as part of his algorithmic work on the game of checkers [25].

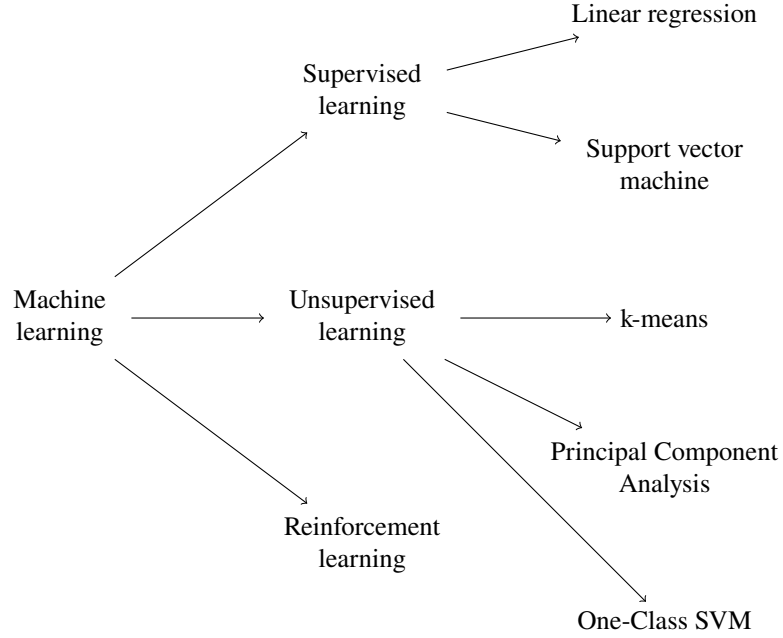


Figure 2.1: Different kinds of machine learning tasks.

The performance of the model can be estimated by calculating the empirical risk on a set of items different from the training set, known as the *testing set*. Failing to differentiate between the training and testing sets results in measuring only the memorization capabilities of the model, rather than its generalization, meaning the ability to predict the output of novel data.

Unsupervised learning models aim to discover patterns in unlabeled data, where the training set is in the form $\{\mathbf{x}^{(i)}\}_{i=1}^n \subset \mathcal{X}$.

2.2.1 Linear regression

Linear regressors are supervised learning models that rely on candidate functions in the hypothesis class

$$H = \left\{ h(\mathbf{x}) = \sum_{i=1}^d w_i x_i = \mathbf{w}^\top \mathbf{x} \mid \mathbf{w} \in \mathbb{R}^d \right\}. \quad (2.4)$$

The learning problem consists of best fitting the data by minimizing the mean square error, leading to the function

$$\hat{h} = \arg \min_{h \in H} \|h(\mathbf{X}) - \mathbf{y}\|^2. \quad (2.5)$$

This can be equivalently rewritten as finding the vector of weights satisfying

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2. \quad (2.6)$$

The solution of this learning problem can be found analytically,

$$\nabla_{\mathbf{w}} [\|\mathbf{X}\mathbf{w} - \mathbf{y}\|] = 0 \quad (2.7)$$

$$\nabla_{\mathbf{w}} [(\mathbf{X}\mathbf{w})^\top \mathbf{X}\mathbf{w} - (\mathbf{X}\mathbf{w})^\top \mathbf{y} - \mathbf{y}^\top (\mathbf{X}\mathbf{w}) + \mathbf{y}^\top \mathbf{y}] = 0 \quad (2.8)$$

$$\nabla_{\mathbf{w}} [\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} - 2(\mathbf{X}\mathbf{w})^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y}] = 0 \quad (2.9)$$

$$2\mathbf{X}^\top \mathbf{X}\mathbf{w} - 2\mathbf{X}^\top \mathbf{y} = 0 \quad (2.10)$$

$$\mathbf{X}^\top \mathbf{X}\mathbf{w} = \mathbf{X}^\top \mathbf{y} \quad (2.11)$$

leading to the weight vector

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \in \mathbb{R}^d. \quad (2.12)$$

When solving this learning problem, calculating the inverse of $\mathbf{X}^\top \mathbf{X}$ can lead to numerical errors if the matrix has nearly singular values. To address this issue, we can add positive values to the main diagonal, reducing the condition number and ensuring a more stable inversion. In this case, the problem becomes

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\| \quad (2.13)$$

with $\lambda \in \mathbb{R}_{\geq 0}$ regularization constant, which is usually chosen heuristically. Such a model is called *Ridge regressor*.

The solution to the Ridge regression problem can be found using the *Lagrangian multiplier method*. This methodology is applied to problems in the following form:

$$\min_{\mathbf{w}} g(\mathbf{w}) \quad (2.14)$$

$$\text{subject to } c_j(\mathbf{w}) = 0, \text{ for } j = 1, \dots, k \quad (2.15)$$

assuming that $g, c_1, \dots, c_k$ are convex. The Lagrangian function is defined as

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}) = g(\mathbf{w}) + \sum_{j=1}^k \alpha_j c_j(\mathbf{w}) \quad (2.16)$$

where the parameters α_j are called Lagrangian multipliers. The solution to this optimization problem can be found by setting the partial derivatives to zero and solving for $\mathbf{w}$ and α . In this case, we can solve two possible optimization problems: $\min_{\mathbf{w}} \max_{\alpha} \mathcal{L}(\mathbf{w}, \alpha)$ or $\max_{\alpha} \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \alpha)$. In general, the solutions to the two optimization problems are different, but under some assumptions, known as the KKT conditions, they coincide. These assumptions hold for Ridge regression.

The learning problem of Ridge regression is not straightforwardly a constraint optimization but can be turned into one with the introduction of some slack (auxiliary) variables $z_j = \mathbf{w}^\top \mathbf{x}^{(j)} - y^{(j)}$ so that we can have the constraint $c_j(\mathbf{z}, \mathbf{w}) \equiv z_j - \mathbf{w}^\top \mathbf{x}^{(j)} + y^{(j)} = 0$ for $j = 1, \dots, n$. By introducing $\mathbf{z}$ in the optimization and the multiplication of the quantity by the term $1/2\lambda$, which is positive and does not change the result, we obtain

$$\arg \min_{\mathbf{w}} \quad \frac{1}{2\lambda} \langle \mathbf{z}, \mathbf{z} \rangle + \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle \quad (2.17)$$

$$\text{subject to} \quad h_j(\mathbf{w}) = 0, \text{ with } j = 1, \dots, n \quad (2.18)$$

whose associated Lagrangian is

$$\mathcal{L}(\mathbf{w}, \mathbf{z}, \alpha) = \frac{1}{2\lambda} \langle \mathbf{z}, \mathbf{z} \rangle + \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle + \alpha^\top (\mathbf{z} - \mathbf{X}\mathbf{w} + \mathbf{y}). \quad (2.19)$$

Solving the optimization problem in the form $\min_{\mathbf{w}} \max_{\alpha} \mathcal{L}(\mathbf{w}, \alpha)$ leads to the solution in the *primal form*,

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X} + \lambda I)^{-1} \mathbf{X}^\top \mathbf{y}, \quad (2.20)$$

for which the model $\hat{h}$ is expressed as a linear combination of its argument's features. However, we can express the model in an equivalent, though interesting, way by solving the *dual form* of the optimization problem, $\max_{\alpha} \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \alpha)$, via

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \alpha^\top \mathbf{X} = 0 \implies \mathbf{w}^\top = \mathbf{X}^\top \alpha. \quad (2.21)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}} = \frac{1}{\lambda} \mathbf{z} + \alpha = 0 \implies \mathbf{z} = -\lambda \alpha. \quad (2.22)$$

Once substituted $\mathbf{w}$ and $\mathbf{z}$, the remaining objective takes the form:

$$\mathcal{L}(\alpha) = \frac{1}{2\lambda} \langle -\lambda \alpha, -\lambda \alpha \rangle + \frac{1}{2} \langle \mathbf{X}^\top \alpha, \mathbf{X}^\top \alpha \rangle + \alpha^\top (-\lambda \alpha - \mathbf{X}\mathbf{X}^\top \alpha + \mathbf{y}) \quad (2.23)$$

$$= \frac{\lambda}{2} \alpha^\top \alpha + \frac{1}{2} \alpha^\top \mathbf{X} \mathbf{X}^\top \alpha - \lambda \alpha^\top \alpha - \alpha^\top \mathbf{X} \mathbf{X}^\top \alpha + \alpha^\top \mathbf{y} \quad (2.24)$$

$$= -\frac{\lambda}{2} \alpha^\top \alpha - \frac{1}{2} \alpha^\top \mathbf{X} \mathbf{X}^\top \alpha + \alpha^\top \mathbf{y} \quad (2.25)$$

$$= -\frac{1}{2} \alpha^\top (\lambda I) \alpha - \frac{1}{2} \alpha^\top \mathbf{X} \mathbf{X}^\top \alpha + \alpha^\top \mathbf{y} \quad (2.26)$$

$$= -\frac{1}{2} \alpha^\top (\mathbf{X} \mathbf{X}^\top + \lambda \mathbb{I}) \alpha + \alpha^\top \mathbf{y} \quad (2.27)$$

$$= -\frac{1}{2} \alpha^\top (G + \lambda \mathbb{I}) \alpha + \alpha^\top \mathbf{y} \quad (2.28)$$

where $G = \mathbf{X} \mathbf{X}^\top$ is the Gram matrix of the inner products between $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$. By setting $\nabla_\alpha \mathcal{L} = -(G + \lambda \mathbb{I}) \alpha + \mathbf{y} = 0$, we obtain the solution,

$$\tilde{\alpha} = (G + \lambda \mathbb{I})^{-1} \mathbf{y}. \quad (2.29)$$

In this case, we express the model with a linear combination of the inner products between the argument of the model and the elements of the training set, instead of as a linear combination of the argument's features.

The primal form (Equation 2.20) exhibits a more favorable scaling for prediction, particularly in scenarios of abundant data, where $n \gg d$. However, the dual form (Equation 2.29) allows us to work with kernel methods, as discussed in the following section.

2.2.2 Kernel regression

When employing linear regression, we rely on specific assumptions, including the linearity of the target function, the independent and identical distribution (i.i.d.) of the data from some distribution, and a constant variance in the noise. If the target function deviates from these assumptions, solving the learning task with a linear classifier becomes challenging.

To create more powerful models, we can transform the data by embedding that to some Hilbert space $\mathcal{H}$ with a richer structure compared by $\mathcal{X}$ according to a feature map $\phi : \mathcal{X} \rightarrow \mathcal{H}$. Once fixed the feature map, we are considering solving the learning problem of selecting candidates from the hypothesis class

$$H = \left\{ h(\mathbf{x}) = \sum_{i=1}^{\dim(\mathcal{H})} w_i \phi(\mathbf{x})_i = \sum_{i=1}^n \alpha_i \langle \phi(\mathbf{x}), \phi(\mathbf{x}^{(i)}) \rangle \right\}. \quad (2.30)$$

A linear model in $\mathcal{H}$, with inner products $\langle \phi(\mathbf{x}), \phi(\mathbf{x}^{(i)}) \rangle$, can capture the same nonlinear relationship in the data introduced with ϕ . It is clear that, for $\dim(\mathcal{H}) \gg 1$, the dual form of regressors in Equation 2.30 is more convenient than the primal one, as the optimization problem is kept n -dimensional and independent from the dimensionality of the Hilbert space. A function in the form

$$\kappa(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle \quad (2.31)$$

is called a *kernel function*. Thanks to Mercer's theorem, any positive semidefinite function κ is a kernel function for some underlying, unknown feature map ϕ and Hilbert space $\mathcal{H}$. This allows us to create complicated but efficiently-computable kernel functions such as the Gaussian kernel,

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{\sigma} \|\mathbf{x} - \mathbf{x}'\|^2\right), \quad (2.32)$$

whose underlying Hilbert space is the infinite-dimensional space $L_2(\mathbb{R}^d)$ and ϕ maps $\mathbf{x}$ to Gaussian functions centered in $\mathbf{x}$ with σ a quantity proportional to the standard deviation. While it is easy to compute Equation 2.32, the corresponding feature map is far from trivial and could be approximated after truncating and discretizing the Gaussians with a large memory expense². The theory of kernel methods can be applied to any similarity-based model. Kernel regressors and classifiers are called *kernel machines*.

Kernel machines, thanks to the representation theorem, continue to guarantee that the learning problem is low (n -) dimensional and convex, leading to an efficient, closed-form solution,

$$\hat{\boldsymbol{\alpha}} = (\phi(\mathbf{X})\phi(\mathbf{X})^\top + \lambda\mathbb{I})^{-1}\mathbf{y} = (K + \lambda\mathbb{I})^{-1}\mathbf{y} \quad (2.33)$$

where $(K)_{ij} = \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ is the *Kernel gram matrix*, and the kernel machine is in the form

$$\hat{h}(\mathbf{x}) = k(\mathbf{x})^\top \hat{\boldsymbol{\alpha}} = k(\mathbf{x})^\top (K + \lambda\mathbb{I})^{-1}\mathbf{y}, \quad k(\mathbf{x}) = \begin{bmatrix} \kappa(\mathbf{x}, \mathbf{x}^{(1)}) \\ \vdots \\ \kappa(\mathbf{x}, \mathbf{x}^{(n)}) \end{bmatrix}. \quad (2.34)$$

Every kernel function defines a unique Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_k$, which is a vector subspace of functions from $\mathcal{X} \rightarrow \mathcal{H}$. The inner product and norm

²It is worth noting that some techniques have been proposed in the literature to approximate a kernel function with a low-dimensional feature map, such as Random Fourier Features [26]. If such a representation is found, we can eliminate the dependency on the data and have a regression model employing a powerful kernel, as allowed in the dual model of the regressor, while keeping the favorable scaling of the primal mode.

of this space are denoted as $\langle \cdot, \cdot \rangle_\kappa$, and $\|\cdot\|_\kappa$, respectively. Such space is characterized by the fact that every evaluation functional, denoted with $\kappa_x : \mathcal{H}_\kappa \rightarrow \mathbb{R}$ is a linear bounded functional such that the reproducing property holds, meaning $\kappa_x[f] = f(x)$ for all $f \in \mathcal{H}_\kappa$ and $x \in \mathcal{X}$. Therefore,

$$\mathcal{H}_\kappa = \overline{\text{span}} \{ \kappa_x \mid x \in \mathcal{X} \}, \quad (2.35)$$

where the span function allows us to obtain a pre-Hilbert space, essentially a vector space with an inner product, while $\overline{\text{span}}$ denotes the closure of the span, ensuring that the space is a proper Hilbert space and satisfies the completeness property. We can decompose a kernel function in terms of an orthonormal (in L_2) set of functions $\{\phi_i\}_{i=1}^\infty$, solution of the eigenvalue problem

$$\int_{\mathcal{X}} \kappa(x, z) \phi_i(z) dp(z) = \lambda_i \phi_i(x) \quad (2.36)$$

where $\{\lambda_i\}_{i=1}^\infty$ is the set of corresponding real, non-negative eigenvalues, sorted in descending order, and p probability distribution over $\mathcal{X}$. The kernel function is then expressed as

$$\kappa(x, x') = \sum_{i=1}^{\infty} \lambda_i \phi_i(x) \phi_i(x'). \quad (2.37)$$

2.2.3 Support vector machine

Consider a classification problem with labels $y^{(j)} = \pm 1$ and assume the dataset is linearly separable. In such cases, a hyperplane exists, denoted by $\mathbf{w}^\top \mathbf{x} + b = 0$, where $\mathbf{w}$ is the normal vector to the hyperplane, and b is the offset from the origin, that separates the two classes perfectly. We can define two parallel hyperplanes such that all elements of one class, those with $y^{(j)} = 1$, remain in the region $\mathbf{w}^\top \mathbf{x}^{(j)} + b > 1$, while all others, with $y^{(j)} = -1$, stay in the region $\mathbf{w}^\top \mathbf{x}^{(j)} + b < -1$. These conditions can be merged into a single inequality:

$$y^{(j)}(\mathbf{w}^\top \mathbf{x}^{(j)} + b) \geq 1. \quad (2.38)$$

Among all the possible pairs of parallel hyperplanes, we aim to select the best one, which is the one that maximizes the margin between the two classes. Since the distance between the two hyperplanes is $2/\|\mathbf{w}\|$, the problem can be formulated as maximizing

this distance:

$$\arg \min_{\mathbf{w}, b} \langle \mathbf{w}, \mathbf{w} \rangle \quad (2.39)$$

$$\text{constrained to } y^{(j)}(\mathbf{w}^\top \mathbf{x}^{(j)} + b) \geq 1, \text{ with } j = 1, \dots, n \quad (2.40)$$

Once the weight vector $\mathbf{w}$ has been found, the model, which is called Support Vector Machine (SVM), takes the form:

$$\hat{h}(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(j)} + b). \quad (2.41)$$

We can solve the learning problem using the method of the Lagrangian multipliers,

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}) = \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle - \sum_{j=1}^n \alpha_j \left[y^{(j)} (\mathbf{x}^{(j)} \mathbf{w} + b) - 1 \right]. \quad (2.42)$$

We solve the dual form of this Lagrangian. Setting $\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0$ leads to $\mathbf{w} = \sum_{j=1}^n \alpha_j y^{(j)} \mathbf{x}^{(j)}$. Similarly, $\frac{\partial \mathcal{L}}{\partial b} = 0$ leads to $\sum_{j=1}^n \alpha_j y^{(j)} = 0$. Finally,

$$\begin{aligned} & \frac{1}{2} \left\langle \sum_{j=1}^n \alpha_j y^{(j)} \mathbf{x}^{(j)}, \sum_{k=1}^n \alpha_k y^{(k)} \mathbf{x}^{(k)} \right\rangle - \sum_{j=1}^m \alpha_j \left[y^{(j)} \left(\mathbf{x}^{(j)} \left(\sum_{k=1}^n \alpha_k y^{(k)} \mathbf{x}^{(k)} \right) + b \right) - 1 \right] \\ &= \frac{1}{2} \sum_{j,k=1}^m \alpha_j \alpha_k y^{(j)} y^{(k)} \langle \mathbf{x}^{(j)}, \mathbf{x}^{(k)} \rangle - \sum_{j,k=1}^n \alpha_j \alpha_k y^{(j)} y^{(k)} \langle \mathbf{x}^{(j)}, \mathbf{x}^{(k)} \rangle - \sum_{j=1}^m \alpha_j y^{(j)} b + \sum_{j=1}^n \alpha_j \\ &= -\frac{1}{2} \sum_{j,k=1}^m \alpha_j \alpha_k y^{(j)} y^{(k)} \langle \mathbf{x}^{(j)}, \mathbf{x}^{(k)} \rangle + \sum_{j=1}^n \alpha_j \end{aligned} \quad (2.43)$$

The solution for $\boldsymbol{\alpha}$ can be found using a quadratic optimization solver.

The points lying on the hyperplanes are referred to as *support vectors*, and they correspond to nonzero coefficients α_j . These support vectors are the only points that contribute to the prediction. Due to the optimization problem we have defined, which prefers $\alpha_j = 0$ as the optimal situation, the solution tends to be sparse. In other words, SVMs typically have only a few support vectors. This characteristic enhances the efficiency of predictions, particularly when compared to linear regression.

In situations where the data is not linearly separable or contains noise, achieving perfect separation with a single hyperplane is not possible. In such cases, a classification technique that accommodates a certain degree of error can be employed. This variant of

SVM is referred to as the *soft margin SVM*, in contrast to the previous version known as the *hard margin SVM*. The objective function is reformulated to include a set of slack variables:

$$\xi_j = \max(0, 1 - y^{(j)}(\mathbf{w}^\top \mathbf{x}^{(j)} + b)) \quad (2.44)$$

Here, for $\xi_j = 0$, it indicates that the j -th point has been correctly classified and lies beyond its class's margin. If $0 < \xi_j < 1$, it means the point is correctly classified but falls between its margin and the middle hyperplane separating the two margins. When $\xi_j > 1$, the point is misclassified. The problem can be reformulated as

$$\arg \min_{\mathbf{w}, b, \xi \geq 0} \langle \mathbf{w}, \mathbf{w} \rangle + C \sum_{j=1}^n \xi_j \quad (2.45)$$

$$\text{subject to } y^{(j)}(\mathbf{w}^\top \mathbf{x}^{(j)} + b) \geq 1 - \xi_j \text{ for } j = 1, \dots, n \quad (2.46)$$

Here, $C \in [0, \infty)$ is a parameter that trades off between classification accuracy and margin size. For $C = \infty$, every error, no matter how small, results in a solution with an infinite cost, effectively recovering the hard-margin SVM. This formulation can still be solved using quadratic optimization techniques. This form is called C-SVM. The dual form is identical to the hard-margin SVM, except for the additional constraint $0 \leq \alpha_j \leq C$.

We can reformulate the C-SVM using a normalized variant of its constant, denoted as ν and lying in the range $[0, 1)$. The value ν is always lower than *fraction of support vector* and higher than *the fraction of margin errors*. The formulation in its primal form is:

$$\arg \min_{\mathbf{w}, b, \xi \geq 0, \rho > 0} \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle - \nu \rho + \frac{1}{2} \sum_{j=1}^n \xi_j \quad (2.47)$$

$$\text{subject to } y^{(j)}(\mathbf{w}^\top \mathbf{x}^{(j)} + b) \geq \rho - \xi_j \text{ for } j = 1, \dots, n \quad (2.48)$$

Note that ρ substitutes 1 in the constraint, meaning the margins are not distant $2/\|\mathbf{w}\|$ anymore, but they are distant $2\rho/\|\mathbf{w}\|$. The nu-SVM is equivalent to a C-SVM with $C = 1/\nu$.

The primary use of SVM is tackling binary and n -ary classification tasks. However, variants of this model have been defined as being employed for unsupervised learning tasks. An example of such variants is the *one-class SVM* (OC-SVM). OC-SVM has been successfully used to solve *anomaly detection*.

The primary form of the optimization problem is

$$\arg \min_{\mathbf{w}, b, \xi \geq 0, \rho > 0} \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle - \rho + \frac{1}{\nu d} \sum_{j=1}^n \xi_j \quad (2.49)$$

$$\text{subject to } (\mathbf{w}^\top \mathbf{x}^{(j)} + b) \geq \rho - \xi_j \text{ for } j = 1, \dots, m, \quad (2.50)$$

where d is the dimensionality of the feature vectors. Note that the $y^{(j)}$ terms have been omitted from the constraints, as the problem is unsupervised. The dual form of the Lagrangian is:

$$\arg \min_{\substack{0 \leq \alpha_j \leq \frac{1}{\nu d} \\ \sum_j \alpha_j = 1}} \frac{1}{2} \sum_{j,k=1}^m \alpha_j \alpha_k \langle \mathbf{x}^{(j)}, \mathbf{x}^{(k)} \rangle, \quad (2.51)$$

whose solution is

$$\hat{h}(\mathbf{x}) = \text{sign} \left(\sum_{j \in SV} \alpha_j \langle \mathbf{x}, \mathbf{x}^{(j)} \rangle - \rho \right). \quad (2.52)$$

where SV is the set of support vectors. We can use a kernel κ instead of $\langle \cdot, \cdot \rangle_{\mathbb{R}^d}$

2.2.4 Neural networks

A neural network with ℓ layers, $h^{(\ell)}$ is a model defined recursively via the equation

$$h^{(0)}(\mathbf{x}, \mathbf{w}_1, \dots, \mathbf{w}_\ell) = \mathbf{x} \in \mathbb{R}^{n_0} \quad (2.53)$$

$$h^{(j+1)}(\mathbf{x}, \mathbf{w}_1, \dots, \mathbf{w}_\ell) = \sigma(\langle h^{(j)}(\mathbf{x}, \mathbf{w}_1, \dots, \mathbf{w}_\ell), \mathbf{w}_j \rangle) \in \mathbb{R}^{n_j} \quad (2.54)$$

where n_j is the number of neurons of the j -th layer, $n_0 = d$ is the input size, and, without loss of generality, we fix the output size to $n_\ell = 1$. Furthermore, σ is the activation function applied elementwise to each element of the vector in input; some notable examples of activation functions are $\tanh$ and $\text{RELU}(\max 0, x)$. In Equation 2.53 we have absorbed the bias values b within the weights $\mathbf{w}$.

The neural network differs significantly from the models we have previously encountered. While linear regression, kernel regression, and SVM are linear models, most neural networks are nonlinear. This also implies that the training is non-convex, providing no guarantees on the efficiency of the process or the optimality of the solution found.

The training of a neural network typically involves stochastic gradient descent.

Initially, random weights are chosen, and the gradient of the loss is calculated using:

$$\mathbf{w} = -\eta \nabla \mathcal{L}(\mathbf{w}), \quad (2.55)$$

where $\mathcal{L}(\mathbf{w})$ represents the loss function (e.g., mean square error as shown in Equation 2.2), and $\eta \ll 1$ is the training step. Multiple training processes starting from different initial guesses or evolving along different paths will likely result in different suboptimal solutions.

Neural networks can be analyzed by approximating them as a linear model. If all layers except the last one are untrained and fixed to their initial random values, the model becomes a kernel machine. The feature map ϕ corresponds to the constant function, mapping the input to the output of the penultimate layer.

Another approach involves truncating the Taylor expansion of the neural network in Equation 2.53 to the first term, resulting in a linear model expressed as a kernel machine, with the kernel named Neural Tangent Kernel (NTK):

$$\kappa_{\text{ntk}}(\mathbf{x}, \mathbf{x}') = \nabla_{\boldsymbol{\theta}} h(\mathbf{x}; \boldsymbol{\theta}) \cdot \nabla_{\boldsymbol{\theta}} h(\mathbf{x}'; \boldsymbol{\theta}), \quad (2.56)$$

Here, $\boldsymbol{\theta}$ represents a fixed value for the model's weights, often the value at initialization. Jacot, Gabriel, and Hongler [27] demonstrated that in certain cases, neural networks are extremely easy to train, experiencing an exponentially fast decrease in loss and independence from the choice of the initial point. These cases are referred to as the large-width regime, occurring when the network is much wider than deeper ($n_j \gg \ell$ for all $j = 1, \dots, \ell$). The wider the network, the closer it is to a linear model, making its training convex. This special regime allows the study of neural networks using kernel methods, providing upper bounds on the error. Further studies have also indicated that, in many contexts, the performance of a neural network and its corresponding NTK, even in finite width, may not diverge significantly. This raises questions about whether such models are genuinely more powerful than kernel methods in many situations.

2.3 Quantum machine learning

2.3.1 Quantum mechanics

The framework of quantum mechanics is described by four postulates.

The first postulate states that a quantum system is represented by a complex Hilbert space, denoted as $\mathcal{H}$, with an inner product $\langle \cdot | \cdot \rangle$. In the Dirac notation, a *unitary* vector in $\mathcal{H}$ is denoted by $|\psi\rangle$ (read as a “ket”), with the additional condition that two vectors differing only by a global phase correspond to the same physical state. Any linear functional over $\mathcal{H}$ corresponds to an element of the dual space $\mathcal{H}^*$, denoted with $\langle\psi|$ (read as a “bra”).

Throughout this document, we will focus only on finite-dimensional spaces, $\mathcal{H} \equiv \mathbb{C}^m$, for which kets and bras are interpreted in terms of column and row vectors, $\langle\psi| = (|\psi\rangle)^\dagger$. The simplest system is the *qubit*, $|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \in \mathbb{C}^2$, with $|\alpha|^2 + |\beta|^2 = 1$, uniquely described up to an (irrelevant) global phase $e^{i\theta}$.

The second postulate states that any *closed* quantum system, subject to no interaction outside the system itself, evolves according to Schrödinger’s equation (a linear ODE), $i(d|\psi\rangle/dt) = H|\psi\rangle$, where H is a self-adjoint (Hermitian) operator known as the Hamiltonian of the system. The solution to this equation is a unitary operator denoted as $U = \exp(-itH)$ for some real constant t . The unitarity of the evolution implies its reversibility.

The space of unitary operators over $\mathcal{H}$, denoted as $\mathcal{U}(\mathcal{H})$, forms a Lie group which, in the finite-dimensional case, is represented by the group of $m \times m$ unitary matrices, $\mathcal{U}(m)$. Consider the Lie algebra of skew-Hermitian operators, denoted by $\mathfrak{u}(\mathcal{H})$, which is equipped with the bilinear Lie bracket operation $[X, Y] = XY - YX$. In the finite-dimensional case, this algebra is represented by the set of $m \times m$ skew-Hermitian matrices, $\mathfrak{u}(m)$. The Lie algebra and its group are related through the exponential map $\exp : \mathfrak{g} \rightarrow \mathcal{G}$; in the case of matrix Lie groups this corresponds to the matrix exponential $\exp(X) = \sum_{k=0}^{\infty} X^k/k!$.

The third postulate defines a quantum measurement. A finite POVM (Positive Operator-Valued Measure) is described by an indexed collection of positive semi-definite Hermitian operators on $\mathcal{H}$, denoted by $M = \{\mu_i\}_{i=1}^s$, where the elements sum up to the identity. Each element μ_i is associated with a measurement outcome labeled by i . During measurement, the process is probabilistic and destructive. The outcome i is observed on the measurement device with a probability of $p(i) = \langle\psi|\mu_i|\psi\rangle$. Subsequently, the quantum state collapses according to $|\psi\rangle \mapsto \mu_i |\psi\rangle / p(i)$.

A *projective measurement* is characterized by the property $\mu_i^2 = \mu_i$, and in such cases, each element is denoted as Π_i . An *informationally complete* POVM has at least $s \geq m^2$ elements and spans the space of self-adjoint linear operators on $\mathcal{H}$. Measurement in the computational basis is defined as $M = \{|i\rangle\langle i|\}_{i=1}^m$.

An *observable* is represented by a Hermitian operator A , and its expectation value for the state $|\psi\rangle$ is denoted as $\langle\psi|A|\psi\rangle$. For finite-dimensional systems, we can (non-uniquely) define a one-to-one correspondence between a POVM measurement and an observable.

The last postulate is about how to compose systems. It states that multiple quantum systems are combined through the tensor product operation. This makes the Hilbert space dimensionality grow exponentially with the number of subsystems, in contrast with classical systems that are combined via the Cartesian product, and explains why it is so challenging to simulate a quantum system with a classical computer. A state defined on $\mathcal{H}_A \otimes \mathcal{H}_B$ is separable if it can be written in the form $|\psi\rangle = |\psi_A\rangle \otimes |\psi_B\rangle$; otherwise, it is entangled.

Formalism of the Density matrices

Another important framework used throughout the document is the density matrix formalism. A density operator and its representation, the density matrix, are probabilistic mixtures of states in $\mathcal{H}$, given by $\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| \in L(\mathcal{H})$. $L(\mathcal{H})$ is the space of linear operators on $\mathcal{H}$. A pure state can be written as $\rho = |\psi\rangle\langle\psi|$; otherwise, the state is mixed. Note that mixed states are statistical mixtures of pure states. The time evolution is described by the Liouville-Von Neumann equation, $i\partial\rho/\partial t = [H, \rho]$, which leads to the evolution $\rho \mapsto U^\dagger \rho U$. The expectation value of the Hermitian operator A is defined as $\text{Tr}[A\rho]$, while for a POVM $M = \{\mu_i\}_{i=1}^s$, we have $p(i) = \text{Tr}[\mu_i^\dagger \rho \mu_i]$ and the state collapses to $\rho' = \mu_i^\dagger \rho \mu_i / p(i)$. If the measurement outcome is unknown the density operator is described as $\sum_i \mu_i^\dagger \rho \mu_i$. The *partial trace* of a state $\rho \in L(\mathcal{H}_A \otimes \mathcal{H}_B)$ is denoted as $\rho' = \text{Tr}_B[\rho] \in L(\mathcal{H}_A)$.

The most general type of evolution allowed in this formalism is the *quantum channel*, which is a completely positive, trace-preserving map (CPTP) $\Psi : L(\mathcal{H}_A) \rightarrow L(\mathcal{H}_B)$, where $\mathcal{H}_A$ and $\mathcal{H}_B$ are Hilbert spaces. A CPTP can encompass various operations, such as unitary evolution, partial trace, and even a POVM measurement.

2.3.2 Quantum computing

Quantum circuit model

A quantum computation can be described in terms of several computational models, the most widely used being the quantum circuit model. In analogy with classical

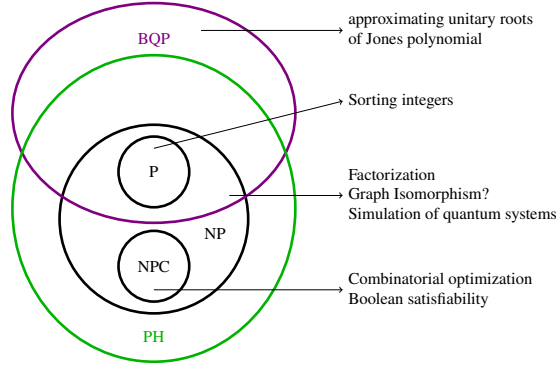


Figure 2.2: Conjectured view of the complexity classes of classical and quantum computation, with some notable examples of problems belonging to each region.

circuits, an n -qubit system is depicted with an n -wire, musical score-like diagram whose time evolution goes from left to right. The system is initialized by convention to $|0\rangle^{\otimes n}$. A sequence of quantum gates, acting non-trivially on a subset of the available qubits, is sequentially placed on the circuit. The fan-in, the number of input wires, has to match the fan-out, the number of output wires. The computation terminates with the measurement, which is often performed as a projective measurement over the computational basis, $M = \{|i\rangle\langle i|\}_{i=0}^{2^n-1}$, and the circuit is executed finitely many times to estimate the distribution of each outcome.

Some notable quantum gates include $\sigma_x = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\sigma_y = Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$, $\sigma_z = Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$, their exponentiation $R_\alpha(\theta) = \exp(-i\frac{\theta}{2}\sigma_\alpha)$ for $\alpha \in \{x, y, z\}$, the identity gate $\sigma_0 = I$, the Hadamard gate $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$, the phase gate $S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$, the T-gate $T = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{pmatrix}$, the controlled-NOT gate $\text{CNOT} = |0\rangle\langle 0| \otimes \sigma_0 + |1\rangle\langle 1| \otimes \sigma_x$.

Complexity-theoretical motivation for the quantum computation

The potential benefit of employing quantum computation resides in the ability to solve certain problems faster than classical computation. Evidence of this potential can be found in complexity theory.

A decision problem L belongs to the complexity class P if there is a uniformly polynomial circuit family $C = \{C_n\}_{n \in \mathbb{N}}$ such that for every $x \in L$ of size n , $C_n(x) = 1$, and for every $x \notin L$ of size n , $C_n(x) = 0$. The use of quantum computing may allow us to solve a larger class of problems. A decision problem L is in the class BQP if there exists a uniformly polynomial family of quantum circuits $C = \{Q_n\}$ such that for all $x \in \{0, 1\}^n$, the probability that $Q_n(x) = 1$ is at least $\frac{2}{3}$ if $x \in L$, and at most $\frac{1}{3}$ if

$x \notin L$. To prove that certain problems can be solved on a quantum computer while being unfeasible on a classical one, we sometimes formulate them in the query model, where the input is provided as a black-box oracle. This model is used because it is difficult to prove lower bounds on the complexity of computing some functions over explicit input data. For example, distinguishing a constant function from a balanced one (under the promise that the function belongs to either of these families) requires exponentially many (in n number of bits of the function) deterministic queries while only a single quantum query is required using Deutsch-Jozsa's algorithm. Furthermore, Shor's algorithm has demonstrated the efficiency of quantum computers in solving the integer factorization problem which is believed to not be solvable in polynomial time on a classical computer, having an exponential speedup over the best classical algorithm.

Some problems are believed not to be able to exhibit an exponential speedup on a quantum computer. That is the case for black-box optimization problems: the lower bound is $O(\sqrt{N})$ for N number of elements in the search space, which is matched by Grover's algorithm. BQP-complete problems, such as (sparse) matrix inversion and the estimation of the unitary root of the Jones polynomial, encompass the full power of quantum computation. It has been proven that a problem exists in $\text{BQP} \setminus \text{PH}$, where PH is the polynomial hierarchy. The relationships between these classes are depicted in Figure 2.2.

The term *quantum supremacy* refers to problems for which there is a formal proof of an exponential speedup guaranteed by a quantum algorithm compared to any classical algorithm. The term *quantum advantage* refers to an exponential speedup of a quantum algorithm compared to the best-known classical algorithm for a particular problem.

Fault tolerant and near-term quantum computers

The current generation of quantum computers does not behave exactly as postulated in quantum mechanics. Various sources of error can affect computations, and the nature of these errors varies significantly depending on the specific physical platform being used. These devices are referred to as NISQ (Noisy Intermediate-Scale Quantum) systems. Notably, one of the primary sources of error is decoherence, which is the tendency of a quantum state to degrade into a classical state. Further limitations arise from the characteristics of specific hardware platforms, such as the available gate set and qubit topology (connectivity). These limitations make it necessary the *transpilation* of abstract quantum circuits, meaning they are converted to equivalent circuits that are optimized

for the given platform’s behavior and potentially add a significant overhead.

According to the *Threshold Theorem*, when a platform achieves a physical error rate below a certain threshold, it can suppress the logical error rate to arbitrarily low levels through the application of quantum error correction schemes. However, these error correction schemes may entail a significant overhead in terms of both qubits and quantum gates. The decoding of the physically measured results (involving many qubits) to obtain logical results also adds overhead in terms of the required classical computation. It might happen that for certain algorithms, despite guaranteeing a significant speedup in terms of query complexity, such an advantage does not outweigh the overhead of error correction upon a more detailed (non-asymptotic) analysis.

2.3.3 Machine learning algorithms for fault-tolerant quantum devices

Sparse matrix inversion

Harrow, Hassidim, and Lloyd [28] proposed the HHL algorithm to solve linear systems of equations. This approximates a solution $\mathbf{x}$ such that $\mathbf{Ax} = \mathbf{b}$ with a real $n \times n$ sparse matrix $\mathbf{A}$ and a vector $\mathbf{b}$. The algorithm runs in time logarithmic in the number of equations (n) and linear in the condition number of $\mathbf{A}$, denoted as $\kappa = |\lambda_{\max}|/|\lambda_{\min}|$. This quantum approach brings about exponential speed-up compared to the best-known classical algorithm. Nonetheless, this advantage holds under strict conditions [29]. Each computational step must occur in logarithmic depth, including loading the vector $\mathbf{b}$, exponentiating the sparse matrix $\mathbf{A}$, and approximately sampling the solution $\mathbf{x}$. Furthermore, the condition number has to grow linearly in n . These conditions make it challenging to trivially apply HHL as one of the linear algebraic subroutines found in many machine learning models such as Support Vector Machines (SVM) or Principal Component Analysis (PCA).

Chia et al. [30], demonstrated that under the assumption of having the same oracles assumed in HHL, such as the efficient loading of $\mathbf{b}$ and sampling of $\mathbf{x}$, a process called *dequantization* of HHL can be achieved. This process involves devising a classical algorithm that performs comparably. The quantum acceleration of HHL can be maintained in specific scenarios, for instance when $\mathbf{b}$ results from a quantum process and is hence already loaded, and when only specific properties of the output vector need estimation, such as its inner product with a reference state.

Topological data analysis

Lloyd, Garnerone, and Zanardi [31] introduced the LGZ algorithm for conducting topological data analysis using a quantum computer. This algorithm is designed to work with a simplicial complex, which is a generalized form of a graph encompassing both the pairwise relationships between vertices and any cliques present in the graph. In this context, the Betti number of degree k refers to the count of k -dimensional voids or holes within this complex. These Betti numbers hold significance as they serve as a topological fingerprint of the simplicial complex, allowing inference.

The core concept behind LGZ is centered around approximating the eigenvalues of the combinatorial Laplacian operator Δ_k . The proportion of samples with zero values is then interpreted as the Betti number for degree k , considering a specific form of normalization. This strategy leverages quantum computation to achieve exponential acceleration compared to classical counterparts. However, this quantum advantage is subject to stringent conditions. Firstly, the loading of the simplicial complex must be efficiently executed, even in the worst-case scenario taking up to n^k time for k -th Betti numbers. Secondly, the combinatorial Laplacian must have a sufficiently large $|\lambda_{\min}|$.

Non-linear differential equations

Liu et al. [32] have presented an algorithm that offers an exponential speedup over the best classical algorithm for simulating non-linear differential equations. However, this speedup is contingent upon the condition that the problem is dissipative, where the degree of dissipation is assessed using a measure akin to the Reynolds number. This approach is based on the Carleman linearization technique. Nonetheless, this approach encounters limitations and is not universally efficient in simulating all types of differential equations. For instance, it struggles with chaotic systems, as discussed in [33].

Despite these limitations, the algorithm remains applicable to significant instances within the realm of machine learning. An important example is the simulation of the (stochastic) gradient descent dynamics during the training of a neural network, itself a non-linear differential equation. In [34], the authors have proposed the use of this algorithm to facilitate the training of sparse neural networks. This specific type of differential equation exhibits dissipative behavior during its initial stages, corresponding to the phase of feature learning. This approach offers several advantages. Notably, the imposition of sparsity functions as a form of regularization, enhancing both the interpretability of the model and its overall performance.

2.3.4 Machine learning algorithms for NISQ devices

In addition to algorithms that are supposed to run on a fault-tolerant quantum computer, researchers have worked extensively to exploit the potential of the current generation of small-scale, noisy quantum hardware, denoted as NISQ [35]. Due to the hardware limitation of NISQ devices, we cannot implement the resource-demanding, deep quantum circuit required for the fault-tolerant algorithms. We instead rely on a parameterized set of shallow circuit, denoted as *parameterized quantum circuit*.

Parameterized quantum circuit

A parameterized quantum circuit (PQC) is a quantum circuit $U(\boldsymbol{\theta})$ that depends on a set of free parameters $\boldsymbol{\theta}$. Formally, it is a function

$$U : \boldsymbol{\theta} \mapsto U(\boldsymbol{\theta}) = \prod_{j=1}^{\ell} U_j(\boldsymbol{\theta}), \quad (2.57)$$

where each U_j can be either a rotational gate acting on one qubit

$$U_j(\boldsymbol{\theta}) = R_{\alpha}(f_j(\boldsymbol{\theta})) = \exp(-i f_j(\boldsymbol{\theta}) \sigma_{\alpha}), \quad (2.58)$$

or on two qubits

$$U_j(\boldsymbol{\theta}) = R_{\alpha,\beta}(f_j(\boldsymbol{\theta})) = \exp(-i f_j(\boldsymbol{\theta}) \sigma_{\alpha} \otimes \sigma_{\beta}), \quad (2.59)$$

or a constant gate, such as CNOT; and $\alpha, \beta \in \{0, x, y, z\}$. The gates U_j are supposed to be elementary gates on the target hardware platform considered, or at least easily implementable. The functions $f_j(\boldsymbol{\theta})$ determine the angle of rotation and can correspond to either a single element of the vector, e.g.

$$f_j(\boldsymbol{\theta}) = \theta_{i_j}, \quad (2.60)$$

or a non-linear function of one or more elements, e.g.

$$f_j(\boldsymbol{\theta}) = (\pi - \theta_{i_j})(\pi - \theta_{i_{j'}}). \quad (2.61)$$

Note that the input to the problem, x , can equivalently be specified either in the input model of a PQC $U(x)$ or as a computational basis state $|x\rangle$ at the beginning of the circuit

(with only a small overhead). Let $|\psi(\theta)\rangle = U(\theta) |0^n\rangle$. The state $|\psi(\theta)\rangle$ can be obtained starting from the state $|\theta\rangle |0^n\rangle$ by applying controlled- $U(2^{-k})$, where each application is controlled by a qubit in $|\theta\rangle$ from the most to the least significant.

Throughout the document, we consistently adopt the formalism of parameterized quantum circuits, even though we can alternatively refer to their CPTP form when needed. This flexibility becomes crucial when applying certain transformations, such as the feature map of the projected quantum kernel (cf. Section 3.3.1), which necessitates the use of the partial trace. Unlike a unitary transformation, the partial trace can be expressed as a CPTP map.

Expressivity of a parameterized quantum circuit

An important property of a PQC $U(\theta)$ is its expressivity, which can be interpreted in many ways according to the definitions used, but informally can be thought of as how large the set of unitaries $\{U(\theta) \mid \theta \in \Theta\}$ generated by randomly sampling θ . If the set is particularly large, close to the unitary group $\mathcal{U}(2^n)$ with n number of qubits, then the PQC is said to be as expressive.

To characterize this aspect, we need to employ tools from group theory. A *Haar measure* for the unitary group $\mathcal{U}(N)$ is its unique left- and right-invariant probability measure μ :

$$\int_{\mathcal{U}(n)} f(U) d\mu(U) = \int_{\mathcal{U}(N)} f(VU) d\mu(U) = \int_{\mathcal{U}(n)} f(UV) d\mu(U), \quad (2.62)$$

for all unitary $V \in \mathcal{U}(N)$ and function f . A *Haar random unitary* is represented as:

$$U \sim \mu, \quad (2.63)$$

capturing the notion of sampling 'uniformly' from the unitary group. The Haar measure is also used to define the mean value of a function over a unitary group:

$$\mathbb{E}_{U \in \mathcal{U}(n)} [f(U)] = \int_{U \in \mathcal{U}(n)} f(U) d\mu(U). \quad (2.64)$$

This is further used to define the *k-moment superoperator* $\mathcal{M}^{(k)} \mu$:

$$\mathcal{M}^{(k)} \mu(O) = \mathbb{E}[U^{\otimes k} O (U^\dagger)^{\otimes k}]. \quad (2.65)$$

Implementing a Haar random unitary is extraordinarily expensive and requires exponentially long circuits in N . However, we can generate a unitary that behaves similarly to a Haar random unitary up to the k -th statistical moment. A *unitary k -design* is a probability distribution ν over a set of unitaries $S \subseteq \mathcal{U}(N)$ such that:

$$\mathbb{E}_{V \sim \nu} [V^{\otimes k} O (V^\dagger)^{\otimes k}] = \mathbb{E}_{U \sim \mu} [U^{\otimes k} O (U^\dagger)^{\otimes k}] \quad (2.66)$$

for all linear operators $O \in L(\mathbb{C}^N)$. Furthermore, an ϵ -approximate unitary k -design is a probability distribution ν such that:

$$\|\mathcal{M}_\mu^{(k)} - \mathcal{M}_\nu^{(k)}\|_\diamond \leq \epsilon. \quad (2.67)$$

Finally, the expressivity of a PQC is defined as the norm of the superoperator $\mathcal{A}$,

$$\mathcal{A}(U(\theta)) = \int_\mu (V |0\rangle\langle 0| V^\dagger)^{\otimes 2} d\mu(V) - \int_{\theta \sim \Theta} (U(\theta) |0\rangle\langle 0| U(\theta))^{\otimes 2} d\theta, \quad (2.68)$$

quantifying how much the distribution of states generated by the PQC $U(\theta)$ diverges from the distribution of states generated by a unitary 2-design. If $\|\mathcal{A}\|_\diamond$ is close to zero, then the PQC is considered expressive. Note that this definition considers both the reached states and their distribution. If a state is sampled by many $\theta \sim \Theta$, this aspect will be considered to differ from Haar random behavior, where each state is uniformly sampled.

Variational algorithms

Variational quantum algorithms [36] are optimization techniques that search for the set of parameters that minimizes a cost function C ,

$$\theta^* = \arg \min_{\theta} C(\theta), \quad (2.69)$$

where the cost is a function of a set of input $\{\rho_i\}_{i=1}^j$, a set of observables $\{O_i\}_{i=1}^k$, and a PQC $U(\theta)$,

$$C(\theta) = g(\rho_1, \dots, \rho_j; O_1, \dots, O_k; U(\theta)). \quad (2.70)$$

A sub-optimal solution to Equation 2.69 can be obtained using stochastic gradient descent. While the gradient of C can be calculated using the finite difference method,

the computation can be greatly affected by noise. Instead, the parameter-shift rule [37] is preferred, as it utilizes a much larger step size ($\pi/2$ instead of $\epsilon \ll 1$), making it less susceptible to the intrinsic noise of NISQ devices.

An important issue that arises when dealing with variational algorithms is the *barren plateau*. This situation occurs when calculating the gradient of the cost function $\nabla_{\theta}C$ has values close to its average value; basically,

$$\text{Var}[\nabla_{\theta}C] \propto \exp(2^{-n}), \quad (2.71)$$

with n being the number of qubits. Practically, the barren plateau occurs when the cost function flattens exponentially fast in n , making optimization challenging for both gradient descent-based and gradient-free approaches [38]. The root causes of this phenomenon are still the subject of investigation, although studies have shown that a highly expressive unitary [39], as well as noise [40] and global observables ([38]), are correlated with the appearance of the phenomenon.

Proposition 1 ([41]). *If the probability distribution ν over the unitaries generated by a n -qubit PQC, $\{U(\theta)\}_{\theta \in \Theta}$, is a unitary 2-design, then*

$$\begin{aligned} \mathbb{E}_{U \sim \nu} [C(\theta)] &= 0, \\ \text{Var}_{U \sim \nu} [C(\theta)] &\in O(2^{-n}). \end{aligned}$$

Quantum kernel

A *quantum feature map* is a mapping from classical data, $\mathbf{x} \in \mathcal{X}$ (usually $\mathcal{X} \equiv \mathbb{R}^d$), to a vector lying in the Hilbert space of an n -qubit quantum system, $\rho(\mathbf{x}) \in (\mathbb{C}^2)^{\otimes n}$. Consequently, a *quantum kernel* [21, 42] is a kernel function κ whose underlying feature map is effectively a quantum feature map, and whose encoded vectors belong to the Hilbert space of the quantum system. Formally, it is represented as a mapping in the form:

$$\kappa(\mathbf{x}, \mathbf{x}') = \text{Tr}[\rho_{\mathbf{x}} \rho_{\mathbf{x}'}]. \quad (2.72)$$

Equation 2.72 can be implemented using the *overlap test* (cf. Figure 3.2), whose underlying quantum circuit performs the transformation:

$$\kappa(\mathbf{x}, \mathbf{x}') = \text{Tr}[(U^{\dagger}(\mathbf{x}')U(\mathbf{x})\rho_0U^{\dagger}(\mathbf{x})U(\mathbf{x}'))|0\rangle\langle 0|]. \quad (2.73)$$

An alternative to the proposed approach is the ‘swap test’, whose quantum circuit is shown in Figure 3.3. In this case, the inner product is proportional to the probability of reading outcome 0 when measuring in the computational basis over the topmost qubit only,

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{x}') &= 2 \operatorname{Tr}[V\rho_0 V^\dagger O] - 1 \\ V &= (H \otimes \mathbb{I}_n \otimes \mathbb{I}_n)(c\text{SWAP})(H \otimes U(\mathbf{x}) \otimes U(\mathbf{x}')) \\ O &= |0\rangle\langle 0| \otimes \mathbb{I}_n \otimes \mathbb{I}_n\end{aligned}\tag{2.74}$$

Additionally, note that the swap test allows us immediately to implement partial measurement by applying CSWAP uniquely to the subset of qubits of interest.

Variational quantum classifier

A *variational quantum classifier* (VQC), sometimes referred to as a *quantum neural network* (QNN), is a machine learning model defined via a quantum circuit:

$$h(\mathbf{x}; \boldsymbol{\theta}) = \operatorname{Tr}[U(\boldsymbol{\theta})V(\mathbf{x})\rho_0 V^\dagger(\mathbf{x})U^\dagger(\boldsymbol{\theta})O].\tag{2.75}$$

Here, ρ_0 is the initial state of the system; $V(\mathbf{x})$ is a feature encoding unitary. As it effectively maps a set of data $\mathbf{x}$ to some Hilbert space, in this case, the Hilbert space of the quantum system, it is definitively a feature map; $U(\boldsymbol{\theta})$ is a variational form or ansatz and depends on the set of trainable parameters $\boldsymbol{\theta}$; finally, O is a Hermitian observable. The training of this model is indeed an instance of variational algorithms, with the cost function defined as:

$$C(\boldsymbol{\theta}) = \sum_{j=1}^n \ell(h(\mathbf{x}^{(j)}; \boldsymbol{\theta}), y^{(j)}).\tag{2.76}$$

Here, ℓ is a binary loss function, e.g., the mean square error (cf. Equation 2.2).

2.3.5 Quantum advantage in supervised learning

The boundaries between tasks that are feasible and convenient to solve using classical machine learning and those that might benefit from quantum approaches are still blurry. However, some limited evidence has been proposed in the literature.

A possible approach to this issue is offered by computational complexity. Classical machine learning can effectively be modeled as the class BPP/samp of randomized

computation with the ability to sample data (assuming the data itself cannot be computed in BPP). Note that BPP/samp is contained in BPP/poly and thus is modeled as an advice class; for a more detailed treatment of this aspect, we refer the reader to Marshall, Gyurik, and Dunjko [43]. Huang et al. [44] has formalized such a class and proven that, under reasonable complexity-theoretical assumptions, you can find 1. tasks that require quantum computation whenever we have no data, but in the presence of data, they can be efficiently solved ($BQP \cap (BPP/samp \setminus BPP)$), and 2. tasks that truly require quantum computation even in the presence of data.

Here, we highlight a few of the most relevant works in the literature. Liu, Arunachalam, and Temme [45] has developed an approach based on the conjectured hardness of the discrete logarithm and used such a problem to define a learning task that can be solved efficiently only on a quantum computer: the task is obtained from a linearly separable dataset, and then the data is obfuscated using exponentiation. Jäger and Krems [46] has extended the approach to BQP-complete tasks, which are beyond the discrete logarithm in terms of hardness. Physically motivated applications have been shown in Gyurik and Dunjko [47] and Molteni, Gyurik, and Dunjko [48]; the advantage is stated in terms of the probably approximately correct (PAC) learning model. A related but not physically motivated work has been proposed by Yamasaki, Isogai, and Murao [49].

These works suggest that a quantum advantage in learning tasks can be obtained whenever the task itself can be reframed in terms of classically hard, quantumly easy tasks; most of them being physically motivated, e.g., learning an observable, learning properties of a time-evolved quantum state, or, in some limited scenarios, of ground states [48].

Chapter 3

Automatic and effective discovery of quantum kernels

This chapter delves into the promises and challenges of using the class of quantum kernels designed for NISQ devices to tackle machine learning tasks. Section 3.1 introduces kernel compatibility, the aspect that shows whenever a specific kernel function can indeed capture the relations within the data at hand, and shows how to measure it. Section 3.2 discusses common issues in using quantum kernels, for example, the ones arising when using both overly expressive and overly simple quantum circuits as feature encoding unitaries. Section 3.3 introduces techniques to control the expressivity of a quantum circuit. In Section 3.4, we propose an algorithm for the automatic discovery of a quantum kernel that best fits the task at hand. In Section 3.5, we test our approach to a well-known problem in high-energy physics, the anomaly detection of beyond Standard Model events. This chapter is based on [6, 7].

3.1 The Kernel Compatibility

Although diverse kernels have been proposed, one cannot expect to find a kernel function that successfully fits all tasks (no free lunch theorem). Different tasks have different relationships between data and output, requiring different kernels to work. A possible measure for the 'fitness' of a kernel for a specific task is the *kernel compatibility*. In the literature, different metrics have been proposed for assessing this aspect, which this section overviews.

3.1.1 Kernel-target alignment

A fundamental measure of compatibility is the Kernel-Target Alignment (KTA), introduced by Cristianini et al. [50]. The KTA quantifies the agreement between a kernel and a particular learning task: given a set of labeled data $(\mathbf{x}^{(j)}, y^{(j)})_{j=1}^n$ with $y^{(j)} = \hat{f}(\mathbf{x}^{(j)})$, the KTA measure states that the kernel machine $\hat{h}(\cdot) = \boldsymbol{\alpha}^\top \mathbf{k}(\cdot)$ learns the target function $\hat{f}(\cdot)$ well if, for $1 \leq i, j \leq n$, we have

$$\hat{h}(\mathbf{x}^{(i)})\hat{h}(\mathbf{x}^{(j)}) \approx y^{(i)}y^{(j)}. \quad (3.1)$$

In the formula, this is quantified as the (normalized) inner product between the kernel Gram matrix K and the label matrix $Y = \mathbf{y}\mathbf{y}^\top$,

$$\text{KTA}(K, Y) = \frac{\langle K, Y \rangle_F}{\sqrt{\langle K, K \rangle_F \langle Y, Y \rangle_F}}, \quad (3.2)$$

Here, the inner product $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product. The KTA can be equivalently interpreted as the cosine of the angle between the (vectorized) K and Y , implying the KTA image being $[-1, 1]$. For values close to 1, we say the two models have high kernel compatibility.

The work of Cortes, Mohri, and Rostamizadeh [51] has proposed an improvement over this metric. Noting that, in certain scenarios, the KTA does not correlate well with the actual generalization error of the model, the authors have proposed to subtract the mean value of each matrix, showing that the results more realistically capture the performance of the model. Their metric, the Centered Kernel Alignment (CKA), is defined as

$$\text{CKA}(K, Y) = \text{KTA}(HKKH, HYYH), \quad (3.3)$$

where H is the centering matrix, $H = \mathbb{I} - (1/n)\mathbb{1} \cdot \mathbb{1}^\top$.

3.1.2 Task-model alignment

Much more recently, the work of Canatar, Bordelon, and Pehlevan [52] has exploited tools from statistical physics, the replica method, to recover important insight into the performance of a kernel machine.

The basic idea behind this approach starts with Mercer's theorem, which studies the

eigendecomposition of the kernel integral operator

$$\int \kappa(\mathbf{x}, \mathbf{x}') \phi_j(\mathbf{x}') d\mu(\mathbf{x}') = \lambda_j \phi_j(\mathbf{x}), \quad (3.4)$$

resulting in an infinite sequence of real, positive eigenvalues $\{\lambda_j\}_{j=0}^{\infty}$ and eigenfunctions $\{\phi_j\}_{j=0}^{\infty}$. Suppose the eigenvalues are sorted in non-descending order. As the set of eigenfunctions is an orthonormal basis for the space of square-integrable functions on $\mathcal{X}$, we can use it to express the target function $\hat{f}$ and the kernel machine $\hat{h}$,

$$\hat{f}(\mathbf{x}) = \sum_{j=0}^{\infty} v_j \lambda_j \phi_j(\mathbf{x}), \quad (3.5)$$

$$\hat{h}(\mathbf{x}) = \sum_{j=0}^{\infty} w_j \lambda_j \phi_j(\mathbf{x}). \quad (3.6)$$

We have avoided explicitly noting the dependency on the dataset and number of samples for clarity of notation. In these equations, we have assumed $\lambda_j > 0$ for all j ¹.

We decompose the generalization error into the direct sum of errors for each individual mode:

$$E = \int (\hat{f}(\mathbf{x}) - \hat{h}(\mathbf{x}))^2 d\mu(\mathbf{x}) = \sum_{j=0}^{\infty} \lambda_j (v_j - w_j)^2. \quad (3.7)$$

This decomposition reveals that the components with larger λ_j are indeed more important, as they contribute more to the generalization error. Experiments in [52] have also shown that these components can be learned with fewer samples compared to the components with smaller eigenvalues.

To capture the idea of spectral bias, i.e., spectral compatibility between the target function and the model, the *task-model alignment* has been introduced. It corresponds to how much of the total power is concentrated in the first k components:

$$C(k) = \frac{\sum_{j=0}^k \lambda_j v_j^2}{\sum_{j=0}^{\infty} \lambda_j v_j^2}. \quad (3.8)$$

The larger $C(k)$, the more power is concentrated in the k easier-to-learn components, making the training more efficient.

¹If $\hat{f}$ has a non-negligible contribution from the eigenfunction ϕ_j , such a contribution will not be learnt at all with the kernel machine (as $\lambda_j \rightarrow 0$ we have $v_j \rightarrow \infty$). It is possible to take into account these contributions, named the out-of-RKHS components.

3.2 Challenges in the design of a quantum kernel

Kernel machines with quantum kernels give rise to a class of models with high complexity, implying that the corresponding $\hat{h}$ will be complicated functions. While these functions may interpolate the data, they may struggle to generalize. In this section, we discuss these issues.

3.2.1 Expressible quantum kernel cannot learn nearly all functions

Kübler, Buchholz, and Schölkopf [53] decay of the eigenvalue distribution (linked to the task-model alignment)

Proposition 2 (Theorem 3 in [53]). *Let (X, μ) be a measure space with $\mu(X) = 1$. Let κ be a normalized kernel, i.e. $\kappa(x, x) = 1$ for all $x \in X$, and $\lambda_{\max}$ be the largest eigenvalue of its integral operator. Let $\hat{f}$ be a square-integrable function. Let $\hat{h}$ be a kernel Ridge regression model with an arbitrary regularization constant, trained on the dataset $\{(x^{(j)}, y^{(j)})\}_{j=1}^n$ with $x^{(j)}$ i.i.d. sampled from μ and $y^{(j)} = \hat{f}(x^{(j)})$. Then, for any $\epsilon > 0$, with probability at least $1 - \epsilon - \lambda_{\max} n^4$, we have*

$$\|\hat{f} - \hat{h}\|_2 \geq \left(1 - \frac{2\lambda_{\max} n^2}{\epsilon}\right) \|\hat{f}\|_2. \quad (3.9)$$

In practice, the theorem states that a kernel machine with a small $\lambda_{\max}$ will have a large error for nearly all functions. In contrast, if $\lambda_{\max}$ is large, the bound is vacuous, and nothing can be said about the error.

This bound is especially important when considering quantum kernels. Given the mean density matrix as

$$\rho_\mu = \int \rho(x) \mu(dx), \quad (3.10)$$

we can prove that $\lambda_{\max}^2 \leq \text{Tr}[\rho_\mu^2]$ (cf. [53, Appendix C]). This implies that the purity of the mean density matrix provides insight into the magnitude of the largest eigenvalue. When the purity is large, it means that any x is mapped to almost the same density matrix; in such a case, the bound in Equation 3.9 becomes vacuous. Conversely, a small purity indicates an expressive embedding that maps x to varying density matrices, leading to the bound holding and resulting in a large error.

Proposition 3 (Theorem 1 in [53]). *Let $\delta < 1$ be an upper bound on the purity of the mean embedding ρ_μ . Let the training sample size n grow polynomially in the number of*

qubits d . Then, there exists a d_0 such that for all $d \geq d_0(\delta)$, the kernel Ridge regressor $\hat{h}$ using the d -qubit quantum kernel κ (Equation xx) cannot learn any square-integrable function $\hat{f}$, meaning for any $\varepsilon > 0$ and probability at least $1 - \varepsilon$,

$$\|\hat{f} - \hat{h}\|_2 \geq (1 - \varepsilon) \|\hat{f}\|_2. \quad (3.11)$$

This strongly negative result highlights that an expressive kernel (indicated by a purity of the mean embedding $\delta < 1$) will not scale with respect to the number of qubits.

3.2.2 Expressible quantum kernel cannot be calculated precisely

Another challenge may arise in the context of quantum kernels. An expressive quantum kernel tends to distribute the encodings $\rho(x)$ across the exponentially large Hilbert space. Consequently, as the number of qubits grows, the inner product $\text{Tr}[\rho(x)\rho(x')]$ rapidly approaches zero exponentially, or more generally, becomes exponentially close to its mean value μ .

The analysis conducted in Thanasilp et al. [54] has demonstrated that, when this phenomenon of *exponential concentration* occurs, it becomes practically impossible to distinguish, for $x \neq x'$, whether $k(x, x')$ is equal to μ or not. This is because estimating such a value on a quantum computer would require an exponential number of samples or shots.

Formally, a quantity $X(\alpha)$, estimated as an observable on a quantum computer operating with n qubits, is considered exponentially concentrated toward its mean value $\mathbb{E}[X(\alpha)]$ if

$$\text{Var}[X(\alpha)] \in O(2^{-n}). \quad (3.12)$$

We can statistically estimate this quantity using a polynomial number of samples. With $N = \text{poly}(n)$ shots, the sampling error is $\varepsilon \in O(1/\sqrt{N})$, while the resolution needed to distinguish the kernel values is much smaller, $r \in O(1/2^n)$. Since $r \ll \varepsilon$, the statistical noise will ultimately corrupt the estimation of the quantity irreparably. Immediately, we observe the barren plateau phenomenon in variational algorithms if we set $\alpha = \theta$ and $X(\alpha) = C(\theta)$. In the case of quantum kernels, we must set $\alpha = (x, x')$ and $X = \kappa$.

This approach enables the reiteration of the results obtained for the barren plateau in variational algorithms within the realm of quantum kernels. Just as the barren plateau emerges for expressible unitaries, highly entangled systems, global measurement use, and in the presence of noise, similar challenges arise when dealing with quantum kernels.

We refer to an important result concerning expressivity.

Proposition 4 (Theorem 1 in [54]). *Let k be a quantum kernel as in Equation xx, and $\mathcal{A} = \mathcal{A}(U(\mathbf{x}), \rho_0)$ be a measure for its expressivity as in Equation xx ($\mathcal{A} \ll 1$ for expressible unitaries). Let $\mathbf{x}, \mathbf{x}'$ be i.i.d. sampled from $\mu(\mathcal{X})$. Then,*

$$Pr\left[\left|\kappa(\mathbf{x}, \mathbf{x}') - \mathbb{E}[\kappa(\mathbf{x}, \mathbf{x}')] \right| \geq \delta\right] \leq \frac{\left(\sqrt{\frac{1}{2^{n-1}(2^n+1)}} + \mathcal{A}\right)^2}{\delta^2}. \quad (3.13)$$

3.2.3 Simple quantum kernels can be efficiently simulated

Although having an expressible quantum kernel is a primary source of issue, another important problem is given by the fact that too simple unitaries are efficiently simulable by a classical computer, and thus render the use of a quantum computer pointless. The reason resides in the works of Somma et al. [55] and Larocca et al. [56] regarding the dynamical Lie algebra of a parameterized quantum circuit.

A Lie algebra, denoted as $\mathfrak{g}$, is a vector space equipped with a binary operation called the commutator, denoted as $[\cdot, \cdot]$. This operation satisfies the skew-symmetry condition $[x, y] = -[y, x]$ and the Jacobi identity, $[x, [y, z]] + [y, [z, x]] + [z, [x, y]] = 0$. The Lie group $e^{\mathfrak{g}}$, which is both a group and a differentiable manifold, is obtained by means of an exponential map $\exp : \mathfrak{g} \rightarrow e^{\mathfrak{g}}$, and the elements of this group are arbitrary products over the exponentiated elements of $\mathfrak{g}$.

In the context of quantum mechanics and quantum optimal control, we deal with the Lie algebra $\mathfrak{u}(n)$ of skew-Hermitian complex $n \times n$ matrices, where the group product is matrix multiplication and the commutator operation is defined as $[A, B] = AB - BA$. The corresponding Lie group is the unitary group $U(n)$, and the exponential map corresponds to the matrix exponential. Now, consider a Lie algebra spanned by a set of parameterized Hermitian operators $H_j(\theta) \in \mathbb{C}^{n \times n}$,

$$\mathfrak{g} = \text{span}_{\theta \in \Theta} \{-iH_j(\theta)\}. \quad (3.14)$$

Then, $\mathfrak{g}$ is called the *dynamical Lie algebra* (DLA, [57]). Furthermore, if the set $G = e^{\mathfrak{g}}$ is equal to the whole unitary group $U(n)$, the system is said to be *controllable*.

The work by Larocca et al. [56] has demonstrated that an expressible PQC $U(\theta)$, where expressivity refers to $U(\theta)$ being a unitary 2-design and thus a source of barren plateau, is controllable, and vice versa. This implies that a DLA with dimension $O(\exp n)$

will result in a barren plateau (in variational algorithms) or exponential concentration of values (in quantum kernels); conversely, a polynomially sized DLA will not be too expressible, thus avoiding this problem, at least concerning the aspect of expressivity (though issues may still arise due to factors such as noise).

However, previous works by Somma et al. [55] have shown that a system with a polynomially sized (in n) DLA is efficiently classically simulable. This is possible by creating a Lie-algebraic model of computation with polynomially sized representations of states and operators. This aspect significantly impacts our prospects for achieving a quantum advantage because exponentially sized DLA leads to useless models, and polynomially sized DLA are classically simulable. However, a window for an advantage remains if we consider that we can indeed have a very large DLA while controlling the expressivity by restricting the set of parameters Θ in Equation 3.14 to a fairly small range. This intuition gives rise to the bandwidth approach for expressivity control (cf. Section 3.3.2).

3.2.4 Choosing the quantum kernel requires insight on the task

A last important aspect to consider is that even if we achieve a statistically favorable and hard-to-simulate quantum kernel, we have no guarantee that such a similarity measure will give us positive results. In contrast, many works that have applied quantum kernels to real-world machine learning scenarios have shown degraded performances compared to their classical counterpart [23].

It has been proposed that quantum kernels might benefit learning physical processes, which occur in many fields such as chemistry, material science, pharmacology, and high-energy physics [58]. However, the quantum kernel should be designed to capture the relationship between the input data and the output of such a process, and this requires at least some knowledge of the nature of the task itself. An example of how to accommodate these requirements is shown in Section 3.3.3.

3.3 How to control the expressivity of a quantum kernel

We have seen that expressivity causes several problems in using quantum kernels. Fortunately, some techniques mitigate these problems, recovering a functional kernel.

3.3.1 Projected quantum kernels

The projected quantum kernel was introduced by Huang et al. [44]. It involves leveraging a unitary operation on a large Hilbert space, making the transformation hard to simulate classically. The encoded vector is then projected into a smaller Hilbert space by tracing out most of the qubits. The function is defined as follows:

$$\kappa(\mathbf{x}, \mathbf{x}') = \text{Tr}[\tilde{\rho}_{\mathbf{x}} \tilde{\rho}_{\mathbf{x}'}], \quad (3.15)$$

where

$$\tilde{\rho}_{\mathbf{x}} = \text{Tr}_Q[U^\dagger(\mathbf{x})\rho_0 U(\mathbf{x})], \quad (3.16)$$

represents the reduced density matrix obtained by tracing out a subset $Q \subset \{1, \dots, n\}$ of the n qubits in the system. The Hilbert space of the reduced density matrix needs to be relatively small, with a constant number of qubits, for the approach to scale as desired.

Huang et al. [44] has applied this technique to learn a quantum process and introduced a test for quantifying potential quantum advantages when using a quantum kernel. The quantum process is defined by the function

$$\hat{f}(\mathbf{x}) = \text{Tr}[U^\dagger(\mathbf{x})\rho_0 U(\mathbf{x})O], \quad (3.17)$$

where $U(\mathbf{x})$ is a parameterized quantum circuit, ρ_0 is the initial state, and O is an observable. The function $\hat{f}$ is accessible only through the (noiseless) dataset $\{(\mathbf{x}^{(j)}, y^{(j)})\}_{j=1}^n$, where each label satisfies $y^{(j)} = \hat{f}(\mathbf{x}^{(j)})$. Let K be the kernel Gram matrix of κ , and we normalize it such that $\text{Tr}[K] = n$ or $\kappa(\mathbf{x}, \mathbf{x}') = 1$; while $\hat{h}$ is the kernel regression model using κ .

The *geometric test* to verify a potential quantum advantage relies on two quantities: the S-value $s(K)$, a measure of the model's complexity, and the geometric difference $g(K_1, K_2)$, a measure of how much two kernel functions diverge. The S-value is defined as

$$s(K) = \sum_{i,j=1}^n (K_{i,j}^{-1}) \hat{f}(\mathbf{x}^{(i)}) \hat{f}(\mathbf{x}^{(j)}). \quad (3.18)$$

This value quantifies how well the kernel function captures the behavior of the quantum process. The kernel can successfully capture the relationship within the data if

$$\kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \approx f(\mathbf{x}^{(i)}) f(\mathbf{x}^{(j)}). \quad (3.19)$$

It is important to note that $s(K) = \|\hat{h}\|$, making it a measure of the model's complexity. Higher values of $s(K)$ suggest that the kernel machine becomes a more complex function, which can lead to overfitting and poor generalization performance. The geometric difference of two normalized kernel Gram matrices is defined by

$$g(K_1, K_2) = \sqrt{\|\sqrt{K_2}K_1^{-1}\sqrt{K_2}\|_\infty}, \quad (3.20)$$

where $\|\cdot\|_\infty$ is the spectral norm, i.e., the largest singular value.

The geometric test operates as follows. Let K_C and K_Q be the kernel Gram matrix of a classical kernel and a quantum kernel, respectively. Then,

- When $g(K_C, K_Q) \ll \sqrt{m}$, a classical kernel exhibits behavior similar to the quantum kernel, rendering the use of the quantum computer not necessary.
- When $g(K_C, K_Q) \approx \sqrt{m}$, the quantum kernel significantly deviates from all tested classical kernels. The outcome depends on the complexity of classical kernel machines:
 - If the complexity of any classical kernel machine is low ($s(K_C) \ll m$), classical kernels perform well, and the quantum kernel's divergence from classical K_C , doesn't yield superior performance.
 - When the complexity of all classical kernel machines is high ($s(K_C) \approx m$), classical models struggle to learn the function $\hat{f}$. In this scenario:
 - * If the quantum model's complexity is low ($s(K_Q) \ll m$), the quantum kernel successfully solves the task while the classical models do not.
 - * If the quantum model's complexity is high ($s(K_Q) \approx m$), even the quantum model struggles to solve the problem.

3.3.2 Bandwidth

Another technique, either used in conjunction with or as an alternative to projection, can be employed to control expressivity.

This technique is rooted on the observation that any unitary $U(\theta)$, no matter how expressive it is, can be limited to an arbitrarily small portion of the Hilbert space. This can be obtained relying to the interpretation of the unitary gate as a rotation of the quantum state, and limiting this rotation to a certain portion of the possible angles.

Operatively, we require to multiply θ by a small positive parameter β called bandwidth that can be either constant or dependent by n number of qubits. In this way, we can still employ 'hard to simulate' unitaries without dealing with how expressible it is, as effectively the quantum states will be spread across a limited section of the Hilbert space.

Canatar et al. [59] demonstrated that the use of a bandwidth can render the bound in Equation 3.11 vacuous. It is important to notice that employing a bandwidth does not necessarily result in a useful quantum kernel, as the bandwidth has no impact on kernel compatibility.

3.3.3 Symmetries in the data

Whenever our tasks present certain kind of structures, we can design simpler models that takes advantage of the corresponding inductive biases. That is the case for datasets having some symmetries [60, 61]. An example of these is the *label symmetry*. The effect of having a model that exploit the symmetries within the dataset result in a simpler model with reduced expressivity. Although symmetries within the data is not relevant for our modeling, for the sake of completeness we show an example based on the concept of *equivariance*.

Formally, let $h : X \rightarrow \mathcal{Y}$ be a mapping, S be a symmetric group, and $R : S \rightarrow \text{Aut}(X)$ be a representation of such a group, i.e. $R(g_1 \circ g_2) = R(g_1) \circ R(g_2)$. Then, h has a *label symmetry* if and only if it is invariant under S ,

$$h(R(g)(x)) = h(x) \quad \forall g \in S. \quad (3.21)$$

We often want to design a QNN $h_\theta(x) = \text{Tr}[U_\theta \Psi(x) U_\theta^\dagger O]$ invariant under S , just as shown above. In this case, $\Psi : X \rightarrow \mathcal{H}$, $\Psi(x) = \rho_x$ is the feature map, U_θ is the ansatz and O the observable.

Then, Ψ is equivariant under $g \in S$ if exists a unitary $U_R(g)$ such that $\Psi(R(g)(x)) = U_R(g) \Psi(x) U_R^\dagger(g)$. Then, gate and observable are equivariant whenever they commute with $U_R(g)$ for all $g \in S$. Combining an equivariant feature map, an ansatz made of equivariant gates and an equivariant observable then we can obtain a QNN that is invariant with respect to S :

$$h(R(g)(x); \theta) = \text{Tr}[U(\theta) \Psi(R(g)(x)) U^\dagger(\theta) O] \quad (3.22)$$

$$= \text{Tr}[U(\theta) U_R(g) \Psi(x) U_R^\dagger(g) U^\dagger(\theta) O] \quad (3.23)$$

$$= \text{Tr}[U_R(g)U_R^\dagger(g)U(\boldsymbol{\theta})\Psi(\boldsymbol{x})U^\dagger(\boldsymbol{\theta})O] \quad (3.24)$$

$$= \text{Tr}[U(\boldsymbol{\theta})\Psi(\boldsymbol{x})U^\dagger(\boldsymbol{\theta})O] = h(\boldsymbol{x}; \boldsymbol{\theta}). \quad (3.25)$$

3.4 Automatic procedure to build a quantum kernel

Extensive research has been devoted to developing automatic procedures to tailor a quantum kernel for each specific task. The procedure mainly involves changing the parameterized quantum circuit characterizing the feature map via an iterative, heuristic process. Many approaches are possible, with the two major families are based on the *variational* and *combinatorial* methods.

The variational method consists of having a parameterized quantum circuit depending on a set of parameters that can be partitioned into parameters used to encode the input and freely trainable parameters. In this sense, the quantum circuit is identical to those of quantum neural networks. We train the free parameters with stochastic gradient descent-based algorithms over a certain loss function. If we were training a QNN, the output of the quantum circuit would be a label y , and we would be training a model; in the case of the quantum kernel, the output is the kernel itself $\kappa(\boldsymbol{x}, \boldsymbol{x}')$, and the process is not training itself but hyperparameter optimization. This technique has been successfully applied to two tasks. Lloyd et al. [62] pioneered the approach and proposed kernel target alignment as the loss function to minimize. Glick et al. [63] applied the approach to data $\boldsymbol{x}$ possessing a group structure.

In contrast, the combinatorial method does not rely on free parameters to be trained, but instead sets up a combinatorial optimization problem in which the variables are the single gates to be chosen and placed on the quantum circuit. This kind of optimization is discrete in nature, given that we have a discrete set of gates to be applied. Some attempts to use this method have been seen in quantum chemistry-related tasks. Firstly, Ostaszewski et al. [64] pioneered its use and solved the problem via reinforcement learning, successfully applying it to find the ground state of the lithium hydride molecule. Secondly, Meng et al. [65] followed a similar approach but focused on using a Monte Carlo Tree Search method for optimizing the quantum circuit that, in contrast to reinforcement learning approaches, does not need to undergo a training phase. Again, Huang et al. [66] tested evolutionary algorithms. Finally, Du et al. [67] combined both variational and combinatorial approaches in a single optimization process.

In regard to quantum kernels, the topic has not been as widely explored as VQE

and QNNs. A first attempt in this direction is the work by Altares-López, Ribeiro, and García-Ripoll [68]: a finite discrete set of gates comprising CNOT , R_x , R_y , and R_z is arranged on the quantum circuit according to a cost function that simultaneously maximizes accuracy while minimizing the number of two-qubit gates. A limitation of this approach consists of the optimization process finding efficiently classically simulable circuits, due to the explicit penalization of two-qubit gates and the implicit penalization from unfavorable statistical properties of expressible circuits. A similar approach has been developed by Chen and Chern [69].

In this section, we provide our algorithm for the automated building of a quantum kernel. Initially, we represent a quantum kernel with n qubits and m gates as a discrete entity denoted by $\mathbf{k}$,

$$\mathbf{k} \in \mathcal{QK}_{n,m} \quad (3.26)$$

This quantum kernel belongs to the set $\mathcal{QK}_{n,m}$, representing the space of quantum kernels with n qubits and m gates. The element of this set can be represented by matrices (or vectors) of integers, represented by $\mathbf{k} \in \mathbb{N}^{6m+n}$, where $6m+n$ indicates the number of characteristics needed to characterize the kernel. Furthermore, we present a collection of criteria represented by $C = \{C_1, C_2, \dots\}$, where each $C_i : \mathcal{QK}_{n,m} \rightarrow \mathbb{R}_{\geq 0}$. Each of these criteria measures different characteristics of a quantum kernel, including its ability to represent information, its efficiency in being simulated classically, and its compatibility with the specific task being performed. Each of these criteria aligns with distinct theoretical aspects that have been established in the literature. The criteria will be utilized to establish a cost function based on a (potentially non-linear) mapping of these variables,

$$C(\mathbf{x}) = f(C_1(\mathbf{x}), \dots, C_m(\mathbf{x})). \quad (3.27)$$

Furthermore, we need to choose an initial candidate solution. Prior knowledge of the task can help us make an initial guess, where we hypothesize that a candidate solution may be close to a satisfactory one. Alternatively, it can also function as the identity circuit. Ultimately, we utilize a heuristic optimization approach to resolve the task:

$$\mathbf{k}^* = \arg \min_{\mathbf{k} \in \mathcal{QK}_{n,m}} C(\mathbf{k}). \quad (3.28)$$

The nonconvexity of the cost function necessitates the employment of a heuristic technique, which, like other heuristics, does not provide any assurance of finding an optimal solution. The scheme we have built, along with the accompanying software,

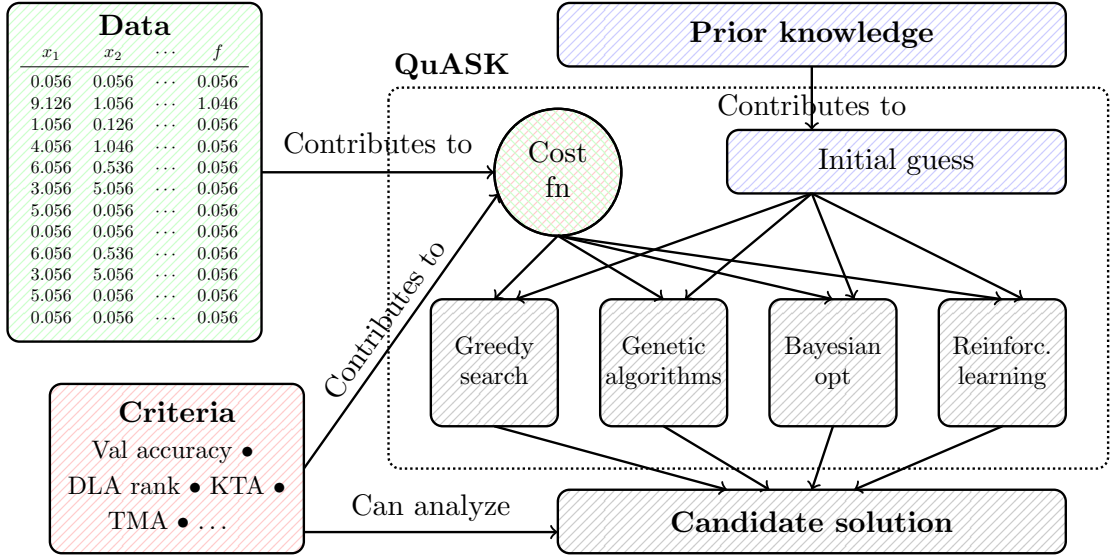


Figure 3.1: Overview of the automatic process for constructing a quantum kernel. The process starts with an initial guess, which can be an empty quantum circuit (identity unitary) or an informed choice based on prior knowledge. Next, a cost function is defined based on the training data and established criteria from the literature that provide insights into the statistical properties of the kernel. This cost function becomes the objective of an iterative optimization process; the specific optimization technique is selected by the researcher from the available options. The same criteria used in the cost function can also be employed to evaluate and debug the candidate solution at the end of the process.

enables the use of various heuristics. This is illustrated in Figure 3.1.

3.4.1 Modeling quantum kernels

The method we present extends the approach demonstrated in [64]. Let d be the dimensionality of the feature vector, $\mathbf{x} \in \mathbb{R}^d$, n the number of qubits in the quantum circuit, and m the number of gates. The parameterized quantum circuit is defined as a matrix of size $6 \times m$; each column corresponds to a gate and is characterized by the generator, the qubit on which it is applied, the classical feature parameterizing the operation, and the bandwidth. Formally, each operation corresponds to:

$$G = \exp\left(-i \frac{\beta_j \mathbf{x}_k}{2} \sigma_\alpha^{(p)} \sigma_\beta^{(q)}\right) = R_{\alpha,\beta}^{(p,q)}(\beta_j \mathbf{x}_k). \quad (3.29)$$

More specifically, $\mathbf{x}_k$ denotes the k -th characteristic of the data point $\mathbf{x}$, which belongs to the vector space $\mathbb{R}^d$. σ_α and σ_β are the Pauli generators of the quantum



Figure 3.2: The ‘overlap test’ quantum circuit is employed to estimate the inner product of two vectors, $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, which are encoded using the unitary transformation U . The likelihood of observing the string 0^n as the result of the quantum circuit with unlimited accuracy (infinite shots) is evaluated using the formula in Equation 2.73.

gate, specifically σ_α and σ_β are elements of the set $\{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$. When either σ_α or σ_β equals σ_0 , the gate functions effectively on a single qubit. The parameters (p, q) are the pair of qubits over which the quantum gates are applied; they must be distinct. Furthermore, the value of β_j lies between 0 and 1, and it serves as an optional scaling factor known as the “bandwidth”. The potential bandwidth values are part of a discrete set $\{\beta_j\}_{j=0}^{b-1}$, where $\beta_{b-1} = 1$. A tuple is used to describe each operation,

$$(\alpha, \beta, p, q, k, j). \quad (3.30)$$

Here, the symbols α and β belong to the set $\{0, 1, 2, 3\}$ and represent one of the four Pauli matrices. The transformation acts on the first qubit, denoted by p , where p is an integer ranging from 0 to $n - 1$. The value of q must be an integer between 0 and $n-2$, inclusive, for the second qubit. Additionally, q cannot have the same value as the first qubit. The feature parameterizing the transformation is denoted by k , which takes values from 0 to $d - 1$. The bandwidth value is denoted by j , which takes values from 0 to $b - 1$. It is important to mention that certain studies do not directly utilize features $\mathbf{x}_j$ as the rotational angle of a parameterized gate, e.g. [21] utilizes angles denoted as $\theta_{i,j} = (\mathbf{x}_i - \pi)(\mathbf{x}_j - \pi)$. The desired outcome can be achieved in our framework by incorporating the engineering features into the feature vector $\mathbf{x}$ as additional components.

Based on the provided information, we may now design the most fundamental type of quantum kernel, which relies on the ‘overlap test’ as described in Equation 2.73. The feature map $U(\mathbf{x})$ is defined as the product of m functions $R_{\alpha_g, \beta_g}^{(p_g, q_g)}(\beta_{j_g} \mathbf{x}_{k_g})$. This feature map encodes the two data points $\mathbf{x}$ and $\mathbf{x}'$ for which we want to compute the kernel. Figure 3.2 shows the circuit that corresponds to this. To compute the inner product, we do measurements in the computational basis and determine the likelihood of obtaining the measurement outcome $|0^n\rangle\langle 0^n|$. An alternate method for calculating the inner product is the ‘swap test,’ which is illustrated in Figure 3.3 (refer to Equation 2.73).

Partial inner products allow us to verify similarity in a much smaller Hilbert space than the n -qubit space, resulting in a kernel function with lower expressivity. Improved

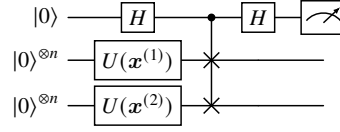


Figure 3.3: The ‘swap test’ quantum circuit is employed to estimate the inner product of two vectors $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}$ that are encoded using the unitary transformation U . The overlap test utilizes a total of $2n + 1$ qubits, as opposed to n qubits, but nevertheless yielding circuits with reduced depth. To calculate the inner product, we can use the method in Equation 2.74 by estimating the chance of obtaining a measurement result of 0 on the topmost qubit. In addition, by utilizing the controlled operation of the swap test on a subset of the n target qubits in each data-related state, we may compute partial inner products that can be understood as a projected quantum kernel (see to Equation 2.73).

generalization performance can result, as explained in Section 3.3.1. A string $s = 0, 1^n$ identifies which qubits to measure, with $s_i = 1$ for the i -th qubit. This approach is comparable to the *projected quantum kernel*, introduced in [44].

A quantum kernel, which has been simulated using our approach, is depicted in Figure 3.4. Each object is defined as a vector $\mathbf{k}$ belonging to the set $Q\mathcal{K}_{n,m}$, which is a subset of $\mathbb{N}^{6m+n}$. The search space has a dimensionality

$$(4 \times 4 \times n \times (n - 1) \times d \times b)^m \times 2^n. \quad (3.31)$$

3.4.2 Criteria

In the preceding section, we emphasized the need of creating criteria for assessing a quantum kernel $\mathbf{k} \in Q\mathcal{K}_{n,m}$. These criteria have a twofold function: they can offer advice during the process of optimization, as seen in Equation 3.27, and they can also be utilized to evaluate potential solutions, either validating their efficacy or exposing their deficiencies.

Three crucial factors have been found in the context of quantum kernels. The initial aspect concerns the *efficient classical simulability*, which refers to transformations that may be achieved on classical hardware. While these transformations may not be entirely useless, they do not provide major advantages in the context of quantum computing. The second issue is *expressivity*, as unitaries with a high degree of expressiveness might result in undesirable statistical features and should be avoided (see Section 3.3). The third aspect is *kernel compatibility*, which refers to the degree to which the selected

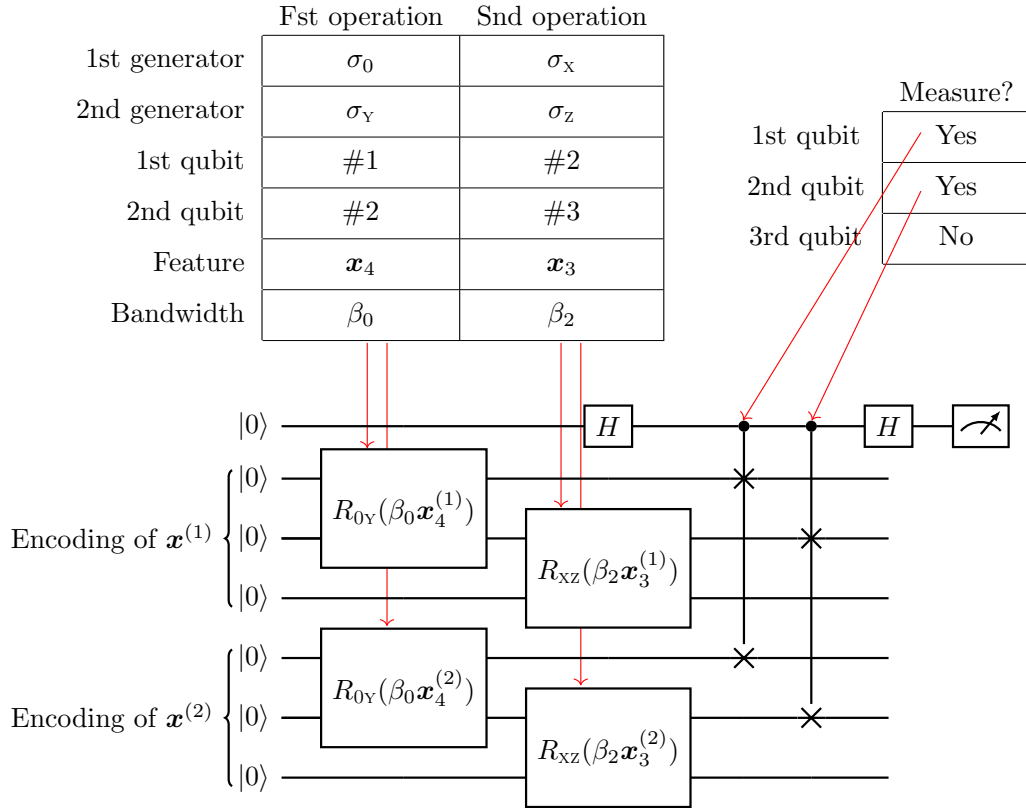


Figure 3.4: Example of a quantum kernel implemented following the proposed approach. The quantum kernel is defined by a feature map that acts on a Hilbert space with a total of 3 qubits. It is composed of 3 operations. Each sample is denoted as $\mathbf{x} \in \mathbb{R}^5$ (just the last two characteristics are utilized in this example). Although the specific values for bandwidth are not mentioned, a typical approach is to assign $\beta_i = i/10$ for $i \in 1, \dots, 10$. This is a projected quantum kernel, where the measurement is conducted on a subset of the three qubits.

kernel is suitable for the specific task being performed. It is important to mention that it is fully possible to build quantum kernels that are difficult to simulate and have moderate expressiveness, yet do not work for specific tasks, as stated by the "no free lunch theorem". Indeed, numerous quantum kernels exhibit inferior performance compared to classical ones across a range of activities. Despite demonstrating progress, understanding the underlying reasons for the outcomes generated by a certain quantum circuit can be a difficult task. An automated technique can provide a clear explanation for the exceptional performance of a particular quantum kernel, demonstrating that the findings satisfy, to a certain degree, all the important requirements we consider necessary for a practical quantum kernel. In this context, we present a number of valuable criteria, elucidating their objectives and providing a quantitative assessment of their computational burden.

Norm of the super-operator A

The norm of the super-operator A as defined in Equation 2.68 quantifies the ability of a quantum transformation $U(\cdot)$ to express different quantum states. It can also serve as a measure of the ability of a quantum kernel to express quantum information. When specifically created, this operator has a size of $2^{n^t} \times 2^{n^t}$. Even when $t = 2$ and for small values of n , the computation of it is generally not possible.

Rank of the dynamical Lie algebra

The dynamical Lie Algebra (DLA) linked to the quantum transformation $U(\mathbf{x}) = \prod_{g=1}^m R_{\alpha_g, \beta_g}^{(p_g, q_g)}(\beta_{j_g} \mathbf{x}_{k_g})$ is associated with the quantum transformation. A vector space over $\mathbb{C}$ is formed with a bilinear operation called the commutator, represented by $[\sigma, \eta] = \sigma\eta - \eta\sigma$, by the span of generators of U , which consist of any $\sigma_{\alpha_g}^{(p_g)} \sigma_{\beta_g}^{(q_g)}$ as well as generators obtained by repeatedly commuting these elements. Larocca et al. [56] have demonstrated that a unitary system with a corresponding DLA of dimensions $O(\exp(n))$ exhibits a high level of expressiveness. This criterion serves as a rough measure of expressivity. When the rotational angles of individual gates are constrained, such as when employing a bandwidth, it is feasible to obtain a unitary that is only moderately expressive but has a DLA with an exponentially large size. However, this measure is highly valuable, as it allows for the quantification of expressivity and the assessment of the computing complexity of classical simulation. Indeed, as proven in the study by Somma et al. [55], a quantum computation utilizing a DLA of polynomial size can be effectively emulated on a classical device. The reason we decided to use

generators instead of explicit gates like CNOT is that it is easier to manage the related DLA in our approach.

From an algorithmic standpoint, it is currently unresolved to ascertain the rank of a DLA in a polynomial time frame with respect to n , in a general sense. The iterative strategy, which consists of continually applying previously identified generators until a fixed point is reached, exhibits exponential growth with respect to n , as there can be 4^{n-1} potential generators. Nevertheless, we can coarse-granely approximate this quantity by setting a predetermined threshold T and classifying every unitary as ‘hard to simulate’ if its DLA surpasses the threshold.

(Centered) Kernel-target alignment

The Kernel-Target alignment (KTA), as proposed by Cristianini et al. [50], seeks to measure the degree to which a kernel function is suitable for a certain job. Consider the training set denoted by $\{(\mathbf{x}^{(j)}, y^{(j)})\}_{j=1}^m$. The KTA is determined by taking the (normalized) Frobenius inner product between the kernel Gram matrix K , which is composed of the kernel function values between pairs of data points, and the matrix Y , which is the outer product of the label vector y . Cortes, Mohri, and Rostamizadeh [51] subsequently improved the metric by introducing the *centered* KTA, which involves subtracting the average value from both matrices. Their experiments have shown that this altered measure displays a more pronounced link with the generalization error in most tasks. Furthermore, this strategy provides a cost-effective and practically efficient tool for measuring kernel compatibility.

Task-model alignment

The task-model alignment, as proposed by Canatar et al. (2021), presents a different method for measuring the compatibility of kernels. The method is grounded in the principle of *spectral bias*, which posits that a kernel function κ can be broken down into its real, non-negative eigenvalues $\{\lambda_j\}_{j=0}^{\infty}$ (arranged in ascending order) and their corresponding eigenfunctions $\{\phi_j\}_{j=0}^{\infty}$. It suggests that the components linked to larger eigenvalues are more amenable to learning and necessitate fewer samples. Therefore, by decomposing the target function of our task into eigenvalues and eigenvectors,

$$f(\mathbf{x}) = \sum_{j=0}^{\infty} w_j \sqrt{\lambda_j} \phi_j(\mathbf{x}), \quad (3.32)$$

we can deduce that when the weight w_j associated with the first components of f is large, it suggests that the target function can be learned with greater ease. The idea of task-model alignment measures the extent to which the target function's power is concentrated in the top k kernel eigenfunctions,

$$T(k) = \frac{\sum_{j=0}^k \lambda_j w_j^2}{\sum_{j=0}^{\infty} \lambda_j w_j^2}. \quad (3.33)$$

It is crucial to note that the spectral bias can be utilized to elucidate the reasons behind the success or failure of a certain kernel function by analyzing the contributions of its separate components. If it is not feasible to perform the eigendecomposition analytically, a numerical method can be employed by utilizing the kernel Gram matrix K and the labels $\mathbf{y}$. This process is illustrated in Canatar, Bordelon, and Pehlevan [52, Methods].

Validation error

An efficient way to evaluate the efficiency of a quantum kernel is by calculating the accuracy of the classifier using a validation test of i.i.d. sampled data points and labels. Although this strategy can yield positive results, it is not without its limitations. It necessitates the training of the corresponding model, which can introduce excessive overhead that may prove burdensome for the given task. Furthermore, it does not offer any information about the classifier's performance other than identifying the specific group of samples that it fails to categorize accurately.

3.4.3 Optimization

The optimization phase finds a candidate solution for Equation 3.28. As anticipated, due to the cost function's non-convexity, we can only apply heuristic optimization methods with no guarantee of optimality. Independent of any optimization technique is our approach. We implemented several in our software.

Greedy approach

Our simplest heuristic is the greedy optimization technique. The implementation begins with the initial candidate given as input and proceeds by iterating through all the operations indexed from $i = 1$ to m . During each iteration, we modify the initial cell of the candidate solution and choose the one that minimizes the cost function. During this

disturbance, the other cells stay unaltered. Afterwards, we proceed to stabilize the initial cell and systematically disturb and stabilize the successive cells to their locally optimal values.

Although this strategy is simple, it does not necessarily result in the optimal solution. However, it is especially advantageous during the discovery phase, since its small computing expense enables the examination of many configurations. The greedy optimizer has been developed independently and does not require any external dependencies apart from the Python3 library. The runtime may be calculated as the product of $(4 + 4 + n + (n - 1) + d + b)$ and m , where n represents the number of qubits, m represents the number of operations, d represents the number of features, and b represents the number of bandwidth values. The runtime must be compounded by the cost of evaluating the candidate solution, which ultimately relies on the selected criteria.

Genetic algorithms

Genetic algorithms are a nature-inspired optimization process. An initial population of individuals is created randomly; then, this population undergoes a fixed number of generational changes. This iterative process consists of three phases: selection, crossover, and mutation. The selection process chooses a subset of the current generation to be processed, which can either form the new generation or replace their parents in the current generation. Selection is usually based on a fitness function (negative of a cost). The crossover operation generates new individuals, or children, by combining components from the selected individuals, or parents. Finally, the children can undergo random perturbation in the mutation phase.

At the end of the iterative process, the candidate solution with the highest fitness is considered the sub-optimal solution to the problem. Describing the variants of this process is beyond the scope of this thesis; for further details, we refer the reader to the review by Katoch, Chauhan, and Kumar [70].

Bayesian optimization

Bayesian optimization is a probabilistic method particularly suited for black-box, low-dimensional optimization tasks where the target function is computationally expensive to calculate (an example of this could be a loss function corresponding to the generalization error of a model with respect to a validation set). It is composed of two elements: the *probabilistic model* of the target function, such as a Gaussian process, which attempts

to replicate the most probable target function given the pairs (x, y) of input and output computed so far, and the *acquisition function*, which selects the most convenient unseen input-output pairs to calculate. The decision made by the acquisition function is always a compromise between exploration, i.e., visiting large unknown regions of the input space in the hope of finding a more favorable solution, and exploitation, i.e., visiting regions close to the sub-optimal solution found so far to identify the local minima in such a neighborhood.

Reinforcement learning (RL)

Reinforcement learning-based optimization is a machine learning technique where an *agent* explores an environment and makes informed choices according to a policy determined during its training phase. Once trained, the agent’s actions resemble those of a game player: each step places an additional gate on the quantum circuit (which can also be an identity gate), and the process concludes once the m gates of the quantum circuit are selected.

The success of the process depends, among many factors, on the choice of the algorithm that determines the policy the agent will follow. Many possible approaches have been proposed in the literature, as presented in the review by Matsuo et al. [71]. Our framework implements the SARSA (State-Action-Reward-State-Action) policy for its simplicity of implementation.

3.5 Detection of beyond-Standard Model events

We employed a kernel-based HEP anomaly detection task as a benchmark. A paramount task in high-energy physics is to determine the validity of new physical processes that overcome the limitations of the so-called *Standard Model* (SM), the theoretical framework developed in the 1970s for particle physics. Several models extending the SM have been proposed, and they are collectively denoted as *Beyond Standard Model* (BSM). Being able to recognize in experimental data which events can be related to the SM and which are anomalous (not explained by the SM) will provide insight into which BSM models best capture the actual physical processes (Woźniak et al. [72]). In this section, we aim to automatically construct a quantum kernel machine to tackle the anomaly detection task of identifying BSM events.

This task has been chosen because previous works by Woźniak et al. [72] and

Schuhmacher et al. [73] have shown statistical evidence of the superiority of quantum kernels over classical ones.² They have demonstrated sufficiently high accuracy in identifying BSM events generated from certain selected BSM frameworks. In these experiments, the model is always the One-Class SVM and only the choice of the kernel function changes. Our goal here is to determine whether the automatic approach proposed can obtain equally good or even superior solutions without having to perform the same trial-and-error approach to identify the quantum kernel.

The task in detail After the discovery of the Higgs boson, which has fundamentally motivated the LHC physics program, some theoretically predicted events have not been observed in experimental data. This puzzle has further motivated the need to analyze experimental data from the accelerator to identify anomalous events and to explain them theoretically as well.

Working directly on experimental data is far from ideal, so current research has focused on generating synthetic events from certain theoretically proposed BSM physical processes using Monte Carlo-based techniques and comparing these data with those from the SM. Here, we denote the BSM-generated data as the *signal* and the SM-generated data as the *background*. The synthetic generation of events provides a significant advantage: the dataset is labeled by construction, giving us a natural way to evaluate the efficacy of machine learning models.

3.5.1 Dataset

The dataset is based on artificially generated data, sampled from a Monte Carlo-based algorithm simulating the SM and some BSM processes. An autoencoder model then converts the high-dimensional data into a lower-dimensional latent representation for loading and analysis on quantum devices.

The datasets used in this study are available for public access via Pierini and Wozniak [74]. They pertain to dijet collisions, where each jet has 100 particles with three spatial coordinates, totaling 300 features. We have considered three BSM signals, or "anomalies," as benchmarks: 1) the decay of *narrow* and 2) *broad Randall-Sundrum gravitons into W-bosons*, as well as 3) the decay of a scalar boson A into Higgs and Z bosons ($A \rightarrow HZ$). These events are generated with the software PYTHIA [75]. To

²Note: in these experiments, machine learning models other than kernel machines are not taken into account.

simulate the effect of the detector, DELPHES [76] is used.

In this form, the data is high-dimensional and challenging to tackle with both classical and quantum machine learning models. Therefore, a dimensionality reduction technique must be used. We avoid using PCA as it is considered too simplistic and unable to preserve the complex, non-linear behavior of the task. Instead, a traditional autoencoder is trained to compress particle jet objects without labels on the large dataset. The autoencoder output is a vector living in the *latent space* $\mathbb{R}^\ell$. The work by Woźniak et al. [72] has considered multiple values of ℓ ; however, due to the computational cost of simulating quantum models, only $\ell = 4$ and $\ell = 8$ are considered in our experiments.

Finally, the dataset is $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^m$, where $\mathbf{x}^{(i)}$ represents the (autoencoded) features of a di-jet (ℓ features per jet $\times$ 2 jets colliding) and $y^{(i)}$ is either SM or BSM.

3.5.2 Experimental setup

Our experiment has two phases: *discovery* and *assessment*. The first phase consists of running the automatic approach proposed beforehand, leading to the identification of a function κ . The second phase involves using that kernel function to train and assess the machine learning model’s performance.

Notably, all experiments were conducted using the *quask* framework [3] in a noiseless environment. This setting allows for a fair evaluation of whether unsatisfactory results arise from an unsuccessful optimization process. Introducing noise at this stage would obscure the identification of such potential issues, making it difficult to pinpoint the underlying causes. We emphasize that, since this work targets NISQ devices, it is crucial to also evaluate performance in more realistic, noisy scenarios. However, we leave such an analysis for future work.

Discovery

The kernel discovery phase is an iterative, heuristic optimization process, and the cost function must be relatively inexpensive. While we anticipate that the assessment phase can rely on a large amount of training data, the discovery phase must be significantly more computationally efficient. Consequently, we use a reduced dataset during this phase, which means that decisions about the kernel function may be based on limited information. This trade-off is a necessary compromise to ensure that the overall runtime remains feasible.

Here, we employ a Bayesian optimization (BO) technique, which is well-suited for complex black-box optimization problems such as the one considered. BO is advantageous because it makes informed decisions about which data points to evaluate, unlike, for example, genetic algorithms, which rely more heavily on exploratory sampling. Since Bayesian optimization is itself an iterative process, we experimented with both 5 and 10 iterations. In each iteration, 5 points were sampled using the acquisition function.

The optimization is initialized with the identity matrix. Kernel discovery is guided by a cost function based solely on classification accuracy over a small validation set $\mathcal{D}'$, though other cost functions incorporating multiple criteria could also be explored. The cost is defined as

$$C(\mathbf{k}) = \frac{1}{|\mathcal{D}'|} \sum_{(\mathbf{x}, y) \in \mathcal{D}'} \mathbb{1}(f_{\mathbf{k}}(\mathbf{x}) = y). \quad (3.34)$$

Here, $f_{\mathbf{k}}$ denotes the kernel machine (OC-SVM) trained on a subset of 75 data points sampled i.i.d. from the original dataset, and $\mathbb{1}(\cdot)$ denotes the evaluation of a predicate.

The optimization is parameterized by the number of qubits, the latent dimension (i.e., the number of features of the data points), and the number of operations. For simplicity, we set the number of qubits equal to the latent dimension, which we recall are $\ell = 4$ or $\ell = 8$. The number of operations was fixed to either $m = 8$ or $m = 12$. These values were chosen to balance the trade-off between expressiveness and optimization tractability, given the exponential growth of the search space and the potential drop in fidelity on NISQ devices when increasing the circuit depth.

A more extensive comparison using different optimization algorithms is provided in Incudini et al. [7, Supplementary Material 5].

Assessment

The second phase assesses the quality of the candidate solutions. The assessment will compare the kernel machine with the identified kernel against all other classical and quantum kernel machines, tested in the work by Woźniak et al. [72]. In contrast to the first phase, the second phase is not iterative and therefore we can use larger datasets to thoroughly evaluate the model. Specifically, we use the identified kernels to train an instance of OC-SVM over a much larger dataset consisting of 200 elements for the training set and 1500 elements for the testing set. These are divided into 5 sub-datasets of 300 elements to calculate the mean and standard deviation of the accuracy.

Approach	Dataset Narrow G	Dataset A $\rightarrow$ HZ	Dataset Broad G
Best classical kernel (in [72])	99.54 ± 0.42	97.94 ± 0.69	50.90 ± 3.97
Best quantum kernel (in [72])	99.65 ± 0.23	98.05 ± 0.58	55.20 ± 3.96
This work ($m = 8, t = 5$)	99.28 ± 0.58	97.66 ± 0.61	47.62 ± 4.17
This work ($m = 8, t = 10$)	99.28 ± 0.58	97.66 ± 0.61	61.61 ± 4.67
This work ($m = 12, t = 5$)	99.37 ± 0.46	94.57 ± 1.60	58.70 ± 4.38
This work ($m = 12, t = 10$)	99.73 ± 0.20	98.34 ± 0.37	58.70 ± 4.38

Table 3.1: Comparison of kernel machines using classical kernels, quantum kernels fixed a priori, and quantum kernels chosen via the discovery process in terms of accuracy over a testing set (best results only). Here, the dataset has a latent dimension ℓ . The best-performing quantum kernel in Woźniak et al. [72] uses 2ℓ rotational gates and $\ell - 1$ CNOT gates. The best-performing discovered kernel uses m two-qubit rotational gates, $m \in \{8, 12\}$, and is found after 5 or 10 iterations of BO. Best-performing classifier in bold.

3.5.3 Results

We evaluate the performance of the models with respect to the *Area Under the Curve* (AUC) of the *Receiver Operating Characteristic* (ROC) curve. Such a plot shows the value of True Positive Rate (TPR) over the False Negative Rate (FNR) with respect to the change in a certain criterion value, which in our case is the outlier proportions ν .

The analysis focuses on a latent dimension of $\ell = 8$, the number of operations $m \in \{8, 12\}$, and iterations of the BO process $t \in \{5, 10\}$. Table 3.1 presents the results. We outperform classical and quantum kernels for the Narrow G and A $\rightarrow$ HZ datasets. However, such a results are found when the number of gates is higher, $m = 12$, while for $m = 8$ the performances are sub-optimal. Notably, Broad G dataset is problem that is harder to analyse and also is the one that benefit the most from our approach, and get the best results with fewer operations ($m = 8$). The motivation for holds in the fact that the optimization for $m = 8$ is cheaper and more effective, despite the family of circuit being less expressive, and is not unrealistic that the easiest optimization well compensate the other aspects. Even with a higher value of $m = 12$, we have a higher AUC than both methods being compared, despite the increased complexity.

ROC curves, as shown in Figures 3.5 and 3.6, can provide a more comprehensive performance evaluation. Note that these results are compared to the one obtained by analyzing these tasks with classical kernels and some fixed quantum kernels known in the literature (we do not compare these results with other models, i.e., neural networks).

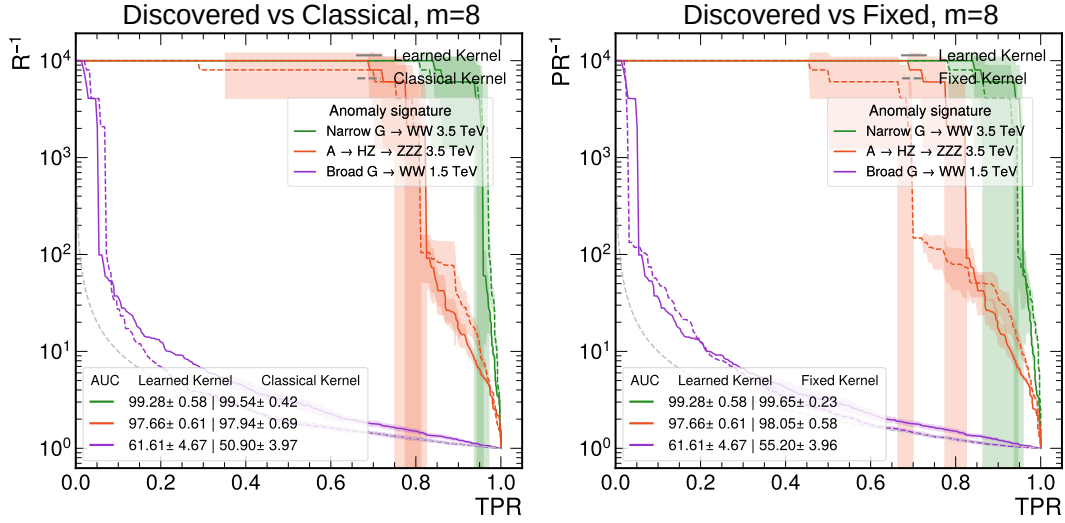


Figure 3.5: Comparison of the performance of the automatically discovered quantum kernels (solid line), the best-performing classical kernel (dashed line, plot on the left), and the best-performing quantum kernel (dashed line, plot on the right) in terms of the ROC-AUC curve over the evolving parameter ν characterizing the one-class SVM. The data points belong to the latent space with $\ell = 8$, and the automatically discovered kernels have quantum circuits with $m = 8$ gates.

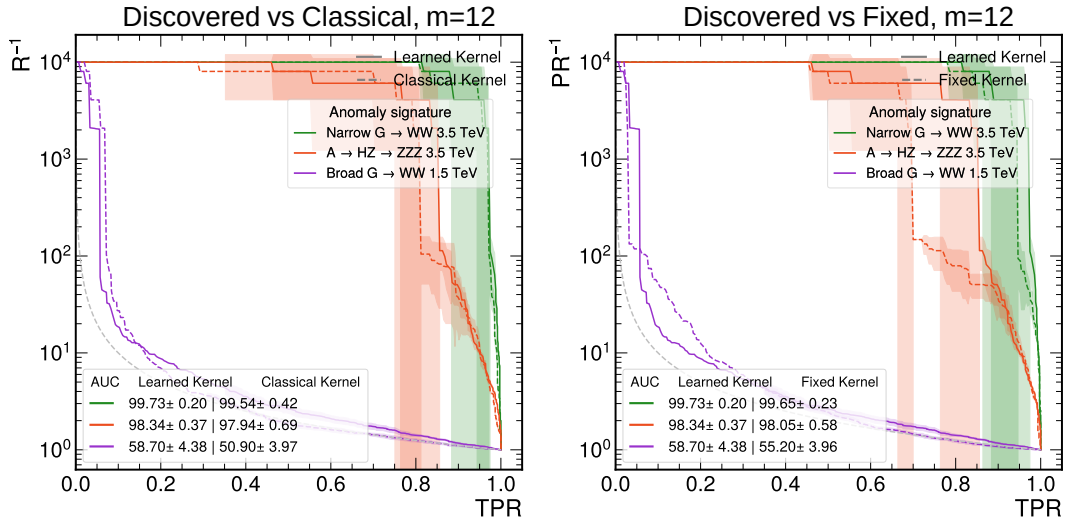


Figure 3.6: Comparison of the performance of the automatically discovered quantum kernels (solid line), the best-performing classical kernel (dashed line, plot on the left), and the best-performing quantum kernel (dashed line, plot on the right) in terms of the ROC-AUC curve over the evolving parameter ν characterizing the one-class SVM. The data points belong to the latent space with $\ell = 8$, and the automatically discovered kernels have quantum circuits with $m = 12$ gates.

Testing more configurations

Up to now, we have tested the automatic discovery approach below its potential. In fact, the optimization process has been guided by a cost function depending solely on the model’s accuracy over a small validation set. However, we can gain many insights into the kernel by focusing on other criteria. Furthermore, we have only tested the performance of the Bayesian optimizer. While trying all possible configurations is infeasible and beyond the scope of this work, trying and comparing more configurations might give us novel insights into the tasks. Here, we test both cost functions: 1) validation accuracy and 2) a linear combination of validation accuracy and task-model alignment, using the optimizers A) Bayesian optimization and B) the reinforcement learning-based optimizer. These four possible configurations are tested on all three datasets.

In Figure 3.7, we show the results for the Narrow G dataset. We observe the RL engine performing poorly for a small number of iterations; furthermore, there is a performance difference of 1% to 3% between the two cost functions depending on the specific configuration, for which we have not found a convincing explanation. Such minor improvements could be the result of luckier training.

In Figure 3.8, we show the results for the $A \rightarrow HZ$ dataset. Similar to the previous case, the optimizers perform comparably once trained for a sufficiently high number of epochs, and the difference between the criteria is negligible.

In Figure 3.9, we show the results for the Broad G dataset. Changing the configuration has a negligible effect on kernels with $m = 12$ gates. However, the effect is much more evident for smaller quantum circuits: BO and validation accuracy perform significantly better. At the same time, RL is usually disadvantaged by the stochasticity of the optimization and the small size of the validation set.

These experiments provide a partial answer regarding how to configure the automatic discovery process. The choice of the number of gates m is essential. Still, we need to find out a priori if our specific task should have fewer operations (more effective training, lower expressivity) or a higher one (more flexibility). Bayesian optimization is the best choice because it is relatively inexpensive and does not require extra configuration or hyperparameter optimization. Using KTA and TMA instead of validation accuracy might lead to slightly more efficient optimization with negligible adverse effects.

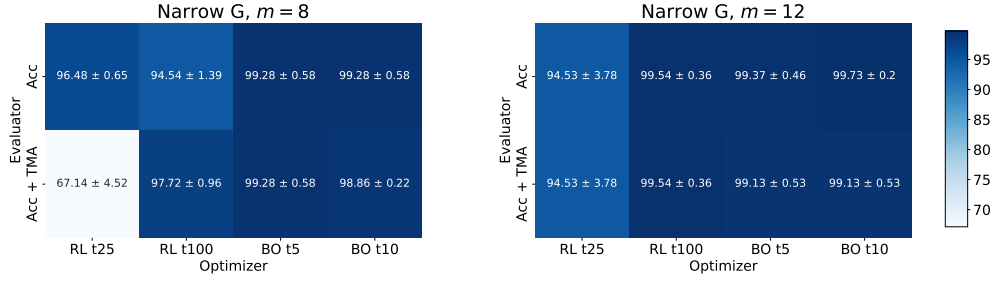


Figure 3.7: Comparison of the performance of the automatically discovered quantum kernels over different configurations of the optimization process on the Narrow G dataset. The performance is stated in terms of AUC-ROC. The left column shows results for $m = 8$, and the right column shows results for $m = 12$. Experiments with RL (SARSA) have been conducted for 25 and 100 iterations (episodes).

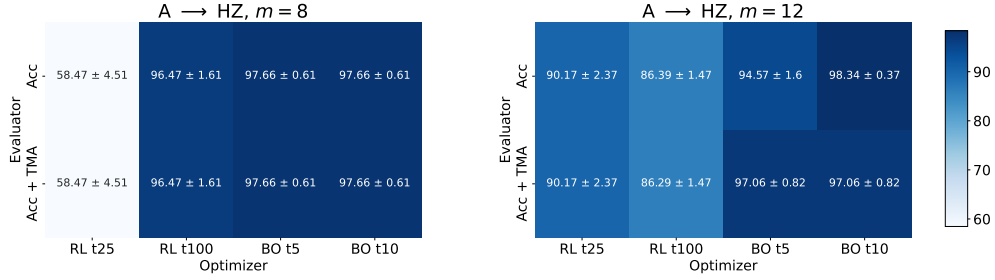


Figure 3.8: Comparison of the performance of the automatically discovered quantum kernels over different configurations of the optimization process on the A $\rightarrow$ HZ dataset. The performance is stated in terms of AUC-ROC. The left column shows results for $m = 8$, and the right column shows results for $m = 12$. Experiments with RL (SARSA) have been conducted for 25 and 100 iterations (episodes).

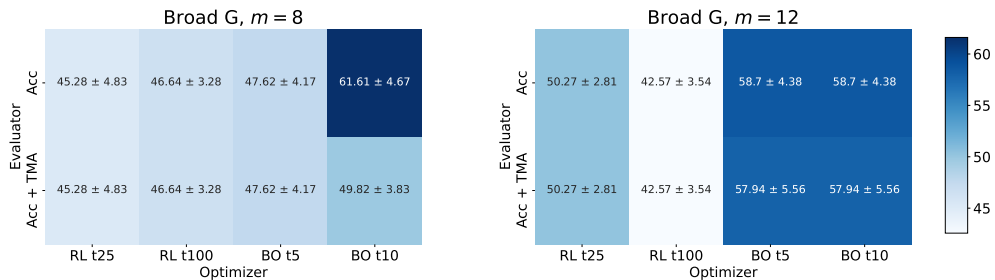


Figure 3.9: Comparison of the performance of the automatically discovered quantum kernels over different configurations of the optimization process on the Broad G dataset. The performance is stated in terms of AUC-ROC. The left column shows results for $m = 8$, and the right column shows results for $m = 12$. Experiments with RL (SARSA) have been conducted for 25 and 100 iterations (episodes).

Chapter 4

Toward “deep” quantum machine learning models

A critical aspect of NISQ-related quantum machine learning research is determining whether these models can be applied to a variety of popular real-world scenarios, as is the case with classical machine learning, or whether they are only useful for a limited number of particularly structured tasks. The path to answering this involves comparing quantum kernel machines (and quantum neural networks) with the most commonly used models, such as deep neural networks.

To explore this topic, we first identify the characteristics that contribute to the unprecedented success of deep models. In Section 4.1, we highlight three main points: the ability to perform *feature learning* (i.e., changing the features during training), having a *hierarchical structure* where features become more refined at each level of the model or network, and being *over-parameterized* (i.e., able to easily interpolate the entire training set). In Section 4.2, to investigate feature learning capabilities, we introduce the Neural Tangent Kernel (NTK). In Section 4.3, to examine hierarchical feature learning, we present the Path Kernel (PK). We then discuss their impact on quantum machine learning, with the NTK demonstrating that QNNs perform feature learning (Section 4.4) and the PK showing how to achieve a hierarchical representation in quantum models (Section 4.5). Finally, in Section 4.6, we explore how to over-parameterize QNNs by using ensembles of smaller models, thus avoiding the need for large, expensive models. This chapter is based on [4, 2].

4.1 Characteristics of deep machine learning models

4.1.1 Feature learning

The most important characteristic of popular deep learning models is their ability to perform *feature learning* or *representation learning*. Feature learning refers to the model's capability to identify the best representation of data features during training. In a feature learning model, the features are both learned and used to solve the task. This contrasts with *feature engineering*, where the developer manually selects and encodes the features used for the learning task. The archetype of a feature engineering model is the kernel machine, where we manually choose the feature map, while the archetype of a feature learning model is the neural network, which is fed with raw data and acts as a black-box¹.

The concept of feature learning is intertwined with the one of *non-linearity* in a machine learning model. In simple terms, whenever a machine learning model exhibits non-linearity, it is a feature learning model and vice versa. The kernel machine, often referred to as a linear model, does not exhibit feature learning, while the neural network is nonlinear (whenever its activation function is as such) and, for that, exhibit feature learning. However, non-linearity is an overloaded term; for example, a kernel machine is sometimes referred to as having non-linear boundaries in contrast to the linear regressor.

To better clarify these concepts, we delve into a more formal treatment of the subject and present the definitions of *linear model*, *nonlinear model*, and *feature learning*. This formalization was first presented in the book by Roberts, Yaida, and Hanin [77].

A model is *linear with respect to its trainable parameters* θ if it can be written in the form:

$$f(\mathbf{x}; \theta) = \sum_{j=1}^p \theta_j \phi_j(\mathbf{x}), \quad (4.1)$$

where $\{\phi_j : \mathcal{X} \rightarrow \mathbb{R}\}_{j=0}^p$ are the *feature functions*, whose values correspond to the model features. Note that this model captures a kernel machine whenever the feature functions correspond to $\phi_j(\mathbf{x}) = \kappa(\mathbf{x}, \mathbf{x}^{(j)})$ for $\mathbf{x}^{(j)}$ being an element of the training set, $j = 1, \dots, n$. Furthermore, we might consider an additional feature $\phi_0 \equiv 1$ that incorporates the bias.

Equation 4.1 depict a linear model, in the sense that the function is linear with respect

¹Clearly, this is an exaggerated simplification. In a real-world scenario, the feature engineering phase is paramount even whenever neural network-based models are employed.

to the *parameter space*, $\mathcal{P} = \mathbb{R}^p$, and it can be rewritten as an inner product in such a space,

$$f(\mathbf{x}; \boldsymbol{\theta}) = \langle \boldsymbol{\theta}, \boldsymbol{\phi}(\mathbf{x}) \rangle_{\mathcal{P}}, \quad (4.2)$$

with $\boldsymbol{\theta} = (\theta_1, \dots, \theta_p)$ and $\boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_p(\mathbf{x}))$. In this sense (i.e., with respect to the training parameters), kernel machines are linear. Whenever we refer to the fact that kernel machines are non-linear, we intend that *with respect to the input space*: this means that the inner product is in the form $\langle \cdot, \cdot \rangle_{\mathcal{P}}$ and not in the form $\langle \cdot, \cdot \rangle_{\mathcal{X}}$.

A model is *non linear with respect to its trainable parameters* $\boldsymbol{\theta}$ if it can be written in the form:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \sum_{j=1}^p \theta_j \phi_j(\mathbf{x}) + \frac{\epsilon}{2} \sum_{j,k=1}^p \theta_j \theta_k \psi_{j,k}(\mathbf{x}) + \frac{\epsilon}{3!} \sum_{j,k,\ell=1}^p \theta_j \theta_k \theta_\ell \psi_{j,k,\ell}(\mathbf{x}) + \dots \quad (4.3)$$

The model is characterized by the first-order feature functions $\{\phi_j\}_{j=1}^p$, but also the second-order ones $\{\psi_{j,k}\}_{j,k=1}^p$ and up to arbitrary orders. Note that these functions are unique up to a permutation of their variables (e.g., $\psi_{j,k} = \psi_{k,j}$), thus the terms $1/n!$ in Equation 4.3 compensate for the elements counted multiple times. The strength of the nonlinearity is controlled by a parameter $\epsilon \in (0, 1]$.

Now, we can define feature learning more rigorously. For the sake of simplicity, we will concentrate on a basic example. Consider the quadratic model [78], corresponding to an instantiation of Equation 4.3 truncated to the quadratic term. In this case, the loss function is quartic and generally non-convex.

The parameter dynamics of the quadratic model are given by:

$$f(\mathbf{x}, \boldsymbol{\theta} + d\boldsymbol{\theta}) \quad (4.4)$$

$$= f(\mathbf{x}, \boldsymbol{\theta}) + \sum_{j=1}^p d\theta_j \left[\phi_j(\mathbf{x}) + \epsilon \sum_{k=1}^p \theta_j \psi_{j,k}(\mathbf{x}) \right] + \frac{\epsilon}{2!} \sum_{j,k=1}^p d\theta_j d\theta_k \psi_{j,k}(\mathbf{x}) \quad (4.5)$$

$$= f(\mathbf{x}, \boldsymbol{\theta}) + \sum_{j=1}^p d\theta_j \phi_j^E(\mathbf{x}; \boldsymbol{\theta}) + \frac{\epsilon}{2} \sum_{j,k=1}^p d\theta_j d\theta_k \psi_{j,k}(\mathbf{x}). \quad (4.6)$$

In the last equation, we note that the first-order feature functions depend on and evolve according to the parameters $\boldsymbol{\theta}$; in this case, we denote such functions with ϕ^E and name them *effective feature functions*. This means that in the quadratic model, we are able to have first-order feature functions learned during the optimization phase, while the second-order ones remain constant. Clearly, we can generalize this reasoning to

higher-order terms: having a degree- n term implies the model will have feature functions of degree- n . Models enabling effective feature functions are said to exhibit *feature learning* capabilities.

4.1.2 Hierarchical feature learning

While feature learning is a characteristic property of models nonlinear with respect to the training parameters, without further assumptions on the architectural aspect of the model itself, *hierarchical feature learning* is a characteristic property of deep, layer-wise architectures [79, 80]. The primary aim of such deep architectures is to learn a complex target function as a composition of much simpler functions, their building blocks, discovered during training.

This concept is illustrated beautifully when we focus on Convolutional Neural Networks (CNNs), an extension of the multi-layer perceptron tailored for tasks such as the analysis of images and videos. Zeiler and Fergus [81] has shown that, when applied to image recognition, a CNN discovers patterns of different complexity during training. Specifically, neurons closer to the raw inputs are activated by simpler patterns such as lines and curves, neurons in the middle layers are activated by intermediate complexity patterns such as eyes, petals, and wheels, and neurons closer to the output layer are activated by patterns closely resembling the characteristics of the output signals of the various classes. This concept is illustrated in Figure 4.1.

4.1.3 Over-parameterization

Another aspect typical of deep learning is that the model is *overparameterized*. To explain this concept, we must first introduce the *bias-variance tradeoff*. In statistical learning, the bias-variance tradeoff justifies the performance of a model in terms of its complexity. Models that are too simple will underfit, meaning they will not be able to express the target function, while models that are too complicated will overfit, meaning they will interpolate the points perfectly according to an unrealistically complicated learned mapping. According to this tradeoff, huge models such as deep learning ones should tremendously overfit and thus generally show an unsatisfactory testing error. However, in contrast, we experience unreasonably effective models. It turns out that, with statistical learning tools, we can visualize only part of this picture: there is a threshold after which the increased complexity of the model results in an improvement

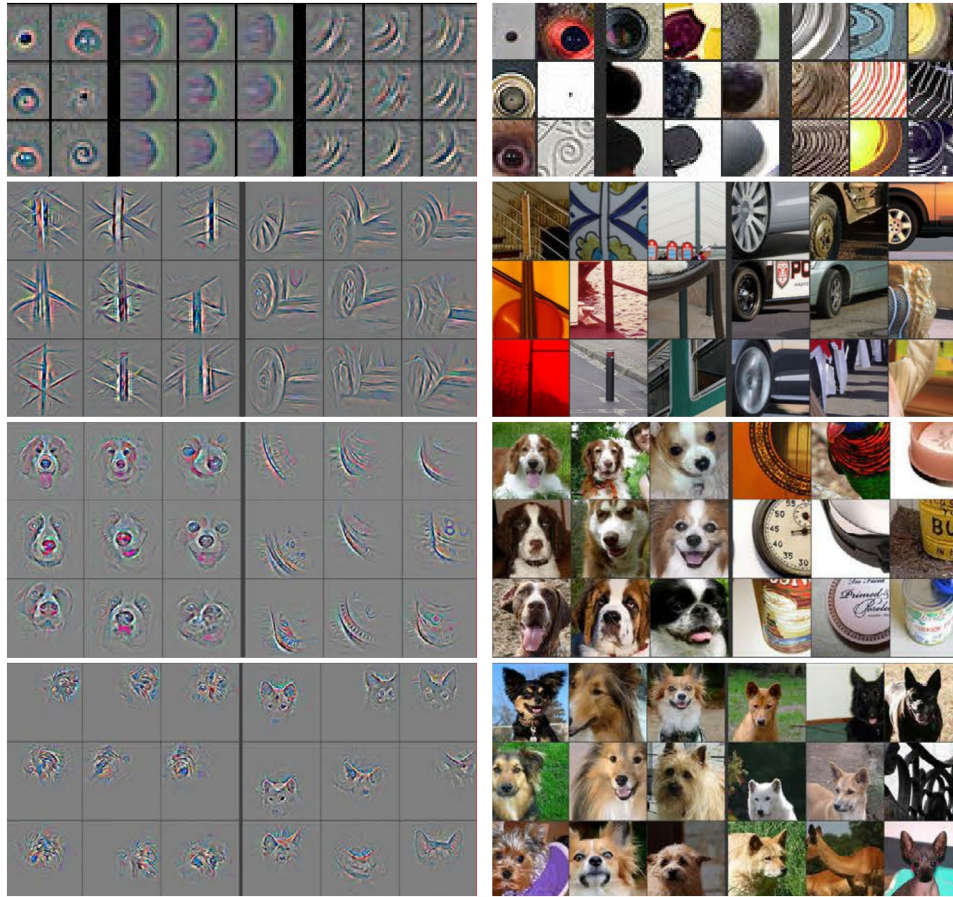


Figure 4.1: Layerwise inspection of the patterns activating the neurons of a convolutional neural network. The model has been trained for image recognition tasks. From top to bottom, the features learned by neurons from layers 2 to 5 are shown on the left, and examples of images activating the neurons are shown on the right. Adapted from [81, Figure 2], reused with permission © 2014, Springer International Publishing Switzerland.

of performance, a regime denominated overparameterized.

We can explain this concept more rigorously. In statistical learning, a model's performance can be decomposed into three components: $MSE = Bias^2 + Variance + Irreducible\ Error$. Here, the *bias* refers to an error that arises when choosing an overly simplistic model, leading to an inability to accurately capture the inherent relationships within the data. Highly biased models are often termed as *underfitting*, exhibiting a high training loss. The *variance* denotes an error connected to selecting an excessively complex model, making it sensitive to even minor fluctuations in the data and potentially capturing the noise in the dataset. High-variance models are known as *overfitting*, characterized by low training loss but high testing loss, indicating poor generalization capabilities. Finally, the *irreducible error* depends uniquely on the inherent noise within the dataset and cannot be eliminated.

The bias-variance tradeoff suggests that a model achieves optimal performance when the number of parameters strikes a balance between simplicity (few parameters, leading to underfitting) and complexity (many parameters, resulting in overfitting). Within this delicate equilibrium, the model performs best. However, modern deep learning practices have challenged the traditional bias-variance tradeoff. These practices often intentionally employ models with many parameters while achieving state-of-the-art performance. Although efforts to formulate a new theory are ongoing, researchers have proposed a phenomenon that explains the behavior of both small models (where the bias-variance tradeoff applies) and large models. This phenomenon is known as *double descent* [82], as illustrated in Figure 4.2.

At the frontier of the traditional statistical regime, denoted as the *interpolation threshold*, models become sufficiently complex to interpolate the data perfectly. However, achieving perfect interpolation often involves employing intricate, unrealistic functions. Even though these functions result in zero training error, they typically lead to a poor generalization error. As we continue to increase the model size, the space of functions capable of achieving zero training error becomes larger. Within this space, some functions are 'simpler' than others. We can still achieve a favorable generalization error if we select the simpler function from this expanded space. Models to the left of the interpolation threshold are called *underparameterized*, while those to the right are called *overparameterized*.

According to the *implicit regularization* hypothesis, training the model via stochastic gradient descent leads to a solution with the minimum L2 norm, representing the simplest solution to the problem. This minimization helps prevent overfitting. However, as of

now, we only have empirical evidence to support this hypothesis. Double descent has been observed across various models, including kernel regression and neural networks.

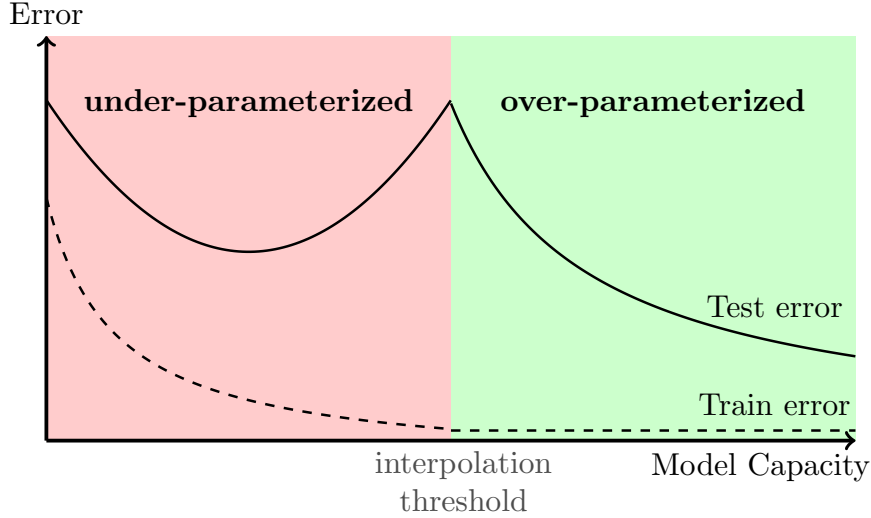


Figure 4.2: Pictorial representation of the double descent phenomenon. In the classical, under-parameterized regime, the test error follows the bias-variance tradeoff rule, which has a sweet spot in model capacity (the model is neither too small nor too big) that exhibits the best performance. At the interpolation threshold, the model can perfectly interpolate the data (the training loss is zero), but in doing so, it overfits. A further increase in model capacity will enter the over-parameterized regime, where the bias-variance tradeoff no longer holds. Any increase in model size will help the optimization process lead, among all the solutions having zero loss, to the one with the smallest norm, thus the simplest function.

4.2 Neural Tangent Kernel

A possible direction for understanding the success of deep machine learning models is by connecting them to kernel machines [83]. Kernel machines (Equation 2.37) are, at least in principle, less powerful than other nonlinear models such as neural networks (Equation 2.53). For example, it was proven in the previous section that feature learning is a characteristic unique to nonlinear models. Meanwhile, overparameterization and double descent phenomena can appear in kernel machines, neural networks, and other models [82, 84]. Can we determine in which cases a linear model is sufficiently powerful to solve a problem and, conversely, when a deep machine learning model is necessary? A particular tool called the Neural Tangent Kernel (NTK, [27]), has been developed to

address this question.

Consider a machine learning model $f(\mathbf{x}; \boldsymbol{\theta})$ and suppose it has undergone training via a stochastic gradient descent optimization algorithm. Therefore, the model can be approximated by a first-order Taylor expansion:

$$f(\mathbf{x}; \boldsymbol{\theta}) \approx f(\mathbf{x}; \boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}} f(\mathbf{x}; \boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0). \quad (4.7)$$

This approximation allows f to be represented as a kernel machine via the NTK:

$$\kappa_{\text{ntk}}(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\mathbf{x}; \boldsymbol{\theta}) \cdot \nabla_{\boldsymbol{\theta}} f(\mathbf{x}'; \boldsymbol{\theta}). \quad (4.8)$$

Whenever f is a neural network in the form of Equation 2.53 in the asymptotic regime of 'infinite width' (i.e., when the width of each layer is much larger than the number of layers), the non-linear contribution of f becomes negligible. In this case, the NTK is independent of $\boldsymbol{\theta}$, meaning the model itself becomes independent of its initial value $\boldsymbol{\theta}_0$, which is chosen randomly at the start of training. Furthermore, as an effectively linear model, the landscape of its loss function becomes convex and easy to optimize to its global optimal solution, a condition sometimes called *lazy training* [85].

It is relevant to understand when a nonlinear model, such as a neural network, is necessary to solve a task or if it is possible to solve it using a linear model, such as a kernel machine, or the linearization of a nonlinear model. From an empirical perspective, this varies from task to task. A notable point is shown in the work of Ghorbani et al. [86]: the authors suggest that whenever a kernel function (the NTK or any other) is suffering from the curse of dimensionality as its feature map is exploiting a large Hilbert space, these will show worst performance compared to neural networks as the latter model can learn a meaningful, low-dimensional feature representation. An example of such a behavior has been noted in Refinetti et al. [87] on the Gaussian XOR mixture problem; this particular task will be analyzed in the subsequent section. Another more concrete example has been investigated by Bai and Lee [88], showing that a neural network can learn sparse polynomials with better sample complexity than its NTK. Finally, Bai et al. [89] has compared the empirically compared performance of a wide neural network and its linearization on the popular benchmark CIFAR-10, showing that the neural network consistently outperforms its linearization for real-world tasks, motivating their use; furthermore, the difference of performance between the two grows when decreasing the width of the model (as expected).

4.3 Path Kernel

The Path Kernel was introduced in [90] to allow any gradient descent-based machine learning model to be transformed into one that (superficially) resembles kernel machines. Its primary motivation is to investigate the role of feature learning: if any model can be led back to a kernel machine, does feature learning exist at all? (This question is ill-posed and a brief comment in this regard is found in Section 4.3.1). Its secondary motivation, which is the focus of our interest, is that the Path kernel induces a hierarchy of parameters in the machine learning model during the training phase, possibly opening the door for a different notion of hierarchical feature learning; the usual one is the hierarchy of parameters from one layer of the neural network to the subsequent one.

More concretely, let h be a machine learning model working on data $\mathbf{x} \in \mathbb{R}^D$ and with parameters living in the space $\mathbb{R}^P$, i.e., $h : \mathbb{R}^D \times \mathbb{R}^P \rightarrow \mathbb{R}$. The Path kernel is defined as

$$\kappa_{\text{pk}} : \mathbb{R}^D \times \mathbb{R}^D \times ([0, T] \rightarrow \mathbb{R}^P) \rightarrow \mathbb{R}, \quad (4.9)$$

$$\kappa_{\text{pk}}(\mathbf{x}, \mathbf{x}_i; \gamma) = \int_0^T \kappa_{\text{ntk}}(\mathbf{x}, \mathbf{x}_i; \gamma(t)) dt, \quad (4.10)$$

Here, $\bar{\gamma} : [0, 1] \rightarrow \mathbb{R}^P$ denotes the parameter path, with the terminal parameter value $\bar{\gamma}(1) = \bar{\mathbf{w}}$. The Neural Tangent Kernel can similarly be expressed as a parametric kernel:

$$\kappa_{\text{ntk}} : \mathbb{R}^D \times \mathbb{R}^D \times \mathbb{R}^P \rightarrow \mathbb{R}, \quad (4.11)$$

$$\kappa_{\text{ntk}}(\mathbf{x}, \mathbf{x}_i; \mathbf{w}) = \nabla_{\mathbf{w}} h(\mathbf{x}; \mathbf{w}) \cdot \nabla_{\mathbf{w}} h(\mathbf{x}_i; \mathbf{w}). \quad (4.12)$$

[90, Theorem 1] shows that h can equivalently be rewritten as a function of the Path Kernel

$$h(\mathbf{x}; \bar{\mathbf{w}}) = h'(\mathbf{x}, \bar{\gamma}) = \sum_{i=1}^m \alpha_i(\mathbf{x}) \kappa_{\text{pk}}(\mathbf{x}, \mathbf{x}_i; \bar{\gamma}) + \alpha_0(\mathbf{x}). \quad (4.13)$$

Note that Equation 4.13 is not a kernel machine; the difference lies as the coefficients α_i having a dependence on $\mathbf{x}$. Equation 4.13 holds under the assumption that h is differentiable with respect to $\mathbf{w}$ and is trained using gradient descent on the given training dataset $\{(\mathbf{x}_i, y_i^*)\}_{i=1}^m \subseteq \mathbb{R}^D \times \mathbb{R}$ with a convex differentiable loss function $L(\mathbf{w}) = \sum_{i=1}^M \ell(h(\mathbf{x}_i, \mathbf{w}), y_i^*)$.

4.3.1 Path Kernel as a tool to investigate the role of feature learning

One of the motivations behind the use of the Path Kernel is to investigate the role of feature learning:

“[...] the most significant implication of our result for deep learning is that it casts doubt on the common view that it works by automatically discovering new representations of the data, in contrast with other machine learning methods, which rely on predefined features (Bengio et al., 2013). As it turns out, deep learning also relies on such features, namely the gradients of a predefined function, and uses them for prediction via dot products in feature space, like other kernel machines.” (Domingos [90])

In simpler terms, do we need feature learning at all if the trained model can be expressed as a kernel machine?

To the best of the author’s knowledge, this question remains unanswered: even if, in theory, we can well-approximate any model with a kernel machine, in practice, the direct construction of such a kernel machine is possible only thanks to the feature learning process that has discovered the best feature functions. This is because the Path Kernel requires the prior training of the model $f(\mathbf{x}, \theta)$ and access to the sequence $\theta_0, \dots, \theta_T$ of parameters during stochastic gradient descent. This non-convex optimization generally does not lead to the global optimum solution of the learning problem. Only once the model f has been trained and feature learning has taken place can we use the Path Kernel to define an approximately linear equivalent model, where the non-linear part is negligible.

In conclusion, the theorem proves (or conjectures, taking into account the comments in Section 4.3.4) that it exists a kernel machine that behaves approximately as the trained model f . However, this does not imply that models exhibiting feature learning are useless, as such a kernel machine cannot be constructed *a priori* without the knowledge of the best features – it is constructed via the Path Kernel only because the training has taken place. Therefore, this vision of the tool has to be considered limited.

4.3.2 Path Kernel as a tool to investigate the role of hierarchical feature learning

Despite the interesting but unclear role of the Path Kernel in justifying the need for feature learning, or the lack thereof, this tool provides us with means to investigate the

role of hierarchical deep learning.

When considering deep machine learning models, we believe that hierarchical deep learning is exhibited only spatially; for example, the features at the j -th layer are built upon the features at the $(j - 1)$ -th layer in neural network-based models. More precisely, the chain rule obtained in the differentiation of h is the source of the hierarchical dependencies among the parameters (it can be as well pictured by looking at the topology of the network: the connection in the direction output to input indicates the dependencies). We observe that, using the Path Kernel, we can also build a *temporal* hierarchy of features, where the features at the j -th step of gradient descent depend on the features at the $(j - 1)$ -th step.

The hierarchy of parameters can be identified by looking at the curve γ , and its morphological changes characterize whenever we have a simple or rich, complex hierarchy. For linear models with vanilla gradient descent training on a convex loss function $\mathcal{L}$, the parameter path follows a linear trajectory: $\{(1 - t)\theta_0 + t\theta_f \mid t \in [0, 1]\}$, where $\theta_0 \in \mathbb{R}^P$ represents the initial parameters and $\theta_f \in \mathbb{R}^P$ represents the parameters at convergence to the (ideally global, realistically it depends from case to case) minimum of the convex loss function. In this scenario, it can be easily verified that the derivative $\nabla_\theta f$ of the linear model is independent of θ , resulting in a constant NTK. For nonlinear models, the loss function $\mathcal{L}$ may be non-convex, leading to a non-linear parameter path γ . In this case, both $\nabla_\theta f$ and the NTK will change over the course of learning.

4.3.3 The Path Kernel is a Mercer kernel

A kernel function can be referred to as such only if it is symmetric and positive semi-definite. In that case, Mercer's theorem allows us to represent the kernel as a (convergent) sum of eigenfunctions of the associated integral operator (cf. Equation 2.36 and Equation 2.37); it is then a 'well-behaved' kernel function, and we can identify a feature map accordingly. We can show that, for the Path kernel, Mercer's theorem holds.

Consider the NTK at a fixed value of its parameters w , $\bar{\kappa}_{\text{ntk}}(x, x') = \kappa_{\text{ntk}}(x, x'; w)$. Such a mapping is Mercer, as it is the kernel induced from the feature map $x \mapsto \nabla_w f(x, w)$. Now, considering the parameter path γ during its training, each infinitesimal contribution $\gamma(t)$ defines a valid NTK. The infinitesimal limit of the combination of positive semi-definite Mercer kernels, as well as finite sums, is still Mercer and, therefore, immediately valid for the Path kernel. The discrete approximation of the Path kernel, in which the integral is replaced with a sum, is clearly Mercer too.

4.3.4 A mathematical caveat in the proof of [90, Theorem 1]

The literature acknowledges a mathematical caveat in [90, Theorem 1], which makes the demonstration non-rigorous. Although not entirely resolved, [91] has taken steps toward addressing this issue by imposing stronger assumptions on the training process. We comment on this caveat. Firstly, the dynamic of any machine learning model trained via stochastic gradient descent takes the form of a first-order non-homogeneous differential equation. Let $h(\mathbf{x}; \mathbf{w}) : \mathbb{R}^D \times \mathbb{R}^P$ be the predictor at hand and let L be the convex, differentiable loss. Then, the dynamic of the training is expressed as:

$$\frac{dh(\mathbf{x}; \mathbf{w})}{dt} = - \sum_{j=1}^P \frac{\partial h}{\partial w_j} \cdot \frac{\partial L}{\partial w_j}. \quad (4.14)$$

We can insert the NTK in this equation, leading to

$$\frac{dh(\mathbf{x}; \mathbf{w}(t))}{dt} = \sum_{j=1}^d \frac{\partial h(\mathbf{x}; \mathbf{w})}{\partial w_j} \cdot \frac{dw_j}{dt} \quad (4.15)$$

$$= \sum_{j=1}^d \frac{\partial h(\mathbf{x}; \mathbf{w})}{\partial w_j} \cdot \left(- \frac{\partial L(\mathbf{w}(t))}{\partial w_j} \right) \quad (4.16)$$

$$= \sum_{j=1}^d \frac{\partial h(\mathbf{x}; \mathbf{w})}{\partial w_j} \cdot \left(- \sum_{i=1}^m \frac{\partial \ell(y_i^*, h(\mathbf{x}_i; \mathbf{w}))}{\partial w_j} \right) \quad (4.17)$$

$$= \sum_{j=1}^d \frac{\partial h(\mathbf{x}; \mathbf{w})}{\partial w_j} \cdot \left(- \sum_{i=1}^m \frac{\partial \ell(y_i^*, y_i)}{\partial y_i} \frac{\partial h(\mathbf{x}_i; \mathbf{w})}{\partial w_j} \right) \quad (4.18)$$

$$= - \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \sum_{j=1}^d \frac{\partial h(\mathbf{x}; \mathbf{w})}{\partial w_j} \frac{\partial h(\mathbf{x}_i; \mathbf{w})}{\partial w_j} \quad (4.19)$$

$$= - \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \nabla_{\mathbf{w}} h(\mathbf{x}; \mathbf{w}) \cdot \nabla_{\mathbf{w}} h(\mathbf{x}_i; \mathbf{w}) \quad (4.20)$$

$$= - \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \kappa_{\text{ntk}}(\mathbf{x}, \mathbf{x}_i; \mathbf{w}) \quad (4.21)$$

For $\epsilon \rightarrow 0$ we have:

$$h(\mathbf{x}) = h(\mathbf{x}; \gamma(1)) = h(\mathbf{x}; \gamma(0)) - \int_0^1 \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \kappa_{\text{ntk}}(\mathbf{x}, \mathbf{x}_i; \gamma(t)) dt. \quad (4.22)$$

Representing such a function straightforwardly as a linear model is not possible. In recovering such a form, the author performs multiplication and division by the Path Kernel itself. This leads to a prediction that looks similar to a linear model, although it has an undesired dependence of $\mathbf{x}$ in the model parameters:

$$h(\mathbf{x}; \gamma(1)) = h(\mathbf{x}; \gamma(0)) + \sum_{i=1}^m \left(-\frac{\int_0^1 \frac{\partial \ell}{\partial y_i} \kappa_{\text{ntk}}(\mathbf{x}, \mathbf{x}_i; \gamma(t)) dt}{\kappa_{\text{pk}}(\mathbf{x}, \mathbf{x}_i; \gamma)} \right) \kappa_{\text{pk}}(\mathbf{x}, \mathbf{x}_i; \gamma) \quad (4.23)$$

$$= h(\mathbf{x}; \gamma(0)) + \sum_{i=1}^m \alpha_i(\mathbf{x}) \kappa_{\text{pk}}(\mathbf{x}, \mathbf{x}_i; \gamma). \quad (4.24)$$

4.4 Non-linearity in quantum neural networks

A crucial aspect of variational quantum classifiers involves the composition of multiple unitary transformations and their collapse into a single unitary. This aspect is derived from Schrödinger's equation being a linear partial differential equation. Here, we assess how this aspect influences this model's feature learning and hierarchical feature learning capabilities.

Initially, one might assume that a QNN behaves as a linear model, as per Equation 4.1. This assumption may also arise from the analogy with neural networks: when employing a linear activation function, each layer collapses into a single one, making the overall model linear. However, such an analogy is imperfect and can lead to misconceptions regarding the capabilities of QNNs.

- In contrast to linearly activated neural networks, all QNNs can exhibit feature learning, albeit with minimal intensity. This phenomenon arises due to the non-linearity of measurement. Such a property can be confirmed by showing the non-constant behavior of the NTK during training.
- In analogy with linearly activated neural networks, QNNs do not possess a hierarchical feature learning structure. The collapse of its layers implies that we cannot identify at which stage of the quantum system's evolution we have a new set of features (it could be any of them).

Finally, Schuld [92] have proven that the QNN is linear *in the Hilbert space of the quantum system*. When this Hilbert space is considered the reference, the model can be considered linear. It is essential to note that this space is distinct from the parameter

space in which optimization is performed; thus, from an optimization perspective, the QNN is definitively a non-linear model, as indicated by Equation 4.3. Throughout the rest of this section, we briefly compare the model for a 'quantum' neural network proposed in the literature, which is not restricted to variational quantum classifiers. Then briefly present the results related to the quantum NTK.

The authors' arguments are based on the non-constancy of the NTK throughout the training. However, we can show a simple example of a QNN to be nonlinear in its parameter space by explicit construction. Consider any model in the form:

$$f(x; \theta) = \text{Tr}[V^\dagger(\theta)\rho_x V(\theta)O]. \quad (4.25)$$

Here, $\rho_x = U_\phi^\dagger(x) |0\rangle\langle 0| U_\phi(x)$ is the output of the feature map over an input value x , and O is a Hermitian observable. This function is linear only in the space of density matrices of the quantum system $\mathcal{H}$, as imposed by Schrödinger. In terms of the space of parameter, the quantum model does not fit the form set out in Equation 4.1. Consider, for instance, a single-qubit quantum model that operates on a single input $x \in \mathbb{R}^1$ and depends on a single parameter $\theta \in \mathbb{R}^1$:

$$f(x; \theta) = \text{Tr} \left[\begin{pmatrix} \cos^2(\theta + x) & -i \sin(\theta + x) \cos(\theta + x) \\ \frac{1}{2}i \sin(2(\theta + x)) & \sin^2(\theta + x) \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \right] = \cos(2(\theta + x)). \quad (4.26)$$

With respect to θ , the function is clearly not linear. This implies that the model can exhibit feature learning.

4.5 Quantum Path Kernel

This section aims to recover a hierarchical feature learning structure in quantum neural networks using the Path kernel. Referring back to the aspects introduced in Section 4.3, we might speculate whether the hierarchical dependencies between parameters that occur spatially (typically found in deep neural networks) could also exist temporally. In other words, the dependencies among the parameters during learning, which arise from the chain rule, might similarly apply to the dependencies between parameters at different time steps, specifically from $\theta(t)$ to $\theta(t + 1)$.

The Quantum Path Kernel (QPK) is defined by explicitly substituting the quantum

neural network into the definition of the Path Kernel, as follows:

$$\begin{aligned} \kappa_{\text{qpk}}(\mathbf{x}, \mathbf{x}'; \gamma) &= \int_{\gamma} \nabla_{\boldsymbol{\theta}} \langle 0 | V^{\dagger}(\mathbf{x}) U^{\dagger}(\boldsymbol{\theta}) O U(\boldsymbol{\theta}) V(\mathbf{x}) | 0 \rangle^T \\ &\quad \cdot \nabla_{\boldsymbol{\theta}} \langle 0 | V^{\dagger}(\mathbf{x}') U^{\dagger}(\boldsymbol{\theta}) O U(\boldsymbol{\theta}) V(\mathbf{x}') | 0 \rangle d\boldsymbol{\theta} \end{aligned} \quad (4.27)$$

$$\begin{aligned} &\approx \frac{1}{T} \sum_{i=0}^{T-1} \nabla_{\boldsymbol{\theta}} \langle 0 | V^{\dagger}(\mathbf{x}) U^{\dagger}(\boldsymbol{\theta}_i) O U(\boldsymbol{\theta}_i) V(\mathbf{x}) | 0 \rangle^T \\ &\quad \cdot \nabla_{\boldsymbol{\theta}} \langle 0 | V^{\dagger}(\mathbf{x}') U^{\dagger}(\boldsymbol{\theta}_i) O U(\boldsymbol{\theta}_i) V(\mathbf{x}') | 0 \rangle \end{aligned} \quad (4.28)$$

where T is the number of points sampled from the curve γ , and $t = i/T \in [0, 1]$. The equation representing the corresponding kernel machine is given by:

$$f(\cdot) = \sum_{i=1}^n \alpha_i \kappa_{\text{qpk}}(\mathbf{x}_i, \cdot), \quad (4.29)$$

where the coefficients α_i are determined during the training process and $\mathbf{x}_i$ are the elements of the training set. Note that, in contrast to the model depicted in Theorem 1 of Domingos [90], the coefficients α in Equation 4.29 do not depend on $\mathbf{x}$, rendering such a model a proper kernel machine.

Figure 4.3 illustrates how to implement a Path kernel. Supposing γ is a morphologically rich, non-straight line, then in most of the points within these line the parameters $\boldsymbol{\theta} = \gamma(t)$ depict a valid parameter value which induces a valid NTK $\kappa_{\text{ntk}}(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta})$. Each $\boldsymbol{\theta}$ is correlated to the value appearing previously in the training, $\boldsymbol{\theta}' = \gamma(t-1)$ and depends on it via the stochastic gradient descent rule. If we 'stack' all the contribution together, we obtain a hierarchy of kernels depending each one on the previous ones. The single contributions can be put in analogy to the layer of a deep neural network, as the contributions hold a different (despite correlated) representation and each is source of representation learning.

The morphological evolution γ is crucial to understand whenever the Path kernel may be of any use for the given task. In fact, if γ is a straight line, i.e., in the form $\{(1-t)\boldsymbol{\theta}_0 + t\boldsymbol{\theta}_f \mid t \in [0, 1]\}$ (where $\boldsymbol{\theta}_0 \in \mathbb{R}^P$ and $\boldsymbol{\theta}_f \in \mathbb{R}^P$ represent the parameters at initialization and convergence to the local minima of $\mathcal{L}$), the derivative of the linear model $\nabla_{\boldsymbol{\theta}} f$ is independent of $\boldsymbol{\theta}$. In this case, the NTK is constant, and the Path kernel collapses into the NTK. However, non-linear models usually have a complex evolution

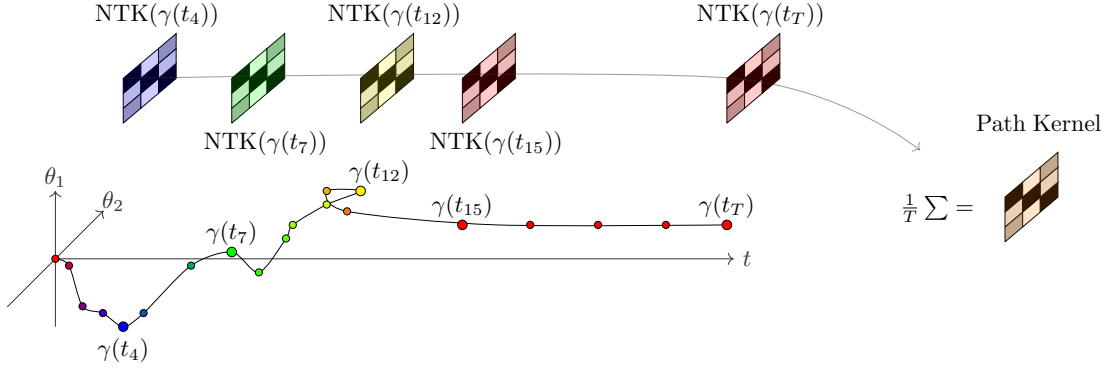


Figure 4.3: Computation of the Path Kernel. *Bottom left*: A typical parameter trajectory γ is depicted, representing parametric evolution during the training phase. *Top left*: As θ evolves, it gives rise to differing NTK matrices corresponding to distinct data representations. Such a sequence of matrices thus gives rise to a hierarchical stack of representations in the feature learning regime. *Middle*: As the training approaches convergence, subsequent matrices become similar, and thus their corresponding representations are correlated. *Right*: The Path Kernel constitutes the average over these representations.

due to the non-convexity of the optimization landscape. Therefore, γ is not constrained to a linear trajectory in general. In this latter case, $\nabla_{\theta} f$ and the NTK will vary over time during learning.

An important factor to consider in the morphological evolution of γ and its impact on the hierarchy of representation is the potential existence of biases. If the optimization process converges rapidly, the contributions will be oversampled towards values that are close to the NTK at the end of the process. To achieve this, the contributions must be uniformly sampled over γ , rather than at fixed times $\gamma(0), \dots, \gamma(T)$, as the difference in norm between consecutive values is expected to decrease due to convergence. Moreover, the process is greatly influenced by the particular setup of the optimization, such as the learning rate and the algorithm employed.

In order to address the issue of excessive bias in consecutive representation, we can effectively depend on modifying the optimization process. Some optimizers have the ability to automatically adjust the learning rate in locally convex regions of the loss function, resulting in a diverse set of parameter values $\gamma(t)$. This leads to fewer contributions in the path kernel and reduces the similarity between these contributions. Another approach could involve perturbing parameters through stochastic or noisy optimization. Additionally, non-gradient-descent-based optimization methods, such as genetic algorithms, could be employed, even though they may lose all connection to the

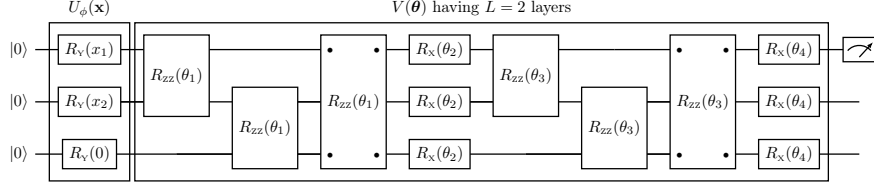


Figure 4.4: Instance of variational quantum classifier ($d = 3$ qubits and $L = 2$ layers) contributing to the definition of the machine learning model leading to the QPK machine.

original definition and motivation of the Path kernel (the hierarchy induced by gradient descent). The expectation is that causally-dependent but decorrelated contributions to the QPK will still result in a hierarchical representation, while also being diverse enough to avoid biasing the model. In addition, adopting an alternative viewpoint, other machine learning methods, such as ensemble learning (e.g. bagging), leverage the decorrelation of features to achieve superior performance.

4.5.1 Experimental evaluation

We want to investigate whether the use of the quantum path kernel brings the architecture and the performance of quantum neural networks closer to the ones of deep learning models due to their natural hierarchy of representation. To this end, our evaluation focuses on the Gaussian XOR Mixture classification problem [93, 94, 95], which has been shown by Refinetti et al. [87] to be an important benchmark to highlight the need of nonlinear models.

The steps of our experimental workflow are depicted in Figure 4.5. Initially, we create the dataset for the specified problem. Subsequently, we proceed to train multiple Quantum Neural Networks (QNNs) in order to achieve the most optimal fit with the generated data. Next, we use the training data to create the QNTK and QPK matrices. The QPK matrix is then employed to train a kernel machine for the final classifications. The analysis continues by investigating the convergence of QNNs as the number of layers grows, emphasizing the influence of architectural parameters on QNNs. Ultimately, we evaluate and compare the accuracy of the QNTK and QPK methodologies in both testing (to capture the actual generalization performance) and training phases (to evaluate improvement in the interpolation power).

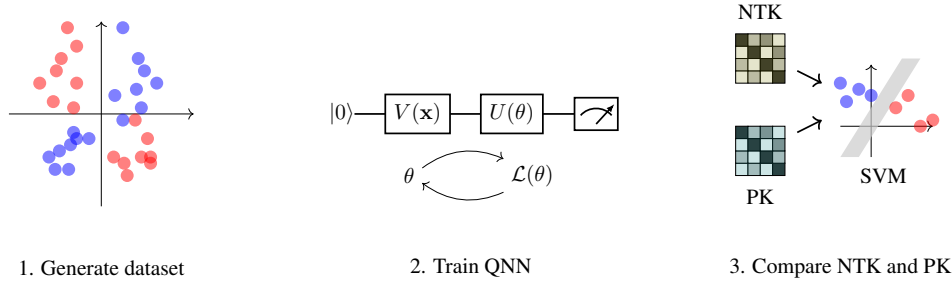


Figure 4.5: Experimental workflow overview. A random labeled dataset is generated for the Gaussian XOR Mixture classification problem. Subsequently, a variational quantum classifier is trained to compute both the Quantum Path kernel and the baseline kernel (QNTK). Finally, the performance of the Quantum Path kernel is compared with that of the QNTK.

Choice of the problem

The Gaussian XOR Mixture classification problem is a natural and paramount test for layer-wise learning models. This is intuitively understood from the 2-dimensional example: we need a first layer learning that the multiplication of the two coordinated leads to a new, higher-order feature identifying the quadrant to which the point belongs, either first-or-third or second-or-fourth. Then, the problem is linearly separable by simply looking at the sign of the first-order feature. According to theoretical analyses by Refinetti et al. [87], random features-based kernel methods struggle to accurately and efficiently classify this dataset.

Dataset generation

The ground truth refers to the objective and accurate information or data that represents the actual state of affairs or reality. The Gaussian XOR Mixture dataset is defined by multiple parameters. The variables in the equation are as follows: d represents the dimensionality of the features, d' denotes the number of non-zero features representing the multidimensional Gaussian XOR Mixture (where $d' \leq d$), $\bar{\epsilon}$ represents the variance of the Gaussian noise, and n represents the number of data points. The composition is structured in the following way:

$$\mathcal{D}_{d,d',\bar{\epsilon},n} = \left\{ ([x_1 + \epsilon_1, \dots, x_{d'} + \epsilon_{d'}, 0, \dots, 0]^T, y_i) \in \mathbb{R}^d \times \{\pm 1\} \right\}_{j=1}^n \quad (4.30)$$

Here, the variables x_i are independently and identically distributed as either +1 or

–1. The variables ϵ_i are independently and identically distributed according to a normal distribution with mean 0 and variance $\bar{\epsilon}$, for $i = 1, \dots, d'$. Finally, the variable y_i is defined as the product of all x_i for $i = 1$ to d' . The dataset is best classified using the oracle function:

$$f_{\text{oracle}}(\mathbf{x}) = \prod_{i=1}^{d'} x_i \quad (4.31)$$

Several datasets, denoted as $\mathcal{D}_{d,d',\epsilon,n}$, are created. The dimensionality of the features, d , varies from 2 to 10. The noise, ϵ , ranges from 0.1 to 1.0. The number of non-zero features, d' , is fixed at 2, and the number of elements, n , is fixed at 32. The dataset is divided into two sets: a training set, denoted as $\mathcal{D}_{\text{train}}$, and a testing set, denoted as $\mathcal{D}_{\text{test}}$. The partitioning is done randomly.

Choice of the model

We generate a uniform family of parameterized quantum circuits in the form:

$$V(\boldsymbol{\theta}) = \prod_{j=1}^L \exp\left\{(-i \theta_{2i+1} \sigma_x^{(j)})\right\} \exp\left\{(-i \theta_{2i} \sigma_z^{(j)} \otimes \sigma_z^{(j+1 \bmod d)})\right\} \quad (4.32)$$

Here, the variable L represents the hyperparameter that determines the number of layers in the model. Selecting the appropriate trainable ansatz is essential in order to prevent the occurrence of barren plateau problems. From a set of unitaries that are not maximally expressive, a particular form is chosen, which has been shown to have favorable trainability based on empirical evidence.

This characterizes the following quantum model:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \text{Tr}[\rho_{\mathbf{x}, \boldsymbol{\theta}} O] = \text{Tr}[V^\dagger(\boldsymbol{\theta}) U_\phi^\dagger(\mathbf{x}) \rho_0 U_\phi(\mathbf{x}) V(\boldsymbol{\theta}) O] \quad (4.33)$$

where data encoding, U_ϕ , is chosen for its simplicity, resulting in a constant-depth circuit,

$$U_\phi(\mathbf{x}) = \prod_{j=1}^d \exp\left\{(-i x_j \sigma_y^{(i)})\right\} \quad (4.34)$$

and the observable, O , is

$$O = \sigma_z^{(0)}. \quad (4.35)$$

The choice of the observable is driven by the need to prevent the barren plateau. Based

on prior research, it is expected that global observables will have gradients with variance $\propto 1/2^n$.

We create an instance of a QNN with a variable number of layers, ranging from $L = 1$ to 20. The value of n is determined by the dimensionality of the datasets' points. The QNNs are initialized at $\theta = \mathbf{0}$ in order to prevent problems with trainability.

Training process

Each QNN is trained using the ADAM algorithm for stochastic gradient descent, which is executed for a total of 1000 epochs. The initial learning rate for adaptation is set to 0.1. The loss function used can be either Binary Cross-Entropy (BCE) or Mean Squared Error (MSE). These loss functions are commonly used for classification and regression tasks, with BCE being the natural choice and MSE tested for the sake of curiosity. The batch size is adjusted to match the exact number of samples in the training set. We examine the influence of different initialization parameters on the convergence of the loss function during training, taking into account both the number of epochs and the depth of the model.

4.5.2 Results

We evaluate two aspects. Firstly, we examine the depthwise convergence properties of the respective $f(x; \theta)$ models, specifically QPK and QNTK at their initialization stage. It is crucial to consider that if models converge too quickly (in a lazy training regime), their behavior becomes almost linear and the QPK converges to the QNTK. This implies that there would be no advantage in using the kernel model. In classical machine learning, lazy training refers to the phenomenon where very wide neural networks exhibit an exponential decrease in loss, while the network parameters remain close to their initial values throughout the training process. Secondly, we assess the training loss in relation to the training epochs and for a progressively larger number of ansatz "layers". The set L consists of the integers from 1 to 20, inclusive. This feature enables us to ascertain whether there is an enhancement in performance when utilizing the QPK. Nevertheless, it is crucial to verify the impact of adding a greater number of layers. In addition, we graph the discrepancy in magnitude between the parameters at the end of training and their initial values. These will be utilized to assess the degree to which parameters deviate from their initial values, with a larger norm indicating a more comprehensive and active training process.

Thirdly, we assess the resilience of the classifiers in relation to random fluctuations during training. This will help us determine whether the noise in the dataset will result in overfitting, where the model incorporates the noise excessively, or if we will instead encounter "benign overparameterization" [96], where the noise is incorporated but the predictor remains simple and performs well in general. The measurement of such an aspect is based on the performance of generalization.

Ultimately, we assess the overall ability of our approach to generalize by comparing it to the performance of the QNTK after the training period. To accomplish this, we assess the test accuracy score of the QPK and QNTK in comparison to the oracle. The QPK's superior performance in solving the Gaussian XOR Mixture problem can be attributed to its hierarchical representations, which is the only distinguishing characteristic between the QPK and the QNTK: note that the NTK has a fixed parameter value at the end of the training process, unlike the work of Jacot, Gabriel, and Hongler [27] where the parameter value is fixed at the beginning.

Depthwise convergence characteristics

In Figure 4.6, the convergence behavior of evaluated quantum machine learning models increases with more layers. The signal-to-noise ratio decreases row by row in Column 1, which shows training distribution examples. Column 2 displays training loss curves. Column 3 shows the difference in parameter vector size from its initial value: $\frac{\|\theta(n) - \theta(0)\|}{\|\theta(0)\|}$. In this context, $\theta(0)$ represents the initial parameter value, while $\theta(n)$ represents their value at the n -th epoch.

According to [82], no model meets the interpolation threshold, which is the point where training data is perfectly fitted with zero training error. At least 32 parameters are needed to fit the training dataset. This yields 16 points with 2 non-zero coordinates each. Even with 40 parameters in the most complex setup, we cannot meet the interpolation threshold. The deliberate selection of a parametrically constrained U as a regularization predicts this behavior. As with classical deep neural networks (DNNs), increasing the number of parameters leads to reduced loss (see Figures 4.6b-4.6e-4.6h). As shown in Figures 4.6c-4.6f-4.6i, the proximity between parameter vectors and initialization increases.

No QNN model shows lazy training. In particular, models with more parameters converge faster but have much greater parameter variation. Smaller models show significant norm difference fluctuation before convergence. Significant differences in

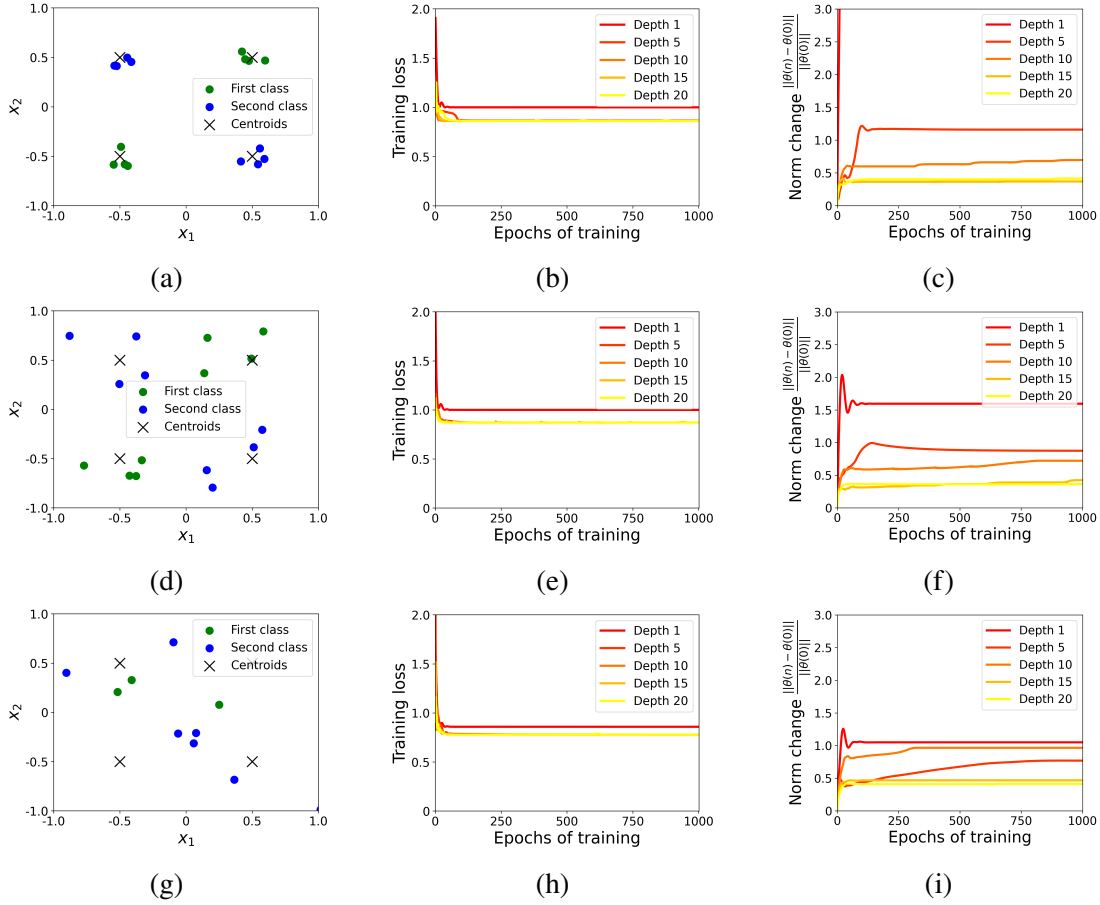


Figure 4.6: Behavior of quantum machine learning models $f(\mathbf{x}; \boldsymbol{\theta})$ during the training phase. Figure (4.6a) displays the training dataset for the parameters $d = 4, \epsilon = 0.1$. The training loss progression for each of the 20 depthwise models ($L = 1, \dots, 20$) is illustrated in (4.6b). Figure (4.6c) quantifies how much the parameter vector deviates from its initial state. For $d = 4$ and $\epsilon = 0.4$, the corresponding data is shown in (4.6d)-(4.6e)-(4.6f). Similarly, for $d = 4$ and $\epsilon = 1.0$, the relevant information is provided in (4.6g)-(4.6h)-(4.6i).

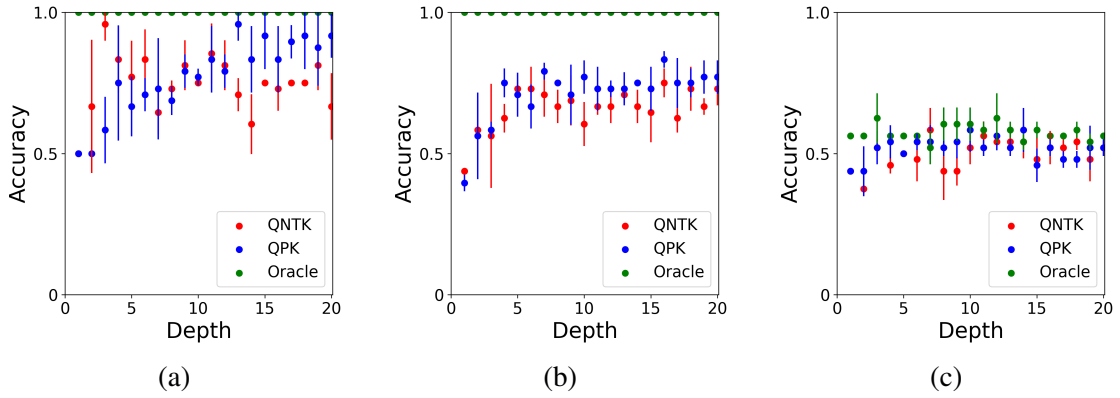


Figure 4.7: Test accuracy scores for the Quantum Path Kernel model, the Quantum NTK, and the oracle.. Error bars indicate the standard deviation across three identical experiments with the following parameters: $d = 4, \epsilon = 0.1$ (Figure 4.7a); $d = 4, \epsilon = 0.4$ (Figure 4.7b); $d = 4, \epsilon = 1.0$ (Figure 4.7c).

training characteristics between the QPK and QNTK suggest a complicated training process.

Test and train accuracy of the Quantum Path Kernel versus the Quantum NTK

In Figure 4.7, test accuracy results demonstrate the models' ability to generalize to new data. The QPK outperforms the Quantum NTK as hierarchical depth increases under the highest signal-to-noise ratios, but both models perform similarly at lower ratios. Figure 4.8 shows the training accuracy progression with depth upon convergence. The QPK consistently outperforms the Quantum NTK across the signal-to-noise range, with this difference increasing at higher noise levels (leading to overfitting relative to the noise-free oracle in panel *c*). The QPK's lower bias compared to the Quantum NTK is expected. These results confirm the QPK's expected performance advantage over the Quantum NTK in the Gaussian XOR mixture.

4.5.3 Further directions

We developed the Quantum Path Kernel (QPK) to integrate complex learning behaviors from classical multi-layer networks, such as hierarchical feature learning, into quantum neural networks through training kernelization. Our benchmark for multilayer learning is the Gaussian XOR mixture classification problem, which requires at least a two-layer solution to approximate the Bayes optimal solution. In high-depth, low signal-to-noise

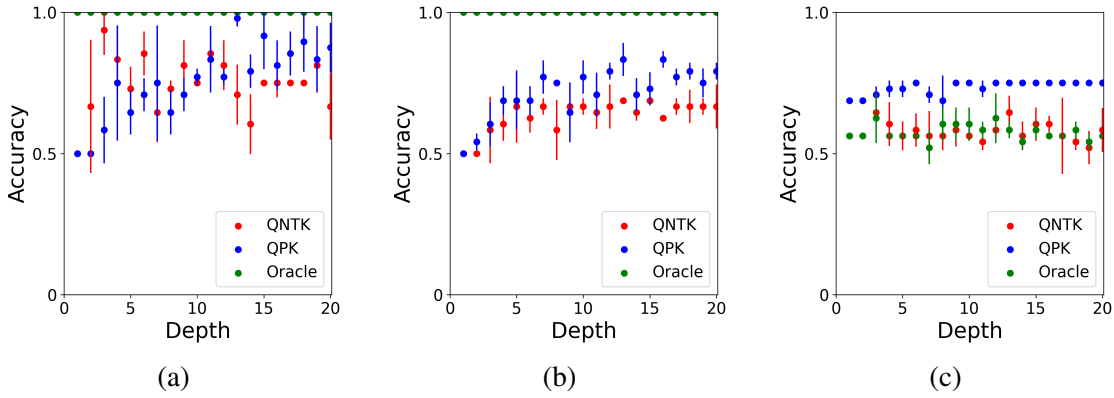


Figure 4.8: Training accuracies of the Quantum Path Kernel model, the Quantum NTK, and the oracle. Error bars represent the standard deviation across three identical experiments with specifications $d = 4, \epsilon = 0.1$ (Figure 4.8a); $d = 4, \epsilon = 0.4$ (Figure 4.8b); $d = 4, \epsilon = 1.0$ (Figure 4.8c).

scenarios, our experiments outperform the Quantum Neural Tangent Kernel (NTK) in generalization.

Our theory shows that the QPK converges to the Quantum NTK only in the lazy training regime, where training loss rapidly decreases to zero and model parameters remain close to their initial values. This is typical of infinitely wide neural networks, which behave like linear models. However, our experiments suggest that quantum neural networks (QNNs) operate differently and are not linear.

We discuss using stochastic, noisy, or non-gradient descent-based optimization techniques to artificially perturb QPK parameter paths to promote more decorrelated feature representations, but this study did not test this. We also suggest heuristic weighting of kernel representations using Multiple Kernel Learning to extend the QPK approach. We also briefly mention the QPK’s ensemble interpretation due to the averaging operation over its kernel matrices.

4.6 From over-parameterization to ensembles in NISQ machine learning models

Over-parameterization has been denoted as one of the characteristics leading to the success of deep models: despite their complexity, having many more parameters to interpolate the data perfectly via a large set of solutions will simplify the optimization and reach minimum-normal solutions. Over-parameterization itself is not directly linked to

deep models, as it may happen in kernel machines, too (consider, for example, a random Fourier feature kernel with an arbitrary number of features), although it is particularly relevant in deep models as it contributes to their effective training.

In a quantum machine learning context, however, it is still difficult to think we will be able, in practice, to have such large models, at least in the NISQ regime. Therefore, an over-parameterization of quantum neural networks or quantum kernels may be regarded as unfeasible². However, we might obtain the same performances of over-parameterized models in different ways.

One idea has been proposed by Geiger et al. [24]. The authors have studied the relationship between double descent, some types of regularization, and ensembling techniques in neural networks. Their work, displayed in Figure 4.9, shows that while the average model performs the worst at the interpolation threshold, the regularization can effectively suppress the peak. Most importantly, as it has motivated our work, an ensemble of few models at the interpolation threshold (20 models with 825 parameters, in their experiment) outperforms the biggest model tested (10^7 parameters).

Such a study motivates an investigation into the ensemble learning of QNNs. The goals of the investigation are to address the following questions:

1. *Can an ensemble of a few small QNNs exhibit superior performance compared to a single larger quantum neural network in a noiseless setting?* This question is motivated by the success of overparameterization [98] and its connection to ensemble techniques depicted in Figure 4.9.
2. *Can ensemble learning be used to save resources (qubits, parameters, gates, etc.) in a manner similar to divide-and-conquer techniques (e.g., quantum circuit splitting [99])?*
3. *Can ensemble learning be employed as a high-level error mitigation technique?* This is motivated by Zhang et al. [100], which demonstrates that ensembles can enhance the performance of several noisy machine-learning tasks. In our NISQ environment, such an improvement may be expected due to the shorter circuits needed.

²Note that Larocca et al. [97] has studied a notion of overparameterization for quantum neural networks. Such a study focuses on navigating the lost landscape freely, i.e., without getting stuck in saddle points that look like minima just because we do not have enough degrees of freedom to escape them. This notion does not directly map to the classical machine learning idea of overparameterization, i.e., reaching and surpassing the interpolation threshold.

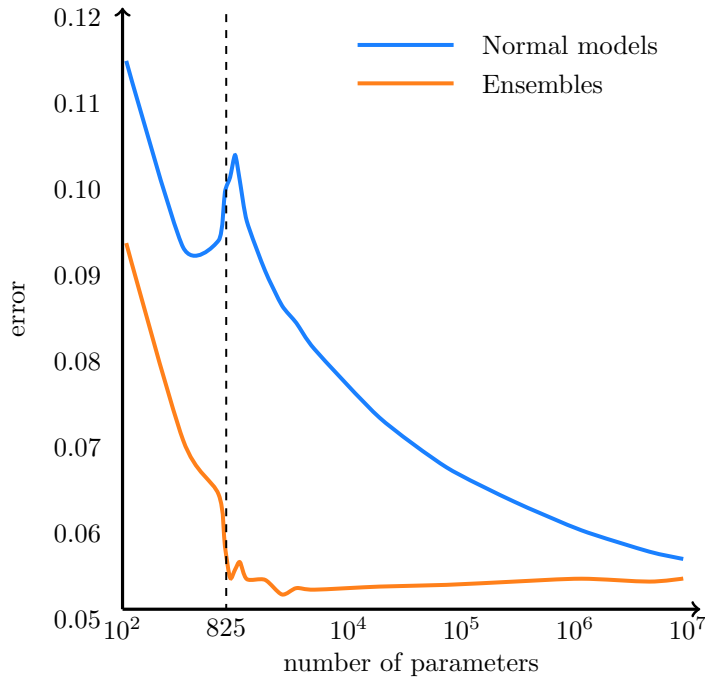


Figure 4.9: Comparison of testing error as a function of the number of model parameters between a single deep neural network applied to MNIST classification and an ensemble of few (20) deep neural networks trained on different datasets. As expected, the single neural networks exhibit double descent behavior, with their worst performance occurring at the interpolation threshold. Acting as a form of regularization itself, the ensemble can mitigate peaks in loss at the interpolation threshold, leading to a monotonically decreasing error with respect to the number of parameters. Remarkably, the ensemble achieves performance comparable to the largest DNN at the interpolation threshold. This is a schematized and simplified adaptation of the results shown in [24, Figure 2].

Data selection strategy			
		Subset of features	
		No	Yes
Subset of samples	No	/	Random subspace
	Yes	Bootstrapping, pasting	Random patch
Composition + training of single model instances			
		Model instances	
		Heterogeneous	Homogeneous
Type of processing	Sequential	/	Boosting
	Parallel	Stacking	Bagging
Combination rule of the outputs			
		Discrete	Continuous
		Majority voting	Average
		Weighted majority voting	Weighted average
		Borda counts	Min Max

Figure 4.10: Taxonomy of ensemble learning models.

4.6.1 Background on ensemble learning

Ensemble systems aim to reduce either model bias or deviation to increase generalization. Such a goal is achieved by training multiple models and combining their outputs using a specific rule. A comprehensive introduction to ensemble techniques is out of the scope; for that, the reader can refer to Zhang and Ma [101]. We briefly enumerate the main points characterizing this family of approaches.

The foundational aspects of ensemble learning lie in Condorcet’s jury theorem [102]. Let a jury of k individuals, each having a probability to correctly judge $p = \frac{1}{2} + \epsilon$ with $0 < \epsilon \ll 1$. Then, the probability that the collegial (majority voting based) decision is correct is $p_{\text{jury}} = \sum_{k=\lceil m/2 \rceil+1}^m \binom{m}{k} p^k (1-p)^{m-k}$. Translating this concept to machine learning, it is immediate to see that combining *weak* learners $h_1, \dots, h_m$ can create a strong learner h_{ens} with significantly improved performance. Any ensemble learning model is characterized by three phases: data selection strategy, model composition, training methods, and output combining rule. Figure 4.10 schematizes these components.

The *data selection* method affects how data is allocated to each model. We need to ensure that the individual instances contributing to the ensemble produce different, uncorrelated outputs. To that end, the *bootstrapping* data selection method can be used, providing each instance with a different subset of the samples, ensuring that the datasets do not overlap. When we forbid overlapping between subsets, bootstrapping is referred

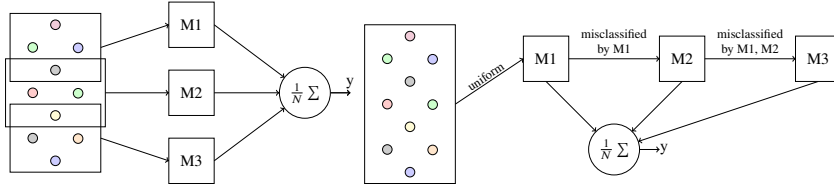


Figure 4.11: Comparison of bagging (left) and 'standard' boosting (right) techniques. In bagging ensembles, models are trained in parallel on dataset subsets drawn uniformly, with predictions combined using an averaging function. In contrast, boosting ensembles train models sequentially: the first predictor draws samples uniformly, while subsequent models draw samples from a probability distribution biased towards previously misclassified items.

to as *pasting*. In these cases, the models are trained using all the features. However, even this aspect can be considered: when we use a single dataset for all instances but perform a vertical slicing of the features, i.e., different subsets of features are given to different instances, the data selection method is known as *random subspace*. The combination of bootstrapping and random subspace is known as the *random patch* technique.

The *composition* method determines the communication, or the lack of such, between the instances. The most common method is *bagging* (i.e., bootstrap aggregation), in which many instances are trained independently over different subsets of the data, and it is usually used in combination with bootstrapping techniques. An example of this approach is the random forest (a bagging ensemble of decision trees). In contrast, *boosting* arranges the single instances in series. The initial predictor is trained on a subset of data uniformly sampled, while the subsequent models are trained over a non-uniform distribution that prioritizes the samples misclassified by the previous instances in the chain. The most popular boosting algorithm is AdaBoost [103]. These strategies are illustrated in Figure 4.11. Finally, a completely different approach is *stacking*, which trains multiple heterogeneous models on the same dataset.

The *combination rule* combines the output of the single instances. Its choice depends on the task we are solving. For classification tasks, where the labels are discrete ($y \in C = \{c_1, \dots, c_k\}$), majority voting is typically used. Alternatively, a weighted majority voting or a ranking over the classifiers (*Borda count*) can be employed to prioritize the output of certain instances considered more reliable. For regression tasks, the mean, minimum, and maximum are usually the most popular choices, depending on the specific task.

4.6.2 Experimental evaluation

We have designed a series of experiments to address the open questions presented at the beginning of this section. One aspect focuses on how to increase the performance of the quantum model; for that, we need to determine whether quantum models overfit and, therefore, could benefit from bagging techniques or if they underfit and thus could benefit from boosting techniques. Another aspect concerns the amount of resources needed to implement the quantum circuit; for that, we need to determine whether the use of data selection techniques that minimize the number of features (e.g., random patch) affects performance significantly negatively. A third aspect involves using ensembles as a high-level error mitigation technique. For that, we empirically test the performance of various ensemble configurations on noisy simulators and IBM quantum hardware and verify the potential performance improvements of the ensembles compared to single models.

Datasets

Here, we briefly introduce the dataset used to assess the performance of each approach. We have focused on real-world tasks of various natures, including synthetic and real-world datasets and regression and classification tasks.

The first dataset is a noisy linear regression task generated artificially. For this task, we can easily create as many datasets as we want due to the procedural generation method: we set a sample size n , a dimensionality d , and noise variance σ ; then, randomly sample from the uniform distribution a weight vector $\mathbf{w} \in \mathbb{R}^d$, $w_i \in [-1, 1]$; finally, the dataset is defined as the set of pairs $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$ where $\mathbf{x}^{(i)}$ is uniformly sampled from $[-1, 1]^d$, $\epsilon^{(i)}$ is sampled from a normal distribution with mean zero and variance σ , and $y^{(i)} = \mathbf{w} \cdot \mathbf{x}^{(i)} + \epsilon^{(i)}$. We have fixed $n = 250$, $d = 5$, and $\sigma = 0.1$.

The second dataset is denoted as *Concrete Compressive Strength* and is a multivariate regression task whose goal is to predict the compressive strength of concrete based on certain characteristics such as its age and its list of ingredients. It has 1030 samples, each sample being $\mathbf{x}^{(i)} \in \mathbb{R}^8$. Note that the target value, i.e., the concrete compressive strength, is a highly nonlinear function of age and the composition of the material.

The third dataset is denoted as *Diabetes* and is a multivariate regression task whose goal is predicting insulin levels from patients by relating their biological and lifestyle characteristics. The dataset has 442 instances, each sample being $\mathbf{x}^{(i)} \in \mathbb{R}^{10}$ (including age, sex, body mass index, and blood serum measurements). Here, the target variable

quantitatively assesses the progression of the disease one year after its diagnosis.

The fourth dataset is denoted as *Wine* and is a multivariate, three-class classification task whose goal is determining the geographic origin of wine samples given their chemical properties. The dataset has 178 samples, each sample being $x \in \mathbb{R}^{13}$ (representing various chemical properties, such as alcohol, malic acid, and ash content), and these include the discrete target variable.

All these datasets are open-source; all but the procedurally generated ones are well-documented in [104, 105, 106]. Their characteristics are summarized in Table 4.1. Our experimental evaluation constantly undergoes the same preprocessing steps: firstly, the data is split into 80% training and 20% testing sets; secondly, raw features are scaled to the range $[-1, 1]$ (the scaling is based on the values of the training set only).

Dataset	Source	Nature	# Features	# Samples	Task
Linear	-	Synthetic	5	250	Regression
Concrete	UCI	Real-world	8	1030	Regression
Diabetes	Scikit-Learn	Real-world	10	442	Regression
Wine	UCI	Real-world	13	178	Classification

Table 4.1: Characteristics of the datasets used to benchmark ensemble models. Note that: 'UCI' refers to the open-source *UCI Repository* available at archive.ics.uci.edu; *Scikit-Learn* is an open-source software library for Python3. The number of features excludes the target variable.

Configurations of ensemble

The taxonomy of ensemble learning models in Figure 4.10 shows that we can create an ensemble over many possible configurations whose exhaustive testing is beyond the scope of this work. Instead, we focus on specific configurations that we find interesting a priori.

We denote with $BAG - [S] - [F]$ the configuration corresponding to a bagging predictor. Such a configuration relies on bootstrapping (each base instance is trained on a subset of data drawn with replacement), which is denoted by $[S]$ and takes values in $\{0.2, 1.0\}$; thus, either each base instance is trained on 20% of the training data, or bootstrapping is disabled, and each base instance is trained on 100% of the training data. Similarly, the configuration relies on a random subspace (each base instance is trained on a subset of features per sample drawn without replacement), which is denoted by $[F]$

Model	Data Loading		Ensemble	#BP	Rule
	[F]	[S]			
BASE	-	-	-	-	-
BAG-0.3-0.2	0.3	0.2	Bagging	10	Avg
BAG-0.3-1.0	0.3	1.0	Bagging	10	Avg
BAG-0.5-0.2	0.5	0.2	Bagging	10	Avg
BAG-0.5-1.0	0.5	1.0	Bagging	10	Avg
BAG-0.8-0.2	0.8	0.2	Bagging	10	Avg
BAG-0.8-1.0	0.8	1.0	Bagging	10	Avg
ADA	(1.0)	(1.0)	AdaBoost	10	W.Avg

Table 4.2: Summary of the ensemble configuration tested.

and takes values in $\{0.3, 0.5, 0.8\}$; in this case, the random subspace is always active, and we work with at most 80% of the features of the dataset. The combination rule is the average. The ensemble is based on ten base instances.

We denote with *ADA* the configuration corresponding to the boosting predictor. The particular variant of boosting hereby considered is AdaBoost.R2 version [107] for the regression tasks and AdaBoost SAMME.R [108] for the classification problem. This configuration does not rely on bootstrapping, as it has its own protocol to draw samples from the dataset depending on the misclassifications of other base instances; neither does it use random subspace. Again, the ensemble is based on 10 base instances. We repeat each experiment 5 times.

We denote with *BASE* the baseline instance trained on the dataset without reducing either the number of samples or the number of features. The characteristics of these configurations are summarized in Table 4.2.

Choice of the quantum neural network

The model used here to construct the single instances of the ensemble are the QNNs used throughout this section, i.e., a predictor in the form $f(x; \theta) = \text{Tr}[U^\dagger(\theta)V^\dagger(x)\rho_0 V(x)U(\theta)O]$. The quantum states lie in a space of n qubits and, for these experiments, n has to match the number of features. The feature map is fixed to be $V(x) = \bigotimes_{i=1}^n R_y^{(i)}(x_i)$ for simplicity (we remark that according to Kübler, Buchholz, and Schölkopf [53], more complex feature maps may lead to poor generalization). The observable is $O = \sigma_z^{(0)}$ and operates on a single qubit to mitigate the barren plateau [38]. The variational form is structured

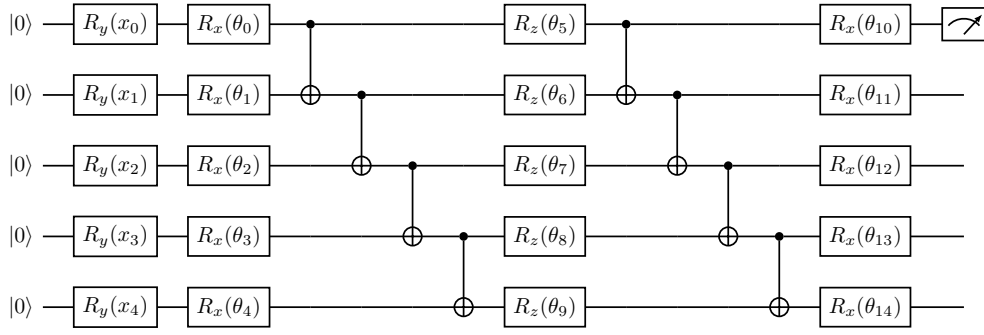


Figure 4.12: Example of the quantum circuit used for the predictor solving the linear regression dataset; in this case, the number of features is 5, corresponding to the number of qubits. The number of layers is fixed at 1. The input feature vector ($\mathbf{x}_i$) is encoded in the free parameters of the rotational gates in the feature map, while the optimization parameters (θ_i) are encoded in the ansatz.

in "layers", each having $3n$ free parameters, and defined as:

$$U_\ell(\theta) = \prod_{k=1}^{\ell} \left[\left(\bigotimes_{i=1}^n R_x^{(i)}(\theta_{3kn+2n+i}) \right) \left(\prod_{i=1}^{n-1} \text{CX}^{(i,i+1)} \right) \right. \quad (4.36)$$

$$\left. \left(\bigotimes_{i=1}^n R_z^{(i)}(\theta_{3kn+n+i}) \right) \left(\prod_{i=1}^{n-1} \text{CX}^{(i,i+1)} \right) \left(\bigotimes_{i=1}^n R_x^{(i)}(\theta_{3kn+i}) \right) \right] \quad (4.37)$$

The quantum circuit associated with such a model is depicted in Figure 4.12.

Training

The training process is based on the stochastic gradient descent algorithm ADAM, with a learning rate of 0.1. We use MSE as the loss function for regression tasks and CCE as the loss function for the classification task, the latter generalizing BCE for more than two possible labels. We compute the gradient of the predictor using the parameter-shift rule introduced by Schuld et al. [37]; the motivation here lies in our experimental evaluation over noisy simulators and IBM hardware, for which the finite difference method would introduce an error large enough to render the gradient estimation impractical.

Devices

We begin by evaluating the ensemble on a noiseless simulator, covering all the configurations introduced. The noiseless simulation is relatively inexpensive given the modest

number of qubits in the quantum systems involved, allowing us to perform extensive testing. We use PennyLane as the software framework. The noiseless simulation also assumes an infinite number of shots. Each noiseless simulation is repeated ten times, with the number of training epochs set to 150.

Next, we evaluate the ensemble on a noisy simulator. Here, the noisy simulation is much more computationally challenging, so we focus on testing only the most promising ensemble configuration, $BAG - 0.2 - 1.0$, and compare it with the baseline models. We again rely on the PennyLane software, using the IBM Qiskit framework as the backend; this choice allows us to use the noise model of a real QPU, specifically IBM Quantum Lima (a 5-qubit superconducting-based quantum computer). The number of shots is fixed at 1024, and the number of training epochs is set to 100.

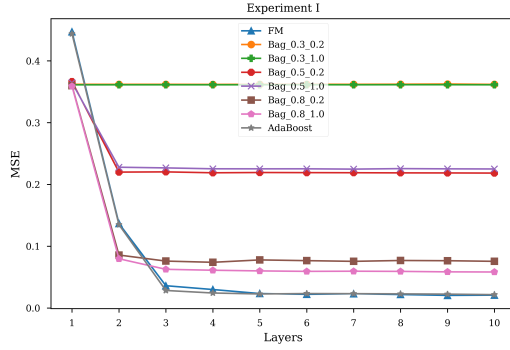
Finally, we evaluate the ensemble on the IBM Lagos QPU (a 7-qubit superconducting-based quantum computer). Again, we compare only the $BAG - 0.2 - 1.0$ configuration with the baseline models. Due to practical and economical constraints in using the QPU, we perform only 30 epochs of training.

4.6.3 Results

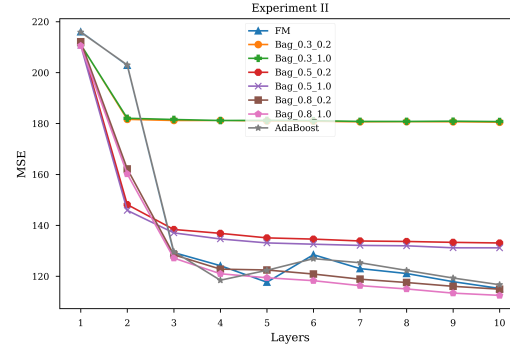
Results on noiseless simulator

We show the results of noiseless simulations in Figure 4.13.

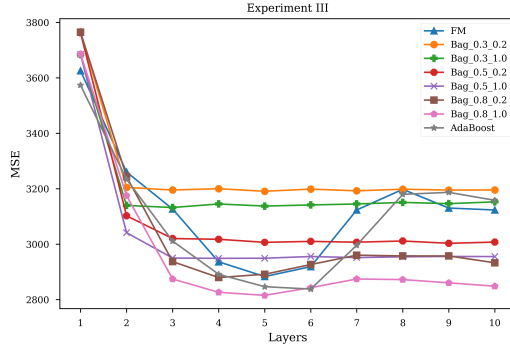
More precisely, Figure 4.13a presents the results over the linear regression datasets. Notably, the graph is partitioned into two regimes: small base instances ($\ell \leq 2$) and large base instances. In the first regime, the bagging ensemble works best; this result can be justified in terms of the limited capacity of the base instances and their inability to capture the complexity of the task. The same argument holds for boosting; there is no need to reduce the bias of a model that is underfitting, hence the similar performance between the two. In the second regime, the base instances do not suffer from underfitting anymore, allowing for better performance. The model is never underfitting, and thus, boosting is unnecessary. In contrast, the bagging suffers from the partiality of the information over each classifier due to the random subspace; in fact, the larger the cut, the larger the error. Increasing the number of layers has no substantial effect, showing a plateau. A positive note is that the bagging ensemble in the configuration $BAG - 0.8 - 0.2$ allows for a reduction in resources with a fairly reasonable decrease in performance, which may make a difference in noisy simulations.



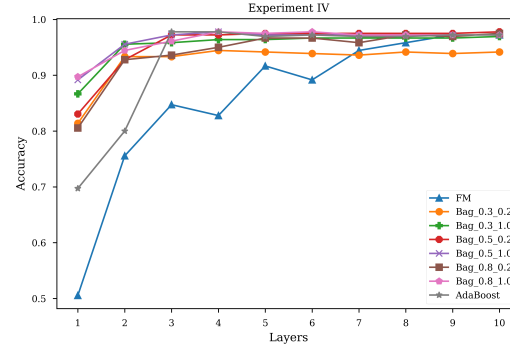
(a)



(b)



(c)



(d)

Figure 4.13: Performance of ensembles of QNN compared to the performance of their base instances on a noiseless simulator, with respect to the number of layers. (4.13a) refers to the linear regression dataset. (4.13b) refers to the concrete strength dataset. (4.13c) refers to the diabetes dataset. (4.13d) refers to the wine dataset. For regression tasks (4.13a-4.13b-4.13c) the performance is measured in terms of MSE, lower is better. For classification tasks (4.13d), the performance is measured in terms of accuracy, higher is better.

Figure 4.13b presents the results over the concrete strength regression datasets. The results propose a transition between a small and high number of layers: whenever the capacity of the model is too low, the bagging ensemble produces a much more successful classifier. In this case, for $\ell < 3$, all the bagging models, even with a random subspace cutting 80% of the features and bootstrapping cutting 80% of the samples, perform better than the base models. Again, the effect of boosting is negligible. At a higher number of layers, the performance partially resembles that of the linear regression dataset; we still experience a plateau with the bagging ensemble, however, if the random subspace is not too strong (0.8), bagging can outperform the other models.

Figure 4.13c presents the results over the diabetes regression datasets. For this dataset, we observe a division into a low-layer regime where any bagging ensemble outperforms the base models, and the considerations made for the previous datasets hold. However, for a higher number of layers, the results become more complex. Bagging with an aggressive random subspace still experiences a plateau, suffering from the inability to capture complex patterns over multiple variables. The base models, however, encounter the bias-variance tradeoff and find a sweet spot in performance at $\ell = 5$ layers. This time, bagging with a negligible random subspace applied has the best performance, and its regularization effect suppresses the peak of decrement in performance when moving away from configurations with $\ell = 5$ layers.

Figure 4.13c presents the results over the wine classification datasets. The measure of performance is based on accuracy. For this dataset, it is evident that ensemble learning is a winning strategy. We achieve up to 95% accuracy for a small number of layers compared to the 75% maximum of the base instances, even with aggressive random subspace involvement. This aspect also highlights that the task depends on patterns expressed over a small number of variables. We reach a plateau at $\ell = 3$ layers where all but the most aggressive random patch reach 97.2% accuracy. In this case, bagging helps convergence with significant resource savings, and these results are expected to be enhanced for noisy simulations as shorter circuits will be less sensitive to NISQ-related issues.

Results on noisy simulators

Now, we evaluate how ensemble techniques can positively influence performance in the presence of noise. We justify this reasoning by noting that a bagging ensemble reduces the number of features, therefore decreasing the circuit width and depth. Noisy

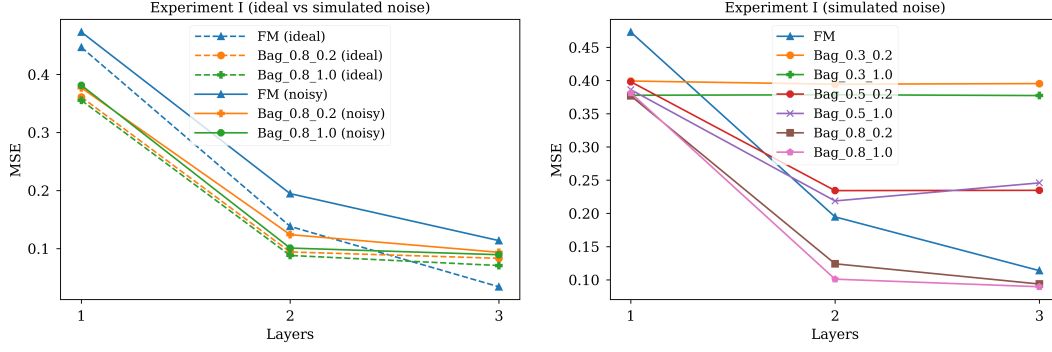


Figure 4.14: Performance of ensembles of QNN compared to the performance of their base instances on a noisy simulator (using the noise model of the QPU IBM Lima), with respect to the number of layers and over the linear regression dataset only. *Left*: MSE difference over 5 executions with all Bagging techniques. *Right*: FM and Bagging models’ performance with 80% features in ideal and noisy settings.

simulations are computationally much more expensive, so we focus exclusively on a single dataset, the synthetic linear regression one, and a limited number of layers. We also expect the bagging ensemble to reduce the length of the circuit. Hence, we focus on bagging only and avoid testing boosting techniques. In Figure 4.14 (left), we observe that in a noisy environment (solid line), the behavior is much different from the noiseless one (dotted line). In fact, the bagging ensemble performs best, whereas in noiseless simulations, the base instances showed the best performance. In Figure 4.14 (right), we show a comparison of all the bagging settings. We can see that a random subspace that is too aggressive still causes a much higher error.

Results on IBM superconducting-based QPU

We have observed the best improvement in performance in a noisy setting with the $BAG - 0.8 - [S]$ configuration on the linear regression dataset. We validate this result by testing it on the QPU IBM Lima, a 7-qubit superconducting-based hardware. We ensure statistical validity by repeating the experiment 10 times. Due to the practical and economic costs of using quantum hardware, we are limited to fixing $\ell = 1$ layer. In Figure 4.15, we show a 33% reduction in MSE and a 50% reduction in variance compared to the average base model. An important aspect is that, across the 10 base model initializations and training, we end up with substantially different models, highlighting the need to train many of them to obtain significant results. In this case, creating an ensemble from the models already trained incurs no additional overhead.

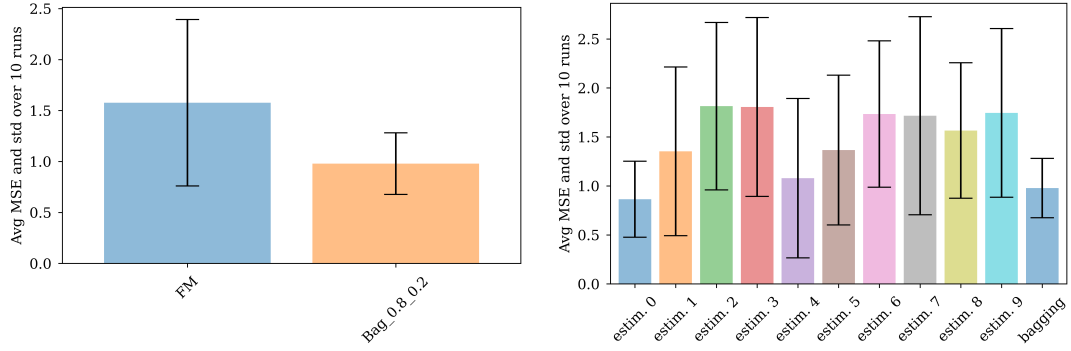


Figure 4.15: Performance of *BAG* – 0.8 – 0.2 ensemble of QNN compared to the performance of their base instances on a QPU IBM Lagos, with respect to the number of layers and the linear regression dataset only. *Left*: Performance of the bagging model relative to its estimators. *Right*: Difference in terms of MSE over 10 executions.

4.6.4 Quantification of the quantum resources for ensemble learning

Finally, we highlight the number of qubits, gates, and parameters saved using bagging ensembles, specifically due to the random subspace approach. Notably, we have shown that a mild random subspace is negligible in a noiseless environment and favorable in a noisy one. However, more aggressive random subspaces cannot compensate for the decrease in performance, even in noisy settings. Nevertheless, these techniques can potentially decrease the number of qubits by 20% to 80%, which can be crucial when the QPU is too small to fit the chosen ansatz. An overview of the resources saved for each configuration is depicted in Figure 4.16.

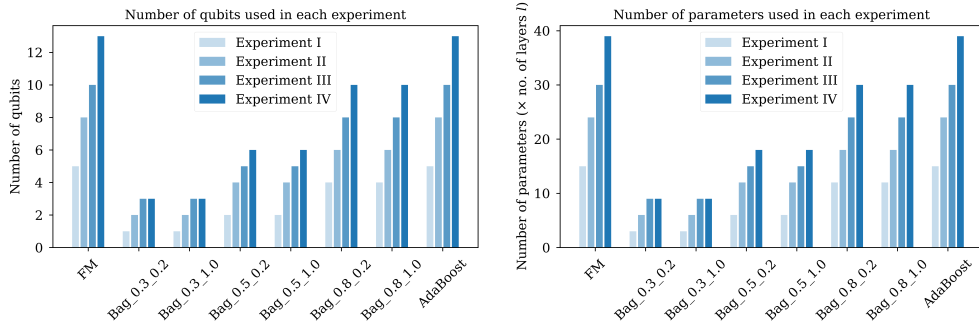


Figure 4.16: Number of qubits and parameters used by the various instances of QNN for different experiments. It is evident how the random subspace technique has been beneficial in reducing the width and length of the circuits across all configurations of 'bagging' ensemble learning tested.

This page has been intentionally left blank.

Chapter 5

Quantum kernels on graph data

Up to now, we have worked with kernels over real data, i.e., $x \in \mathcal{X} \equiv \mathbb{R}^d$. However, the kernel formalism allows us to handle more complex data types, including sets and combinatorial structures. A particularly relevant family of kernels is the *graph kernel*. This chapter explores the impact of quantum computing on graph kernels, with a specific focus on the graph edit distances (GED). GED involves finding the shortest sequence of edits to match two input structures, a combinatorially hard optimization problem that quantum computing is not expected to solve exactly in polynomial time. Nevertheless, it is worthwhile to explore its use as a heuristic.

In Section 5.1, we briefly review the role of machine learning in analyzing graphs. In Section 5.2, we concentrate on kernel-based approaches and delve into the concept of graph edit distance. In Section 5.3, we demonstrate how to heuristically calculate the graph edit distance using quantum annealing and quantum circuit-based devices by reducing the task to QUBO and evaluating the method’s efficacy. This chapter is based on [1].

5.1 Machine learning on graph data

The analysis of graphs is gaining an increasingly predominant role in the machine learning community. Many popular models working with Euclidean data has been adapted to work with graph data, leading to important applications in chemistry, biology, social science, and many other fields.

We briefly recall the definition of a graph and introduce the notation used throughout this chapter. A graph $G = (V, E)$ consists of a finite, non-empty set of vertices V

of cardinality $|V| = n$ and a set of edges $E \subseteq \binom{V}{2}$ of cardinality $|E| \leq n^2$. The graphs of interest throughout this work are *undirected*, i.e., the edges have no direction and are effectively encoded as a set (not an ordered pair), and *simple*, i.e., with no self-loops and at most one edge between two distinct vertices. The set of graphs is denoted as $\mathcal{G}$. Two graphs are isomorphic, denoted as $G_1 \simeq G_2$, if exists a bijection $\pi : V(G_1) \rightarrow V(G_2)$ such that $\pi(V(G_1)) = V(G_2)$ and $\pi(E(G_1)) = E(G_2)$. Here, we have allowed an abuse of notation regarding π , with $\pi(V) = \{\pi(v) \mid v \in V\}$ and $\pi(E) = \{(\pi(v), \pi(v')) \mid (v, v') \in E\}$.

There are different types of tasks defined over graphs, which can be categorized into three classes depending on the set of inputs they work on. These are:

- *Node-level tasks*, i.e., functions in the form $f : V(G) \rightarrow \mathbb{R}$ for a fixed G .
These types of tasks are typically used to model the need for categorizing individuals within a network. For example, they have been used to identify fraudulent activities in social and transaction networks [109] and predict citation counts in academic networks [110].
- *Edge-level tasks*, i.e., functions in the form $f : E(G) \rightarrow \mathbb{R}$ for a fixed G .
These types of tasks are typically used to model the need for categorizing interactions within a network. For example, they have been used for classifying the types of roads in transportation networks [111].
- *Graph-level tasks*, i.e., functions in the form $f : \mathcal{G} \rightarrow \mathbb{R}$.
These types of tasks are typically used to infer characteristics of small graphs or subgraphs of a larger one. For example, they have been used in predicting properties of chemical compounds [112].

The variety of approaches we can use to tackle these tasks is quite broad. Here, we review the most popular ones. The first approach is the definition of a graph (dis)similarity function that allows us to work with kernel machines; this option is explored in the next section. The second approach is the use of *graph neural networks*. The third approach is a generalization of traditional signal processing, named *graph signal processing*.

Graph neural networks are a straightforward extension of multi-layer perceptrons tailored specifically for this type of data. For example, a graph neural network working at the node level is a model in the form $h_G(\mathbf{x}; \Theta)$ with G as a fixed graph, $\mathbf{x} \in \mathbb{R}^d$ as features associated with a single node, and Θ as a set of parameters. These models often intertwine *message passing layers*, which transform the node features into a new,

more complex set of features according to a transformation biased by the topology of the graph, and *local pooling layers*, where the graph is coarsened via a downsampling operation. Just as for multi-layer perceptrons, many different architectures have been proposed, and the reader can refer to [113] for a comprehensive review of them.

Graph signal processing is another technique rising in popularity [114]. The topic intersects with machine learning whenever we define a model mapping an input signal, i.e., a mapping over graphs $x : V \rightarrow \mathbb{R}$, to an output signal, i.e., $y : V \rightarrow \mathbb{R}$. Such a method decomposes a function on a graph according to the eigenvectors of the Laplacian operator of the underlying graph. These eigenvectors are interpreted as the Fourier basis in which the Fourier transform decomposes the signal, and the corresponding eigenvalues are interpreted as frequencies. This allows us to define filters just as with Euclidean-based signal processing.

5.2 Graph (dis)similarity measures

A graph *similarity measure* is a kernel function on the set of graphs $\mathcal{G}$, $\kappa : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}_{\geq 0}$. Defining graph similarity measures allows us to tackle graph-level machine learning tasks using kernel machines. A common setting for such tasks consists of learning a function $f : \mathcal{G} \rightarrow \mathbb{R}$, given the training set $\{(G^{(i)}, y^{(i)})\}_{i=1}^m$. The model will be a kernel machine in the form

$$h(G) = \sum_{i=1}^m \alpha_i \kappa(G, G^{(i)}). \quad (5.1)$$

Once we have a suitable similarity metric between graphs, κ , we can straightforwardly extend the machine learning models we already have (regressors, classifiers, clustering, etc.) to work natively with this kind of data. This is an important advantage compared to other techniques, as we can reuse the same statistical learning frameworks while GNNs and GSP require substantially different formalisms than their Euclidean-based counterparts.

Graph kernels can be categorized as *complete* and *incomplete*. A graph kernel $\kappa(G_1, G_2) = \langle \phi(G_1), \phi(G_2) \rangle$ is complete if and only if $\phi(G)$ is injective. In other words, a complete kernel is at least as powerful as the graph isomorphism test. As there is no polynomial-time classical nor quantum algorithm for solving the graph isomorphism problem¹, we can assume incomplete kernels to be efficient but weak approaches

¹The existence of a polynomial-time algorithm for the graph isomorphism problem is an open question,

and complete ones to be powerful but computationally expensive. Some examples of incomplete kernels are those based on random walks or the Weisfeiler-Leman test [116].

An example of a tautologically complete kernel is the graph isomorphism kernel, which maps the pair (G_1, G_2) to one if the graphs are isomorphic and zero otherwise. This kind of kernel loses any nuance regarding how similar or dissimilar two graphs are. One can rely on *graph edit distances* to overcome this issue.

The graph edit distance, which is a *dissimilarity* measure over graphs, is denoted here as $\text{GED} : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}_{\geq 0}$ and estimates how dissimilar two graphs are by counting the (minimum) number of operations required to modify one graph to make it isomorphic to the other. It can be applied to labeled graphs as well, i.e., structures in the form (V, E, ℓ) where $\ell : V \rightarrow \{1, \dots, k\}$ is a labeling function of the vertices.

More formally, a *graph edit operation* is a mapping $\mathcal{G} \rightarrow \mathcal{G}$ and can be of various types. For unlabeled graphs, which are the ones considered throughout this work, they can be the insertion of a new node, the deletion of an existing node, the insertion of an edge (subject to the constraint that both vertices exist), and the deletion of an edge (subject to the constraint that such an edge exists). For labeled graphs, we also have the relabeling of a vertex, and the insertion of a vertex must specify the label for the new vertex [117]. A composition of (valid) graph edit operations is called a *graph edit path* and is denoted with P . The number of operations in a graph edit path is denoted as $\text{len}(P)$. Finally, the graph edit distance can be expressed as

$$\text{GED}(G_1, G_2) = \min_{P \text{ such that } G_1 \simeq P(G_2)} \text{len}(P). \quad (5.2)$$

In this simplified form, we assume each operation has the same cost. Clearly, for $P = (\text{op}_1, \dots, \text{op}_n)$ we can assign each operation a different cost, $\text{cost}(\text{op})$, then we replace the objective function to minimize with the function $\text{cost}(P) = \sum_{i=1}^n \text{cost}(\text{op}_i)$. An example of the calculation of GED is shown in Figure 5.1. Note that calculating the GED is an NP-Complete combinatorial optimization task and is believed to be harder than the graph isomorphism problem.

with the best known runtime being quasi-polynomial [115].

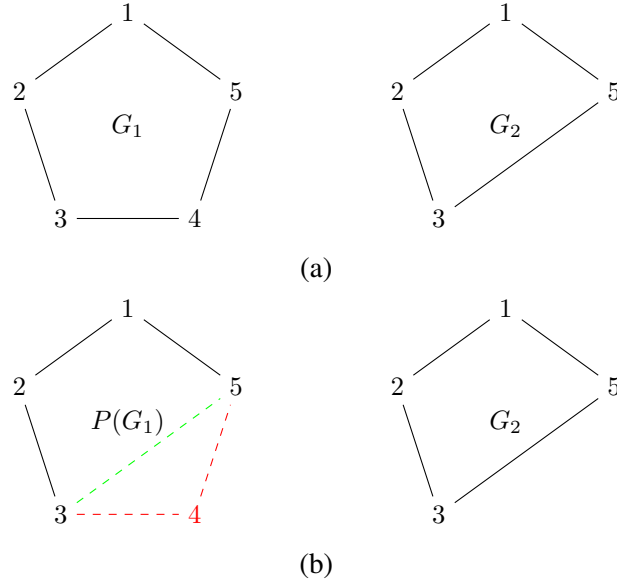


Figure 5.1: Calculation of a graph edit distance on a pair of small graphs. (a) shows the input and (b) highlights the steps of the smallest graph edit path, in green the additions and red the deletions.

5.3 Quantum algorithm for the Graph Edit Distance

In this section, we show a quantum algorithm for the calculation of the GED. We implement such an algorithm on a quantum annealer, thanks to a reduction from a GED to QUBO optimization, and on a (simulated) quantum-circuit-based quantum device, thanks to the QAOA algorithm.

5.3.1 Rephrasing the GED

Our reduction from GED to QUBO works under some mild assumptions. Firstly, we consider the graph unlabelled. Secondly, let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be the graphs at hand; we impose $|V_1| = |V_2|$, and such constraints do not imply any loss of generality as if that's not the case, the first graph edit operations would be necessarily added as many isolated vertices as needed to the smallest graph (as the graph is unlabelled, there is no discussion in regards to which labels is the most convenient for the vertex just added).

We can rephrase the GED formulation by explicitly considering the graph isomorphism between G_1 and $P(G_2)$. Two graphs are isomorphic if it exists a bijection $\pi : V_1 \rightarrow V_2$ between the vertices of the first graph and the vertices of the second

graph such that $\pi(V_1) = V_2$, which trivially holds as the vertices are unlabelled and the same number, and $\pi(E_1) = E_2$. Here, we denote with $\pi(V_1)$ the set of mapped vertices $\{\pi(v) \mid v \in V_1\}$ and $\pi(E_1)$ the set of mapped edges $\{(\pi(v), \pi(v')) \mid v, v' \in E_1\}$. Furthermore, we will denote the inverse of such relations as π^{-1} .

The GED can then be rephrased via a minimization problem over the possible bijections,

$$\text{GED}(G_1, G_2) = \min_{\pi: V_1 \rightarrow V_2} \left(\min_{P: G \rightarrow G} \text{len}(P) \right) \quad (5.3)$$

$$\text{constrained to } \pi(P(E_1)) = E_2. \quad (5.4)$$

Note that the calculation of GED is NP-hard in general; however, finding the value $\min_{P: G \rightarrow G} \text{len}(P)$ over the satisfying constraint can be done in polynomial time once fixed π . This is because the minimum graph edit path, given a fixed bijection of vertices, can be found via local operations only in a greedy fashion: adding the edges present in E_1 and missing in E_2 according to the mapping, and removing the edges present in E_1 and missing in E_2 ; this requires simply traversing the two graphs. Furthermore, note that the procedure of adding and removing edges from E_1 only can be rephrased into adding edges only to both E_1 and E_2 (cf. Figure 5.2). This leads to another simplification of our formulation,

$$\text{GED}(G_1, G_2) = \min_{\pi: V_1 \rightarrow V_2} (|\pi(E_1) \setminus E_2| + |E_2 \setminus \pi(E_1)|). \quad (5.5)$$

5.3.2 Reduction to GED to QUBO

The QUBO optimization problem takes as input an upper-triangular matrix of real values $Q \in \mathbb{R}^{n \times n}$ and aims to output a vector $\mathbf{x}^* \in \{0, 1\}^m$ such that

$$\mathbf{x}^* = \arg \min_{\mathbf{x}} \mathbf{x}^T Q \mathbf{x}. \quad (5.6)$$

Such a problem is an NP-complete combinatorial optimization problem [118], implying that 1) we expect no classical nor quantum algorithm to solve such a problem in polynomial time, and 2) we can use QUBO optimization to solve any combinatorial optimization problem with at most a polynomial slowdown, and that includes the graph edit distance. The relevance of the QUBO problem in quantum computing stems from

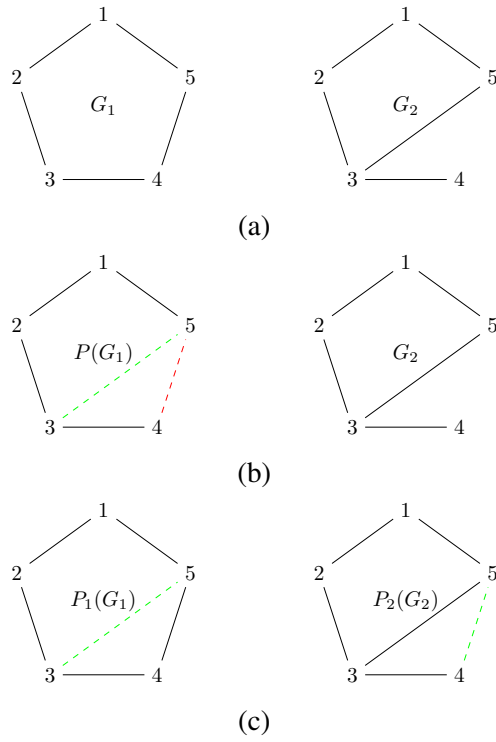


Figure 5.2: Equivalence in the calculation of the GED between a pair of graphs (a) by calculating the length of a single graph edit path P , made of addition and deletion operations, applied to the first graph (b) or by calculating the sum of lengths of two graph edit path, made of addition only, applied to both graphs (c).

their intimate connection with the Ising model that underlines the functioning of quantum annealers, a specialized, non-universal kind of quantum computer.

Now, let $\mathbf{x} \in \{0, 1\}^{n^2}$ be a vector of n^2 binary variables. The vector $\mathbf{x}$ is best represented as a $\{0, 1\}^{n \times n}$ matrix and must respect a constraint ensuring exactly one element per row and column is set. In such a case, $\mathbf{x}$ induces a bijection $\pi_{\mathbf{x}} : V_1 \rightarrow V_2$ such that $\pi_{\mathbf{x}}(v_1) = v_2$ for $v_1 \in V_1, v_2 \in V_2$ if and only if $x_{i,j} = 1$.

The QUBO formulation of our problem is composed of two parts: a term Q_h , denotes as the hard constraint and ensuring $\mathbf{x}$ induces a mapping between vertices, and a term Q_s , denoted as the soft constraint and ensuring the mapping between vertices minimizes the number of edit operations. Here, hard or soft constraint denotes that solutions violating them are completely invalid or sub-optimal, respectively. The hard constraint reads as

$$Q_h(\mathbf{x}) = \underbrace{\sum_{1 \leq i \leq n} \left(1 - \sum_{1 \leq j \leq n} x_{i,j}\right)^2}_{(a)} + \underbrace{\sum_{1 \leq j \leq n} \left(1 - \sum_{1 \leq i \leq n} x_{i,j}\right)^2}_{(b)} \quad (5.7)$$

and ensures solution is a valid one, i.e., $\mathbf{x}$ induces to a bijection π . Specifically, for multiple $v, v' \in V_1$ mapped to the same vertex $v'' \in V_2$, the term (a) will be violated. Similarly, for a single $v \in V_1$ mapped to multiple vertices $v', v'' \in V_2$, the term (b) will be violated. This corresponds to having multiple 1 inside a row of the matrix $\mathbf{x}$ and having multiple ones on its columns, respectively.

The soft constraint Q_s calculates the inner term in Equation 5.5 and is defined as:

$$Q_s(\mathbf{x}) = \sum_{(i,j) \in E_1} R_{i,j}(\mathbf{x}) + \sum_{(i',j') \in E_2} S_{i',j'}(\mathbf{x}) \quad (5.8)$$

$$R_{i,j}(\mathbf{x}) = \sum_{1 \leq i' \leq n} x_{i,i'} \sum_{1 \leq j' \leq n} x_{j,j'} (1 - e_{i',j'}^{(2)}) \quad (5.9)$$

$$S_{i',j'}(\mathbf{x}) = \sum_{1 \leq i \leq n} x_{i,i'} \sum_{1 \leq j \leq n} x_{j,j'} (1 - e_{i,j}^{(1)}). \quad (5.10)$$

Here, $e_{i,j}^{(k)}$ denotes the presence or absence of the edge made of the vertices i, j in G_k ,

$$e_{i,j}^{(k)} = \begin{cases} 1, & (i, j) \in E_k \\ 0, & (i, j) \notin E_k \end{cases}. \quad (5.11)$$

Specifically, $R_{i,j}$ corresponds to $|\pi(E_1) \setminus E_2|$ and $S_{i,j}$ corresponds to $|E_2 \setminus \pi(E_1)|$. Such

a proposition follows by direct calculation,

$$\begin{aligned}
R_{i,j}(\mathbf{x}) &= \sum_{i'} \sum_{j'} \mathbf{x}_{i,i'} \cdot \mathbf{x}_{j,j'} \cdot (1 - e_{i',j'}^{(2)}) \\
&= \mathbf{x}_{i,\pi^{-1}(i)} \cdot \mathbf{x}_{j,\pi^{-1}(j)} \cdot (1 - e_{\pi^{-1}(i),\pi^{-1}(j)}^{(2)}) \\
&= \begin{cases} 1, & e_{\pi^{-1}(i),\pi^{-1}(j)}^{(2)} = 0 \\ 0, & e_{\pi^{-1}(i),\pi^{-1}(j)}^{(2)} = 1 \end{cases}
\end{aligned}$$

leading to

$$\sum_{(i,j) \in E_1} R_{i,j}(\mathbf{x}) = |E_1 \setminus \pi^{-1}(E_2)| = |\pi(E_1) \setminus E_2| \quad (5.12)$$

The QUBO formulation is the convex sum of the two terms Q_h, Q_s parameterized by two coefficients $\lambda_h, \lambda_s \in \mathbb{R}_{\geq 0}$,

$$Q(\mathbf{x}) = \lambda_h Q_h(\mathbf{x}) + \lambda_s Q_s(\mathbf{x}). \quad (5.13)$$

It is convenient to have $\lambda_h \gg \lambda_s$ to ensure all the valid solutions have cost much lower than the invalid ones. Mathematically, it is sufficient to have $\lambda_h > \max\{|E_1|, |E_2|\} \lambda_s$.

5.3.3 Quantum and classical device for tackling QUBO optimization

We compare some of the most popular heuristics solving QUBO, both classical and quantum ones.

Simulated Annealing is a probabilistic algorithm proposed in [119] that implements a random walk influenced by a temperature parameter T . It builds a sequence of candidate solutions. The initial element $\mathbf{x}_0$ is generated randomly, and subsequent elements are generated according to a probabilistic rule. First, we perturb the previous element of the sequence, i.e., $\mathbf{x}'_{i+1} = \text{perturb}(\mathbf{x}_i)$. Then $\mathbf{x}_{i+1} = \mathbf{x}'_{i+1}$ if the solution is qualitatively better (the cost of $\mathbf{x}'_{i+1}$ is lower than the cost of $\mathbf{x}_i$) or probabilistically, with a probability proportional to the difference between costs and the temperature parameter:

$$p(\mathbf{x}_{i+1} \leftarrow \mathbf{x}'_{i+1}) = \min \left\{ 1, \exp \left(-\frac{\text{cost}(\mathbf{x}'_{i+1}) - \text{cost}(\mathbf{x}_i)}{T} \right) \right\} \quad (\text{accept}) \quad (5.14)$$

$$p(\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i) = 1 - p(\mathbf{x}_{i+1} \leftarrow \mathbf{x}'_{i+1}) \quad (\text{reject}) \quad (5.15)$$

At high values of the temperature T , we easily accept worse solutions to explore the landscape of solutions. At lower temperatures, we exploit the neighborhood of the current solution, with $T = 0$ degenerating the algorithm to pure hill climbing. The temperature $T \in [0, 1]$ decreases according to the geometric schedule $T(k) = \exp(-k/T_{\max})$, known as the cooling schedule.

Simulated annealing is intimately connected to statistical physics as one of its main goals is minimizing the energy of a (classical) Ising model. The Ising model over n spins taking values in $\{\pm 1\}$ is described by an upper triangular matrix $H \in \mathbb{R}^{n \times n}$ (sometimes the diagonal coefficients of the model are denoted with $h_i = H_{ii}$ and the off-diagonal coefficients as $J_{ij} = H_{ij}$ for $i < j$). The cost of an assignment s is called energy and is denoted as:

$$\text{energy}(s) = s^\top H s = \sum_{i=1}^n h_i s_i + \sum_{1 \leq i < j \leq n} J_{i,j} s_i s_j. \quad (5.16)$$

It is straightforward to see how a simple change of variables can turn a QUBO minimization problem on n variables into an Ising minimization problem on n spins with negligible overhead.

Quantum Annealing is a probabilistic algorithm proposed in [120]. The original formulation was quantum-inspired, i.e., it was designed with quantum principles in mind but intended to be executed on classical hardware. However, the hardware company D-Wave produces actual quantum annealers, making these computations truly quantum.

Using a quantum annealer is motivated by the *adiabatic theorem*. At time $t = 0$, let the system encode the quantum Hamiltonian $H_0 = \sum_{i=1}^n \sigma_x$, where σ_x is the Pauli-X matrix, and let it be in its (easy to encode) ground state $|\psi\rangle = |+\rangle^{\otimes n}$. The Hamiltonian of the system evolves over time,

$$H(t) = \lambda(t)H_0 + (1 - \lambda(t))H_{\text{ising}}, \quad (5.17)$$

where H_{ising} is the quantum Hamiltonian associated with our problem, and $\lambda : [0, 1] \rightarrow [0, 1]$ with $\lambda(0) = 1$ and $\lambda(1) = 0$ is the scheduling function. According to the adiabatic theorem, if the system evolves slowly enough, the state of the quantum system will remain the ground state of the Hamiltonian throughout the process. However, the required evolution speed is polynomially related to the spectral gap of H_{ising} , which can be exponentially small in n .

In contrast to simulated annealing, which relies on thermal fluctuations, quantum

annealers uses quantum fluctuations and the tunneling effect. Quantum fluctuations allow the system to explore different states by superposition, and quantum tunneling enables the system to transition through energy barriers, potentially finding better solutions more efficiently than classical thermal fluctuations.

QAOA is an optimization algorithm denoted as 'hybrid', i.e., requiring both classical and quantum resources. Originally introduced in [121], the algorithm is based on a parameterized quantum circuit over n qubits, where n equals the number of variables (of the QUBO formulation) or spins (of the Ising model) starting at the state $|\phi_0\rangle = |+\rangle^{\otimes n}$ and evolving accordingly to the ansatz $U(\beta, \gamma) = \prod_{i=1}^p U_C(\gamma_i)U_B(\beta_i)$. In this equation, U_B is named the *mixing Hamiltonian* and applies a rotation over σ_x over each of the n qubits, i.e., $U_B(\beta) = \exp(-i\beta \sum_{i=1}^n \sigma_x^{(i)})$; $U_C(\gamma)$ is named the *cost Hamiltonian* and applies a series of CZ rotations over the terms of the Ising Hamiltonian encoding the problem, $U_C(\gamma) = \exp(-i\gamma H_{\text{ising}})$; p is the number of repetition of the unitaries U_B, U_C ; finally, $\gamma, \beta \in [-\pi, \pi]^{2p}$ are the set of parameters. By performing a measurement in the computational basis, we can calculate $\langle H_{\text{ising}}(\beta, \gamma) \rangle = \langle \phi_0 | U(\beta, \gamma) H_c U^\dagger(\beta, \gamma) | \phi_0 \rangle$.

Such a quantum circuit is embedded, just like any other variational algorithm, in a classical optimization routine. Once picked an *initialization strategy* (e.g. by sampling parameters i.i.d. from the uniform distribution $[-\pi, \pi]$, starting from $\beta = \gamma = 0$, starting from an educated choice coming from a computationally cheap classical routine such as for warm start) and an *optimization algorithm* (e.g. ADAM), we solve the following optimization problem

$$\gamma^*, \beta^* = \arg \min_{\gamma, \beta} \langle H_{\text{ising}}(\beta, \gamma) \rangle. \quad (5.18)$$

5.3.4 Experimental setup

We have compared the presented approach against a randomly generated dataset. Specifically, we have considered graphs drawn from the Erdos-Renyi random model of graphs, with $n \in \{3, 4, 5, 6, 7, 8, 9\}$ vertices. For each n , we generate $m = 4$ different graphs, each having a different probability that the edge is independently drawn between two vertices, $p \in \{0.1, 0.33, 0.66, 0.99\}$. This leads to seven different datasets, $\mathcal{D}_n = \{(G_i, G_j) \mid |V_i| = |V_j| = n\}_{1 \leq i, j \leq m}$ for $n = 3, \dots, 9$. Each dataset will have graphs with the same number of vertices to accommodate the constraints of the GED formulation. Such small instances allow us to use an exhaustive search (based on A* search) to find the global optimum and compare the results of the heuristics with the true

value of the computation. A limitation of this approach is that we are effectively testing the various techniques in a regime of small graphs, which might not be relevant for all tasks the GED might be used for.

Hyper-parameters for the QUBO formulation

The QUBO formulation configures the hyper-parameters λ_h, λ_s in Equation 5.13. We have tested $\lambda_h = 1$ and $\lambda_s \in \{0.01, 0.05, 0.1, 0.2, 0.3, \dots, 1.0\}$. Note that by ensuring $\lambda_h \gg \lambda_s$, we will have a large energy separation between valid and invalid solutions. However, for $n = 9, |E| = n^2 = 81, \lambda_h = 1.0$, we obtain $\lambda_s = \lambda_h/81 \approx 0.012$. Such small values can hardly be compatible with the execution on a (NISQ) quantum device as it will require too large of precision to be encoded perfectly, possibly being approximated as $\lambda_s \approx 0.0$. Therefore, it is useful for practical reasons to test larger values of λ_s and evaluate whenever such choices lead to a satisfactory compromise between hardware limitations and reaching valid albeit sub-optimal solutions.

As the extensive test of any possible configuration on quantum devices is too expensive due to the lack of availability of the hardware, we have first identified the best values for (λ_h, λ_s) with simulated annealing and continue our benchmark with such values only. Again, this is a sub-optimal choice, as we are not guaranteed that the best hyper-parameters when running simulated annealing are necessarily the ones that work best for the other procedures.

We proceed by running simulated annealing with a number of reads equal to 1000, over the seven datasets and all the possible values of λ_s . We measure the average *relative difference*, where relative difference indicates the normalized difference between the global optimal solution found by extensive search and the sub-optimal solution found using simulated annealing. For s optimal solution and $\bar{s}$ sub-optimal one, the relative difference is $\Delta(s, \bar{s}) = (\bar{s} - s)/\bar{s} \in [0, 1]$ or ∞ is the solution $\bar{s}$ is invalid. Table 5.1 shows the results. The values of λ_s that consistently obtain the best results are 0.1, 0.3, 0.4, 0.5, 0.6, 0.7; notably, the only value satisfying $\lambda_h \gg \lambda_s = 0.01$ has performance much worse than some other combinations. Among these values, we choose $\lambda_s = 0.1$ as the Q_s will be larger and thus less affected by the noise and encoding error.

Configuration of the QUBO solvers

Simulated annealing We have implemented simulated annealing in Ocean SDK. The number of reads is set to 10^3 , and the number of sweeps is set to 10^6 . Let (M, m)

n	Values of λ_s											
	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0.05	0.01
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.42	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	0.52	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
6	0.53	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	0.54	0.32	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.09
8	0.48	0.48	0.17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.09	0.15
9	0.62	0.50	0.38	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.09	0.24

Table 5.1: Average relative difference for the experiments with $\lambda_h = 1, \lambda_s \in \{0.01, 0.05\} \cup \{i/10 \mid i = 1, \dots, 10\}$ with respect to the number of vertices n .

be the absolute value of the largest and smallest entry of the QUBO matrix. The temperature schedule is set to the default choice, a geometric sequence beginning with $T_{\text{start}} = M/\log 2$ and ending with $T_{\text{end}} = m/\log 100$.

Quantum annealing We have used three different quantum annealers of the same hardware vendor D-Wave, namely (i) D-Wave 2000Q [122]; (ii) D-Wave Advantage 1.1 [123]; (iii) D-Wave Leap Hybrid Solver. The detailed characteristics of each solver can be found in the respective technical documents; the predominant characteristic here is the number of physical qubits, which are 2041 and 5436 for the 2000Q and Advantage 1.1. The D-Wave Leap is a proprietary black-box optimization device.

We have tested (for 2000Q and Advantage 1.1) four different annealing schedules, characterized by the total time τ and a sequence of values $\{(p_i, \tau_i)\}$ where p_i indicates the progress of the annealing process at the time $\tau_i \leq \tau$. The three annealing schedules are:

1. *default configuration (DC)*: $\tau = 20\mu s$ with $\{(0.0, \tau_0 = 0), (1.0, \tau)\}$;
2. *Short Time (ST)*: $\tau = 1\mu s$ with $\{(0.0, \tau_0 = 0), (1.0, \tau)\}$;
3. *Long Time (LT)*: $\tau = 500\mu s$ with $\{(0.0, \tau_0 = 0), (1.0, \tau)\}$;
4. *Pause Middle (PM)*: $\tau = 100\mu s$ with $\{(0.0, \tau_0 = 0), (0.45, 5\mu s), (0.45, 99\mu s), (1.0, \tau)\}$.

According to [124], the pause has a predominant role in the performance of the quantum annealer and suggest larger pauses to increase the success. Similarly, larger annealing time might help the annealing process to always remain in the ground state throughout the whole evolution but also can be affected much more by external noise than shorter

schedules. The number of reads are different for each schedule: ST with 10^4 , DC with both 10^3 and 10^4 , the others with 10^3 and we have kept only the lowest energy result. Larger annealing times are more expensive economically too.

QAOA We have used the implementation of QAOA present in the framework IBM Qiskit. The optimization of the variational form is done via the stochastic gradient descent-based optimizer COBYLA [125] for a number of iterations equal to 10^3 . The QAOA ansatz has a number of layers of from 1 to 3. The parameters are randomly initialized, drawn from the uniform distribution in the range $[-\pi/2, \pi]$. The quantum circuit is simulated noiselessly with a number of samples drawn from one equal to 2048.

Measuring the performance

We evaluate the performance of the approach in terms of: 1) *average relative difference* as previously introduced; 2) *average success probability*, i.e., as the percentage of GED runs that have relative difference zero; 3) *average high-quality probability*, i.e., as the percentage of GED runs that have relative difference below the threshold 0.2.

5.3.5 Results

Performance of various schedules of quantum annealers

Quantum annealers offer a wide range of devices and schedules. We compare different combinations in Figure 5.3, excluding D-Wave Leap which does not expose any relevant configurations. We observe that the number of runs significantly impacts the quality of the solutions, more so than the annealing time of a single run. The optimal tradeoff between short and long annealing times is found to be at $20\mu s$. Furthermore, these experiments indicate that using a pause does not substantially improve performance.

Performance of the quantum annealers on their best schedule

We evaluate the three types of quantum annealers, including D-Wave Leap, on their best performing configuration ($20\mu s$ of annealing time with 10^4 runs). The results are shown in Figure 5.4. The D-Wave Advantage, being a more modern and technologically advanced device, slightly outperforms the D-Wave 2000Q as expected, notably with the anomalous case $n = 7$. Such cases can be explained in terms of experimental error (cf. error bars).

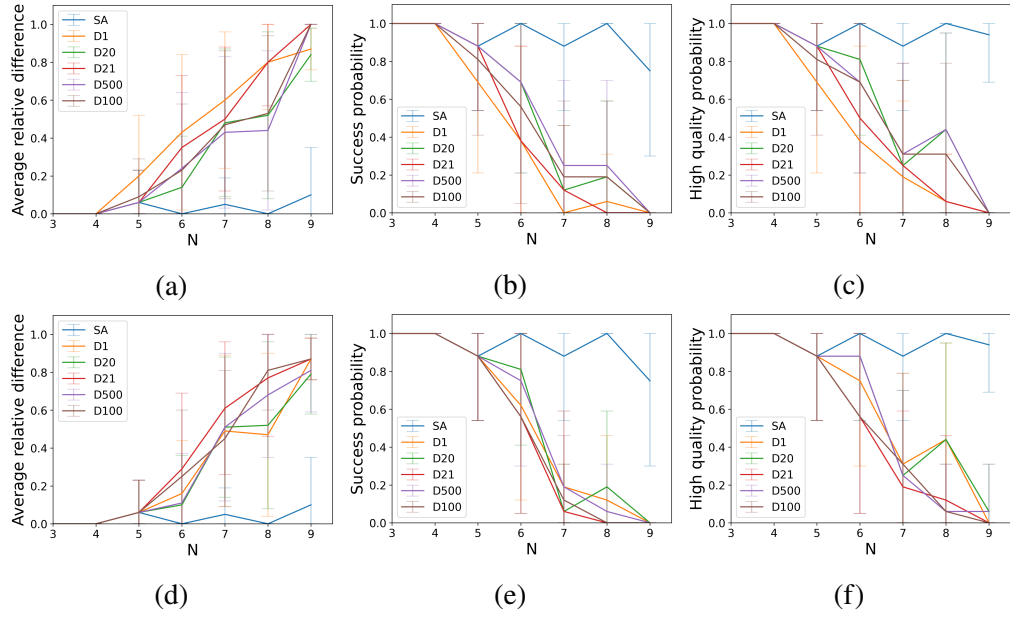


Figure 5.3: Performance of D-Wave 2000Q quantum annealer (plots a-b-c) and D-Wave Advantage 1.1 (plots d-e-f) in terms of relative difference (a-d), success probability (b-e), high-quality probability (c-f) under different configurations. The standard deviation is pictured as error bars. Legend: SA is simulated annealing, D1 is annealing ($1\mu s \times 10^4$ runs), D20 is annealing ($20\mu s \times 10^4$ runs), D21 is annealing ($20\mu s \times 10^3$ runs), D500 is annealing ($500\mu s \times 10^3$ runs), D100 is annealing ($100\mu s$ paused in the middle and $\times 10^3$ runs).

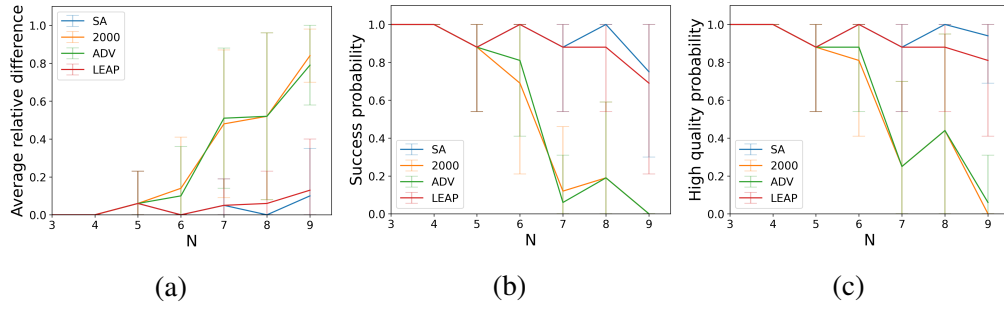


Figure 5.4: Performance of simulated annealing (classical baseline) compared to the D-Wave devices in terms of relative difference (a), success probability (b) and high quality probability (c). The standard deviation is pictured as error bars. Legend: SA is simulated annealing, D2000 is D-Wave 2000Q ($20\mu s \times 10^4$ runs), ADV is D-Wave Advantage 1.1 ($20\mu s \times 10^4$ runs), LEAP D-Wave Leap.

It is important to note the consistent lack of success of quantum annealing in solving this task compared to simulated annealing. This can realistically be attributed to hardware limitations and sensitivity to noise in this generation of quantum annealers, as shown by other benchmarks indicating uncompetitive performance. D-Wave Leap, despite showing performance closer to simulated annealing, cannot be fully classified as quantum annealing due to the lack of transparency in its functioning.

Time to solution of quantum annealers

Finally, we have identified which configuration promises the best tradeoff between the quality of the solution and total annealing time. This observation is relevant because the cost of using a quantum annealer is proportional to the amount of time used by the machine.

We measure the *time to solution* (TTS) as

$$\text{TTS} = \frac{\text{number of runs} \times \text{annealing time per run}}{\text{high quality probability}} \quad (5.19)$$

for both D-Wave 2000Q and D-Wave Advantage. It measures how much time in seconds is required on a quantum annealer to find a high-quality solution. The results are shown in Table 5.2. Our experiments show that having the shortest annealing time with a large number of runs gives the best tradeoff in terms of TTS. For $n = 9$ vertices, no experiment found any high-quality solution, thus it was not possible to estimate such probability.

n	D-Wave 2000Q					D-Wave Advantage				
	A	B	C	D	E	A	B	C	D	E
3	0.01	0.20	0.02	0.50	0.10	0.01	0.20	0.02	0.50	0.10
4	0.01	0.20	0.02	0.50	0.10	0.01	0.20	0.02	0.50	0.10
5	0.01	0.23	0.02	0.57	0.12	0.01	0.23	0.02	0.57	0.11
6	0.03	0.25	0.04	0.72	0.14	0.01	0.23	0.04	0.57	0.18
7	0.05	0.80	0.08	1.61	0.32	0.03	0.80	0.11	2.00	0.32
8	0.17	0.45	0.33	1.14	0.32	0.02	0.45	0.17	8.33	1.67

Table 5.2: Time To Solution for the two hardware configurations, measured in seconds. Legend: n : number of vertices, A: $1\mu s$ per run $\times 10k$ runs, B: $20\mu s$ per run $\times 10k$ runs, C: $20\mu s$ per run $\times 1k$ runs, D: $500\mu s$ per run $\times 1k$ runs, E: $100\mu s$ per run $\times 1k$ runs with paused annealing.

Resource usage of quantum annealers

The D-Wave devices allow the use of a relatively large number of physical qubits, in the order of thousands. However, the limited connectivity of these often forces us to group multiple physical qubits as a single, larger logical qubit characterized by increased connectivity. This grouping is necessary depending on the sparsity of the QUBO matrix of the specific problem. Here, we present a few important statistics regarding our problem: the number of physical and logical qubits used, the *maximum length of the chain*, i.e., the maximum number of physical qubits needed to encode a single logical qubit, and the *chain strength*, i.e., the strength of the interaction between physical qubits belonging to the same chain.

The results are shown in Table 5.3. Notably, the D-Wave Advantage is capable of handling much larger problems than the D-Wave 2000Q (despite having more or less the same overall performance) due to both the higher number of physical qubits and the much more flexible connectivity between them, leading to shorter chains and thus fewer errors.

Performance of QAOA

We evaluate the performance of the QAOA algorithm, focusing on a regime where the corresponding quantum circuits can be executed on the first generation of superconducting QPUs. In Figure 5.5, we compare the performances of QAOA (with one and three repetitions of the ansatz) with simulated annealing. We expand our comparison by testing the VQE algorithm, typically used in quantum chemistry tasks, to determine

n	D-Wave 2000				D-Wave Advantage			
	L	P	CL	CS	L	P	CL	CS
3	9	22	3	1.44	9	11	2	1.44
4	16	65	5	1.76	16	33	3	1.76
5	25	175	9	2.03	25	77	4	2.03
6	36	377	14	2.27	36	159	6	2.27
7	49	711	21	2.48	49	296	9	2.48
8	64	1226	28	2.68	64	506	12	2.68
9	81	1474	28	2.84	81	832	16	2.87

Table 5.3: Resource consumption in quantum annealers. Legend: n : number of vertices, L: logical qubits, P: physical qubits, CL: maximum chain length, CS: chain strength.

whether using a different ansatz than QAOA might lead to better performances despite diverging from QAOA theory. The ansatz chosen for VQE consists of repetitions of R_y and R_z rotations on each qubit, with independent free parameters, and CNOT gates between each pair of qubits (i, j) where $i < j$.

Note that due to the limitation of simulating a quantum circuit on classical hardware, we could test quantum circuits with at most 25 variables (equivalent to computing the GED over graphs with 5 vertices).

Even in a noiseless environment, the variational approach cannot reach an exact solution for any problem with $n \geq 4$ vertices. Several reasons contribute to the failure of the quantum approach, including the fact that the QAOA ansatz with a small number of layers is not guaranteed to reach the correct solution at all (though Farhi, Goldstone, and Gutmann [121] guarantees the possibility of reaching the correct solution, albeit possibly inefficiently, with an infinite number of repetitions of the ansatz). Additionally, increasing the expressivity of the quantum circuit may encounter the issue of the barren plateau.

Resource usage of QAOA

We quantify the usage of resources (qubits, gates, parameters) for the variational approach in Table 5.4. To ensure the resource estimation is realistic, we transpiled all quantum circuits targeting a fully connected QPU with a universal gate set $\{U_3, CNOT\}$. Notably, VQE with the chosen ansatz is much more resource-intensive.

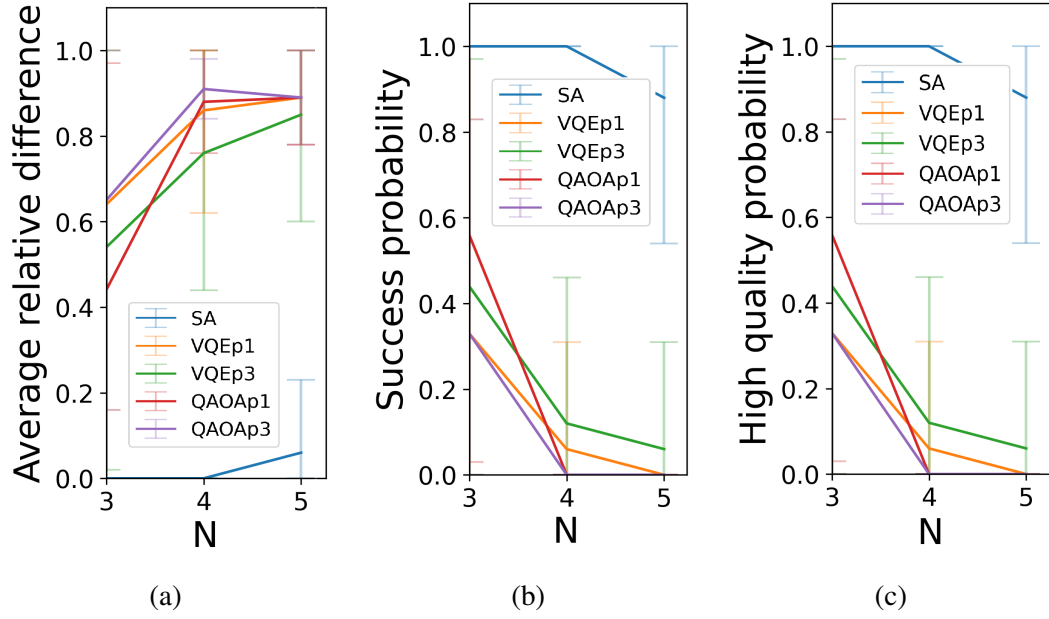


Figure 5.5: Performance of variational algorithms in terms of relative difference (a), success probability (b) and high-quality probability (c). The standard deviation is pictured as error bars. Legend: V1 is VQE ($p = 1$), V3 is VQE ($p = 3$), Q1 is QAOA ($p = 1$), Q3 is QAOA ($p = 3$).

n	Q	VQE						QAOA					
		$p = 1$			$p = 3$			$p = 1$			$p = 3$		
		P	D	S	P	D	S	P	D	S	P	D	S
3	9	36	17	54	72	37	144	2	40	96	6	88	258
4	16	64	31	152	128	65	424	2	79	272	6	119	792
5	25	100	49	350	200	101	1000	2	125	638	6	187	1876
6	36	144	71	702	288	145	2034	2	183	1220	6	275	3606
7	49	196	97	1274	392	197	3724	2	257	2109	6	385	6255
8	64	256	127	2144	512	257	6304	2	329	3625	6	494	11259
9	81	324	161	3402	648	325	10044	2	423	5535	6	635	16263

Table 5.4: Resource consumption in variational algorithms. Legend: n : number of vertices, Q : number of qubits, P: parameters, D: depth, S: size.

Comparison with other classical heuristics

We finally compare the QUBO optimization approach with various non-QUBO classical heuristics proposed in the literature. Conveniently, the library GedLib [126] contains implementations of these heuristics. Using the same datasets as in the previous experiments, we present the success probability of each classical heuristic implemented in GedLib in Figure 5.6a and compare them with the QUBO solutions, both classical and quantum, in Figure 5.6b.

V	B1	B2	B3	B4	B5	P1	H1	R1	A1	W1	I1	B6	S1	N1	R2	B7	R3	B8	SA	H2	S2
3	1.00	1.00	1.00	1.00	0.00	0.00	0.00	1.00	0.89	1.00	0.78	1.00	1.00	0.78	0.78	0.67	1.00	1.00	1.00	0.00	1.00
4	0.88	0.88	1.00	0.88	0.00	0.00	0.00	0.88	0.94	0.88	0.88	0.88	0.75	0.75	0.62	0.75	0.94	1.00	1.00	0.00	0.88
5	0.62	0.62	0.69	0.62	0.00	0.00	0.00	0.75	0.69	0.62	0.62	0.62	0.62	0.62	0.62	0.56	1.00	0.81	1.00	0.00	0.62
6	0.44	0.44	0.56	0.44	0.00	0.00	0.00	0.56	0.75	0.44	0.81	0.44	0.31	0.38	0.25	0.25	0.81	0.69	1.00	0.00	0.38
7	0.38	0.38	0.50	0.38	0.00	0.00	0.00	0.56	0.50	0.31	0.62	0.38	0.44	0.38	0.12	0.19	0.56	0.56	1.00	0.00	0.31
8	0.38	0.38	0.38	0.38	0.00	0.00	0.00	0.56	0.75	0.44	0.56	0.38	0.19	0.38	0.31	0.06	0.75	0.75	0.94	0.00	0.44
9	0.38	0.38	0.38	0.38	0.00	0.00	0.00	0.50	0.62	0.25	0.75	0.38	0.19	0.38	0.25	0.00	0.62	0.50	0.69	0.00	0.25

(a)

V	SA	D-Wave 2000						D-Wave Advantage						D-Wave Leap	VQE		QAOA	
		D1	D20	D21	D500	D100	D1	D20	D21	D500	D100	$p = 1$	$p = 3$		$p = 1$	$p = 3$		
3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.33	0.44	0.56	0.33		
4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.06	0.12	0.00	0.00		
5	0.88	0.69	0.88	0.88	0.88	0.81	0.88	0.88	0.88	0.88	0.88	0.88	0.00	0.06	0.00	0.00		
6	1.00	0.38	0.69	0.38	0.69	0.56	0.62	0.81	0.56	0.75	0.56	1.00	-	-	-	-		
7	0.88	0.00	0.12	0.12	0.25	0.19	0.19	0.06	0.06	0.19	0.12	0.88	-	-	-	-		
8	1.00	0.06	0.19	0.00	0.25	0.19	0.12	0.19	0.00	0.06	0.00	0.88	-	-	-	-		
9	0.75	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.69	-	-	-	-		

(b)

Figure 5.6: Average Success probability (higher is better), Legend: V: number of vertices, B1: Branch, B2: Branch Fast, B3: Branch Tight, B4: Branch Uniform, B5: Branch Compact, P1: Partition, H1: Hybrid, R1: Ring, A1: ANCHOR_AWARE_GED, W1: Walks, I1: IPFP, B6: BIPARTITE, S1: SUBGRAPH, N1: NODE, R2: RING_ML, B7: BIPARTITE_ML, R3: REFINE, B8: BP_BEAM, SA: Simulated Annealing (GEDLib implementation, not D-Wave's) H2: HED, S2: STAR. Average success probability (higher is better). Legend: V: number of vertices, SA: Simulated Annealing (D-Wave's implementation), D1: annealing $1\mu s \times 10^4$ runs, D20: annealing $20\mu s \times 10^4$ runs, D21: annealing $20\mu s \times 10^3$ runs, D500: annealing $500\mu s \times 10^3$ runs, D100: annealing $100\mu s$ paused in the middle, $\times 10^3$ runs, p : number of repetition of the variational form.

Chapter 6

Quantum kernels on topological data

The previous section has discussed the potential advantages of quantum kernels when dealing with more structured objects. This has led to a quantum algorithm that works on graphs. Justifying the use of a quantum device can certainly benefit from more significant speedups, which might be obtained by relying on even more structured data. An example of such structures is the *abstract simplicial complex* (ASC).

An ASC encodes the relationship between n entities or vertices, both pairwise and in larger groups. These relationships are called p -simplices, which are subsets of $p + 1$ vertices and can be combinatorially many in n . The interaction between p -simplices is highly structured: the neighborhood of any p -simplex has $O(n^2)$ elements¹. Such a structure opens the possibility for the *quantum linear algebra* framework to analyze the spectral properties of linear operators defined over the ASCs. These operators will be large but constrained in their sparsity structure, with a straightforward procedure to encode them on a quantum computer but very challenging to represent explicitly on a classical system.

An example of such an approach has been utilized by topological data analysis algorithms on quantum devices, which exploit the spectral properties of the Combinatorial Hodge Laplacian operator to infer (a value intimately related to) the Betti number, a topological invariant over ASCs.

This chapter is devoted to investigating the formulation and applications of defining quantum kernels over the topological features extracted from quantum algorithms. It explores how such an approach compares to more flexible methods, such as the graph

¹Two p -simplices are adjacent if they have a common subset of p elements, such a subset is a $(p - 1)$ simplex and is named a *face*. In such a case, these two simplices are denoted as lower-adjacent. Similarly, if they both are faces of a common $(p + 1)$ -simplex, they are upper-adjacent.

edit distance. Additionally, it addresses the broader question of whether relying on a more constrained context that potentially leads to larger speed-ups compromises their applicability.

In Section 6.1, we introduce a series of tasks on ASCs that can benefit from the use of machine learning. In Section 6.2, we detail the quantum algorithms for topological data analysis and discuss their applicability. In Section 6.3, we model a quantum kernel over the topological features extracted by a quantum algorithm and discuss a potential implementation on fault-tolerant hardware. In Section 6.5, we discuss the difference between such an approach and the graph edit distance. This chapter is based on [5].

6.1 Machine learning on topological data

We start by introducing the basic notions of abstract simplicial complexes and their applications in topological data analysis. Then, we briefly discuss the role of topological features in machine learning.

Abstract simplicial complexes and homology

An ASC $\Gamma = (V, \Sigma)$ consists of a finite, non-empty set of vertices V of cardinality $|V| = n$, and a set of *simplices* Σ , its elements named simplices $\sigma \in \Sigma$ being non-empty subsets of V , and such that Σ is closed under inclusion. We denote a simplex σ of cardinality $p + 1$ as a p -*simplex*, and denote the p -simplices of Γ with $\Sigma_p \subset \Sigma$. Finally, $\deg(\Gamma) = \max_{\sigma \in \Sigma} |\sigma| - 1$.

A linear algebraic structure can be given to these combinatorial objects by defining the p -chain space as the (real-valued) span of the set of p -simplices,

$$C_p = \text{span}(\Sigma_p; \mathbb{R}). \quad (6.1)$$

The elements of such a space are called p -chains. An important linear operator is the *boundary* $\partial_p : C_p \rightarrow C_{p-1}$, which maps (a linear combination of) p -simplices to the alternating sum of their faces,

$$\partial_p c = \partial_p \left(\sum_{i=1}^m \sigma_i \right) = \sum_{i=1}^m \partial_p \sigma_i = \sum_{i=1}^m \sum_{j=0}^p (-1)^j (\sigma_i \setminus [\sigma_i]_j). \quad (6.2)$$

Here, the p -simplex σ_i is encoded as an ordered tuple $(v_0, \dots, v_p)$ such that $v_0 < \dots < v_p$

according to the natural global ordering of the vertices; as such, $[\sigma_i]_j$ is the j -th element of the tuple and $(\sigma_i \setminus [\sigma_i]_j) = (v_0, \dots, v_{j-1}, v_{j+1}, \dots, v_p)$. Note that $\partial \circ \partial = 0$. The p -Combinatorial Hodge Laplacian is the linear operator $\mathcal{L}_p : C_p \rightarrow C_p$ defined as

$$\mathcal{L}_0 = \mathcal{L}_0^\uparrow = \partial_1^\dagger \partial_1 \quad p = 0 \quad (6.3)$$

$$\mathcal{L}_p = \mathcal{L}_p^\downarrow + \mathcal{L}_p^\uparrow = \partial_p^\dagger \partial_p + \partial_{p+1} \partial_{p+1}^\dagger \quad p = 1, \dots, \deg(\Gamma) - 1 \quad (6.4)$$

$$\mathcal{L}_p = \mathcal{L}_p^\downarrow = \partial_p^\dagger \partial_p \quad p = \deg(\Gamma). \quad (6.5)$$

Here, $\mathcal{L}_p^\downarrow$ denotes the down Laplacian and encodes the lower-adjacent relationship between p -simplices, i.e., sharing a face. Similarly, $\mathcal{L}_p^\uparrow$ denotes the up Laplacian. The dimension of $\ker(\mathcal{L}_p)$ is called the p -Betti number,

$$\beta_p = \dim \ker(\mathcal{L}_p). \quad (6.6)$$

Such a quantity is an important topological invariant, intuitively corresponding to the number of ' p -dimensional holes' in Γ . These can be obtained by counting the p -cycles (p -chains with boundary zero, or elements of $\ker(\partial_p)$), keeping the ones that are linearly independent, and removing from this set of p -cycles the ones that are covered by a group of $(p + 1)$ -simplices (elements of $\text{im}(\partial_{p+1})$). Algebraically, this corresponds to

$$\beta_p = \dim \left(\ker(\partial_p) / \text{im}(\partial_{p+1}) \right). \quad (6.7)$$

The quotient space $\ker(\partial_p) / \text{im}(\partial_{p+1})$ is called the p -homology group.

Construction of ASCs from existing data

Real-world datasets for simplicial complexes are rather rare; such an object is usually constructed using a graph as a skeleton. That is the case for clique complexes, i.e., objects derived from a graph $G = (V, E)$ via $\Gamma = (V, \Sigma_{\text{cl}})$ where $\Sigma_{\text{cl}} = \{\emptyset \neq \sigma \subset V \mid \forall v, v' \in \sigma : (v, v') \in E\}$.

Another popular approach for obtaining ASCs is the *Vietoris-Rips complex*. This is obtained starting from a cloud of points embedded in an Euclidean space, $PC = \{\mathbf{x}_1, \dots, \mathbf{x}_m\} \subseteq \mathbb{R}^d$. We fix a threshold $\tau \in \mathbb{R}_{\geq 0}$, and define the parameter of a subset of the given points as

$$\text{diam}(I) = \max_{i, j \in I} \|\mathbf{x}_i - \mathbf{x}_j\|_2. \quad (6.8)$$

Then, the corresponding Vietoris-Rips complex is the ASC

$$\text{VR}_\tau(PC) = (\{1, \dots, m\}, \Sigma), \quad \Sigma = \{\emptyset \neq \sigma \subseteq V \mid \text{diam}(\sigma) \leq \tau\}. \quad (6.9)$$

Filtrations and persistent homology

A notable aspect of this construction is the role of topological features in a noisy setting. As the point cloud might have a certain error, it might result in a perturbation of the topological features, i.e., p -holes appearing and disappearing uniquely due to the noise. A more robust approach has been proposed, the *persistent homology*.

Such an approach is based on the use of *filtrations*. A filtration $\mathcal{F} = \{O_i\}_{i \in I}$ over the indices $I \subset \mathbb{N}$ is an indexed sequence of algebraic structures such that $i < j \implies O_i \subseteq O_j$. These algebraic structures can be abstract simplicial complexes too, assuming they are defined on the same set of vertices. In such a case, the filtration $\mathcal{F}$,

$$\mathcal{F} = \{\Gamma_i = (V_i, \Sigma_i)\}_{i \in I} \quad (6.10)$$

is such that all $V_i, i \in I$ correspond to the same set of vertices and $i < j \implies \Sigma_i \subseteq \Sigma_j$.

A common way to create filtrations from a point cloud is the *Vietoris-Rips filtration*,

$$\text{VR}(P) = \{\text{VR}_{\epsilon_i}(P)\}_{\epsilon_i \in \mathcal{E}} \quad (6.11)$$

where

$$\mathcal{E} = \{\|\mathbf{x}_i - \mathbf{x}_j\|_2 \mid i, j = 1, \dots, m\} \cup \{\infty\} \quad (6.12)$$

An example of VR filtration is shown in Figure 6.2(b).

Now, a topological feature will appear at a certain index of the filtration (birth) and disappear at another index (death) or possibly persist even as $\tau \rightarrow \infty$. The p -holes that persist for the longer intervals $\tau_{\text{death}} - \tau_{\text{birth}}$ are more likely to be true topological features, while those that appear and disappear quickly are more likely induced by noise. This characteristic is named *persistence*.

To study the persistence of the topological features we can define a variant of Equation 6.3, the *persistent p -Combinatorial Hodge Laplacian*. Let $\Gamma_i \subset \Gamma_j$ be two ASCs of the same filtration, $i < j$, and let $C_p^{(i)}, C_p^{(j)}$ be the respective chain spaces. Then, we define the subspace

$$C_p^{(i,j)} = \{c \in C_p^{(i)} \mid \partial_p^{(i)} c \in C_p^{(j)}\}, \quad C_p^{(i,j)} \subseteq C_p^{(i)}. \quad (6.13)$$

Then, the persistent boundary operator is the restriction of the boundary over Γ_i to the defined subspace,

$$\partial_p^{(i,j)} = \partial_p^{(i)}|_{C_p^{(i,j)}}; \quad (6.14)$$

similarly, the persistent variant of the Combinatorial Hodge Laplacian is

$$\mathcal{L}_p^{(i,j)} = (\partial_p^{(i,j)})^\dagger \partial_p^{(i,j)} + \partial_{p+1}^{(i,j)} (\partial_{p+1}^{(i,j)})^\dagger. \quad (6.15)$$

The number of p -holes that persist throughout the interval starting from the Γ_i and ending at Γ_j is denoted as the *persistent Betti number*,

$$\beta_p^{(i,j)} = \dim \ker(\mathcal{L}_p^{(i,j)}). \quad (6.16)$$

Topological features and machine learning

Topological features have been paired with machine learning models for two decades, leading to important results. A recent survey by Hensel, Moor, and Rieck [127] comprehensively explores the methods and their implications. In analogy with the presentation of machine learning tasks on graphs, we can define machine learning tasks on p -simplices (generalization of node-level, edge-level tasks) and tasks on ASCs/filtrations. We will focus on machine learning tasks at the filtration level only.

A relevant issue is to identify the most convenient representations to integrate the topological features with the popular machine learning models. These representations are usually expressed as a function of the *persistence diagrams*, a mathematical object collecting any information about persistent Betti numbers. Let $\mathcal{F} = \{\Gamma_{\epsilon_i}\}_{i=1}^m$ be a Vietoris-Rips filtration, its persistence diagram $\mathcal{P}$ is a multiset with elements in $(\mathbb{R} \times \mathbb{R}) \cup (\mathbb{R} \cup \{\infty\})$; for any point (ϵ_i, ϵ_j) we associate the multiplicity value $\mu_p^{(i,j)}$,

$$\mu_p^{(i,j)} = (\beta_p^{(i,j-1)} - \beta_p^{(i,j)}) - (\beta_p^{(i-1,j-1)} - \beta_p^{(i-1,j)}). \quad (6.17)$$

Informally, $\mu_p^{(i,j)}$ counts how many p -holes that are born at distance ϵ_i and die at ϵ_j . Here, a persistence diagram contains information about one specific order of Betti numbers (p); clearly such an object can be extended to contain multiple ones.

At this point, the persistence diagram can be discretized and vectorized, leading to its use as feature vectors in neural networks or as a feature map. Furthermore, the literature has suggested several techniques to embed the persistence diagram in a function space, leading to a proper infinite-dimensional kernel function. An example of such is the

persistence landscape [128], although its treatment is beyond the scope of this work. A much simpler representation is the Betti curve and is the object of our investigation in the subsequent sections.

6.2 Topological data analysis on a quantum computer

The authors in [31] proposed a quantum algorithm for approximating β_p of a simplicial complex Γ_i . This algorithm, known as LGZ from the initials of its authors, has paved the way for quantum topological data analysis. In this section, we briefly discuss this algorithm and explore both its potential benefits and limitations.

6.2.1 Task

The task of calculating the p -Betti number is generally #P-hard, and remains so even when restricted to its multiplicative approximation. Neither classical nor quantum devices are expected to solve this task efficiently [129]. However, estimating an *additive approximation* of a normalization of Betti numbers is tightly related to a spectral estimation task that has been proven to be DQC1-hard, i.e., hard for a classical device but feasible for the one-clean qubit model of quantum computation. This sub-universal model initializes the quantum state to the maximally mixed state for all but one qubit, which is initialized to $|0\rangle$. This implies a strong separation between quantum and classical capabilities for the task at hand [130].

However, there are some caveats. This separation is the result of reducing the task to the estimation of the low-lying portion of the spectrum of $\mathcal{L}_p$. Firstly, we need to know the spectral gap of such an operator and ensure it is not exponentially small in the number of vertices of the ASC. If that is not the case, we cannot distinguish zero from almost-zero eigenvectors. Secondly, the quantity will not be truly the Betti number, but a normalization of it by a factor of $1/|\Sigma_p|$.

Let Γ be an ASC and $\mathcal{L}_p$ its p -Combinatorial Hodge Laplacian. We denote the following quantity as the *normalized quasi-Betti numbers*,

$$\bar{\beta}_p = \frac{|\{\lambda \in \text{spectrum}(\mathcal{L}_p) \mid \lambda < \tau \in 1/\text{poly}(n)\}|}{|\Sigma_p|}. \quad (6.18)$$

The dependence of $\bar{\beta}_p$ on the ASC Γ is not shown explicitly for clarity of notation. The task here is finding an additive approximation of $\bar{\beta}_p$. This corresponds to the normalized

Betti number if we are guaranteed that τ is smaller than the spectral gap of the operator $\mathcal{L}_p$, and can be solved in polynomial time (in the number of vertices of Γ) on a quantum computer under some mild assumptions.

6.2.2 Overview of the quantum algorithms

The quantum algorithm solving the *normalized quasi-Betti numbers* problem is fundamentally based on the *low-lying spectrum estimation* problem. Let H be a Hermitian operator with dimensionality 2^n and Schatten 2-norm $\|H\| \in O(\text{poly}(n))$. Let $0 \leq a < b \leq 1$ with $b - a \in 1/O(\text{poly}(n))$. Supposing we have oracle access to H , we can define a quantum algorithm that runs in time polynomial in n and calculates an ϵ -additive estimation of

$$\frac{|\{\lambda \in \text{spectrum}(H) \mid a \leq \lambda \leq b\}|}{\dim(H)}. \quad (6.19)$$

This quantity is estimated by loading the uniform mixture of vectors in the computational basis,

$$\rho = \frac{\sum_{j=0}^{2^n-1} |j\rangle\langle j|}{\dim(H)} = \frac{\mathbb{I}_H}{2^n}. \quad (6.20)$$

Note that ρ can be expressed as the linear combination over the vectors of the computational basis as well as any other basis, including the one $\{|\psi_j\rangle\}_{j=0}^{2^n-1}$ of the eigenvectors of H ,

$$\rho = \frac{\mathbb{I}_H}{2^n} = \frac{\sum_{j=0}^{2^n-1} |\psi_j\rangle\langle\psi_j|}{2^n}. \quad (6.21)$$

Then, we can use quantum phase estimation with precision $\tau = b - a$ (which by assumption is not exponentially small) over the unitary $\exp(-iH/\|H\|)$. Let $t = \lceil \log_2 \tau \rceil$ be the number of bits required to encode the eigenvalues with precision τ . By feeding this algorithm with the quantum state ρ , we induce an eigendecomposition of such state in terms of pairs $(\tilde{\lambda}, |\psi\rangle)$ of eigenvalues and eigenvectors,

$$\text{QPE} \left[\exp(-iH/\|H\|) \right] \cdot (|+\rangle\langle+|)^{\otimes t} \otimes \rho \quad (6.22)$$

$$= \text{QPE} \left[\exp(-iH/\|H\|) \right] \cdot (|+\rangle\langle+|)^{\otimes t} \otimes \frac{\sum_{j=0}^{2^n-1} |\psi_j\rangle\langle\psi_j|}{2^n} \quad (6.23)$$

$$= \frac{\sum_{j=0}^{2^n-1}}{2^n} \left(\text{QPE} \left[\exp(-iH/\|H\|) \right] \cdot (|+\rangle\langle+|)^{\otimes t} \otimes |\psi_j\rangle\langle\psi_j| \right) \quad (6.24)$$

$$= \frac{\sum_{j=0}^{2^n-1}}{2^n} (|\tilde{\lambda}_j\rangle\langle\tilde{\lambda}_j| \otimes |\psi_j\rangle\langle\psi_j|) \quad (6.25)$$

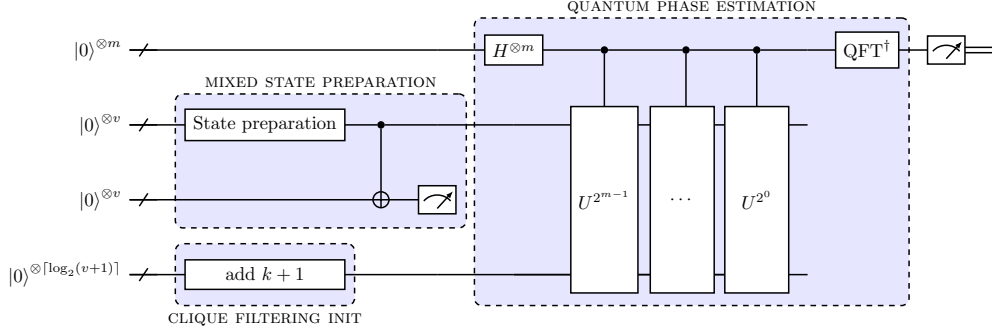


Figure 6.1: Quantum circuit implementing the LGZ algorithm.

Note $\tilde{\lambda}$ is an approximation of the true eigenvalue λ up to the desired precision τ . By measuring the eigenvalue register of the quantum phase estimation, we are sampling *uniformly* from $\text{spectrum}(H)$. The number of samples required to get an estimate with precision ϵ and with probability at least $1 - \mu$ is $O(\epsilon^{-2} \log \mu^{-1})$ (follows directly from Chebyshev's inequality).

Turning this scheme to our task, the estimation of $\bar{\beta}_p$ can be obtained by setting $H = \mathcal{L}_p$, $\rho = \mathbb{I}_{\mathcal{L}_p} / \dim(\mathcal{L}_p)$, $a = 0$, $b = \tau$ (any value that is reasonably small, not exponentially small, and smaller than the spectral gap of $\mathcal{L}_p$). The norm of $\mathcal{L}_p$ is upper bounded by $O(n)$ due to the circle theorem; however, no lower bound on the spectral gap is known.

Here, a caveat is that $\dim(\mathcal{L}_p) = |\Sigma_p| \leq 2^n$, so spanning $\mathbb{I}_{\mathcal{L}_p}$ requires us to sample p -simplices of our ASC and not vectors of the computational basis. This task can limit the performance of the approach as, for certain structures, sampling p -simplices is NP-complete (e.g., for clique complexes and $p \in O(n)$).

The circuit is shown in Fig. 6.1.

6.2.3 Encoding of the p -simplices

The simplest way of defining an encoding for the p -simplices of the ASC $\Gamma = (V, \Sigma)$ with $|V| = n$ is using an n -qubit quantum system. In fact, considering the direct sum of chain spaces,

$$\bigoplus_{p=0}^n C_p \simeq \mathbb{R}^{2^n}, \quad (6.26)$$

any p -chain can be encoded as a vector of the subspace of the Hilbert space of n -qubits having Hamming weight $p + 1$, i.e., $p + 1$ qubits out of n are set. Alternatively, McArdle,

Gilyén, and Berta [131] has proposed a logarithmic encoding using only $O(p \log n)$ qubits. Such an improvement in space is not expected to have any significant speedup in terms of runtime.

6.2.4 Preparation of the uniform mixtures of cliques

Let $\Gamma = (V, \Sigma)$ be the ASC at hand, and $\mathcal{L}_p$ the p -th Combinatorial Hodge Laplacian over Γ . The uniform mixture of p -simplices, denoted as ρ , can be constructed by uniformly sampling p -simplices using a classical procedure and loading the corresponding value into the input register of QPE in the quantum circuit described above.

We can obtain a quadratic speedup if the sampling is performed using Grover's search on a quantum device. Suppose we are given the oracle O that tests the membership of an input bitstring $\sigma \in \{0, 1\}^n$ to the set of p -simplices of Γ . In that case, we can set up a Grover's search using as the diffusion operator the unitary that prepares the Dicke state, i.e., the uniform superposition of computational basis vectors with Hamming weight $p + 1$, instead of the routine for the uniform superposition. The Dicke state preparation is achieved in time $O(pn)$. Such a procedure is efficient whenever the number of p -simplices is large, with $|\Sigma_p|/\binom{n}{p+1} \in 1/O(\text{poly}(n))$. This step can be, in general, NP-complete.

6.2.5 Encoding of the operator

Creating a block encoding for $\mathcal{L}_p$ can be done rather efficiently thanks to the sparsity structure of such an operator. More precisely, its matrix representation has $|\Sigma_p| \leq \binom{n}{p+1} \in O(n^p)$ rows, each row corresponding to a different p -simplex σ , which can be lower-adjacent or upper-adjacent to at most n^2 other p -simplices (follows by direct calculation).

This allows us to define a sparse oracle access. Let O_{row} be the oracle encoding the adjacency list of the nonzero elements of each row,

$$O_{\text{row}} |\ell\rangle |\sigma\rangle = |\ell\rangle |\sigma'\rangle \quad (6.27)$$

with $|\ell\rangle$ indicating the ℓ -th element, $|\sigma\rangle$ indicating the row, and $|\sigma'\rangle$ being the column of the ℓ -th nonzero element in the row σ . The register $|\ell\rangle$ must be large enough to fit the sparsity constant of the operator; in the worst-case scenario, this is $\ell = \lceil \log_2 n^2 \rceil$. Let

O_{entry} be the oracle encoding the value of each nonzero entry,

$$O_{\text{entry}} |0\rangle |\ell\rangle |\sigma\rangle = \left[(\mathcal{L}_p)_{\sigma,\sigma'} |0\rangle + \sqrt{1 - (\mathcal{L}_p)_{\sigma,\sigma'}^2} |1\rangle \right] |\ell\rangle |\sigma\rangle. \quad (6.28)$$

These two oracles are pieces of classical computation. In particular, the row oracle can compute all the possible nonzero locations irrespective of the actual structure of the ASC itself, as they may correspond to a zero entry in the other oracle. The entry oracle sets each (truly nonzero) entry to ± 1 according to an alternating pattern.

Given these, we can construct, following the procedure in [132], an oracle access for $\mathcal{L}_p$. Subsequently, the Hamiltonian simulation of the operator can be performed in linear time (neglecting logarithmic factors) using the quantum signal processing + qubitization routine described in [133]. At this point, the quantum phase estimation can be applied, and we are able to perform the computation in Equation 6.22.

Note that in the literature there is an alternative construction based on the similarity between $\mathcal{L}_p$ and the Hamiltonian of the hardcore fermion model. This allows us to obtain oracle access to $\mathcal{L}_p$ by defining such a Hamiltonian as a composition of twice the Dirac operator (which is n -sparse and not n^2 -sparse) and two projectors Π_{p+1}, Π_p toward the subspace spanned by the $(p+1)$ - and p -simplices, respectively.

6.2.6 Runtime

The runtime of the procedure, ignoring any logarithmic factor, is shown below. Here, we assume the density of p -simplices to be $|\Sigma_p|/\binom{n}{p+1} \in O(n^{-c})$ for some constant c , and the spectral gap of the operator to be $\in O(n^{-s})$ for some constant s . Whether these assumptions hold or not depends on the structure of the ASC at hand. However, if that is not the case, the quantum algorithm will not run in polynomial time in n .

The runtime is:

$$\tilde{O}\left(\begin{array}{l} pn \times n^{c/2} \times \alpha + \\ \beta \times 1 \times n \end{array} \right) \quad \begin{array}{l} \text{Dicke state preparation,} \\ \text{oracles for the quantum search,} \\ \text{cost of each oracle} \end{array} \quad (6.29)$$

$$\quad \begin{array}{l} \text{cost of row and entry oracles,} \\ \text{cost of Hamiltonian simulation for fixed time,} \\ \text{bound on the norm of } \mathcal{L}_p \end{array} \quad (6.30)$$

$$\times n^s \epsilon^{-2} \quad \begin{array}{l} \text{precision of QPE} \\ \text{precision of sampling} \end{array} \quad (6.31)$$

Here, the cost of the oracles is hidden behind the constants α, β . These depend on the ASC at hand, too. For example, the oracle for the quantum search will have a different cost whenever we are considering clique complexes (the cost will be the one checking whether the given input forms a clique and will depend on the number of edges of the skeleton graph) or abstract simplicial trees (the cost will depend on the number of maximal simplices).

Note that there exist variants of the above-presented algorithm with slightly different characteristics and asymptotic runtimes. In particular, Grover's search can be replaced with a fixed-point quantum search to avoid the need of knowing the exact number of oracle calls to apply, the operator can be constructed via the Dirac operator, the QPE can have a window different from the rectangular one obtained via $H^{\otimes t}$ (e.g., the Kaiser window), and the sampling can be quadratically improved as in [134] by turning the problem into an amplitude estimation.

6.2.7 Practical applicability

Despite evidence of a separation between classical and quantum algorithms for Betti number estimation (cf. Gyurik, Cade, and Dunjko [130]), the performance of such approaches in real-world scenarios is still up for debate. A practical class of instances leading to a super-polynomial quantum speedup is based on the independent complex, as shown in [134]. A more general scenario involving the Vietoris-Rips construction has demonstrated a quartic speedup [129]. Furthermore, when the spectral gap of $\mathcal{L}_p$ is constant, the task becomes much easier and can be effectively performed on a classical device [135].

Larger speedups might be obtained by considering different tasks than Betti number estimation. One potential task is spectral entropy estimation [130]. Finding tasks that overcome the weaknesses related to Betti number estimation, such as the efficient sampling of p -simplices, is an area of active research.

6.3 Topological feature-based quantum kernels

We show how it is possible to embed the information extracted from the LGZ quantum algorithm within a kernel function and discuss the implications of such a construction. Our modeling is based on the *Betti curve* introduced in [136].

6.3.1 Betti curves

The persistence diagram and its multi-set format are not trivially compatible with most machine learning models. Their cooperation requires a change of representation. One of the simplest ways to encode some of this information is via the Betti curve. It is defined from a persistence diagram $\mathcal{D}$ as

$$\mathbf{b}(t) = \sum_{(b_j, d_j, \mu_{ij}) \in \mathcal{D}} \omega(b_j, d_j) \cdot \mathbb{1}(t \in [b_j, d_j]). \quad (6.32)$$

Here, ω is an arbitrarily chosen weight function that accounts for the possibility that $\epsilon_j = \infty$. Such an object is known as a *summary statistic*, where summary denotes its non-injectivity: in fact, the information about the multiplicity is lost. As such, it does not permit the tracking of single features, rendering the information at each t essentially non-persistent Betti numbers.

Consider the case of Vietoris-Rips filtrations with $\mathcal{E}$ being the set of distances between points. Then, the Betti curves allow this discrete representation, $\mathbf{b} \in \mathbb{R}^{|\mathcal{E}|}$,

$$\mathbf{b}_j = (\epsilon_{j+1} - \epsilon_j) \times \beta_p(\Sigma_j), \quad (6.33)$$

for $j = 1, \dots, |\mathcal{E} \setminus \{\infty\}|$. Basically, it corresponds to the number of p -holes weighted by their interval of persistence. Clearly, a single Betti curve depends on a single order of homology (p). We can extend this concept to multiple orders via the multi-dimensional Betti curves, $\mathbf{b} \in \mathbb{R}^{P \times |\mathcal{E}|}$,

$$\mathbf{b}_{p,j} = (\epsilon_{j+1} - \epsilon_j) \times \beta_p(\Sigma_j), \quad (6.34)$$

for $p = 0, \dots, P$ cutoff degree. Such a representation induces a kernel function,

$$\kappa(\mathcal{F}, \mathcal{F}') = \exp\left(-c \left\| \mathbf{b}^{\mathcal{F}} - \mathbf{b}^{\mathcal{F}'} \right\|_2^2\right). \quad (6.35)$$

This mapping is obtained by vectorizing the Betti curves of the two filtrations, calculating their norm, and processing their value according to a negative exponential for any constant $c \in \mathbb{R}_{\geq 0}$.

6.3.2 Betti curves with normalized quasi-Betti numbers

The Betti curves are an interesting representation for the information we can extract with the quantum algorithm introduced. The main adjustment we need to consider is the use of normalized quasi-Betti numbers (Equation 6.18) instead of Betti numbers (Equation 6.6). This also implies the Betti curve will be parameterized by the cutoff degree P but also the threshold for the spectral gap we use to distinguish low-energy and high-energy eigenvectors of $\mathcal{L}_p$ as well as the additive precision ε of the estimation.

This model has some practical implications in solving machine learning tasks on filtrations.

Firstly, our variant of the Betti curves not using persistent features allows us to immediately rely on LGZ to retrieve the components of such an object. Although approaches to efficiently encode the persistent Combinatorial Hodge Laplacian have been studied by Hayakawa [137], in a regime of limited resources (e.g., early fault-tolerant quantum computing [138]), such a characteristic can be beneficial.

Secondly, the use of normalized quasi-Betti numbers instead of Betti numbers shifts the focus from the absolute number of p -holes to the density of p -holes in the structure.² This can drastically change the behavior of kernel machines. For example, let $\mathcal{F}, \mathcal{F}'$ be two Vietoris-Rips filtrations over n, n' points with $n \ll n'$. We can create pathological examples in which the "smaller" filtration has much fewer Betti numbers than the "larger" one, but has the same density. In this case, the kernel over Betti curves will regard them as different, while the one based on normalized quasi-Betti numbers will regard them as similar. Whether this behavior is favorable or penalizing depends on the task at hand.

A third aspect to consider is that the quantum kernel defined in Equation 6.35 is substantially different from other approaches introduced. In particular, NISQ-ready quantum kernels encode the data as a vector of the quantum Hilbert space, while here, we use the quantum algorithm to extract some feature and embed it in a classical, relatively small Hilbert space. We think the latter approach is ideal for the task. This can be understood by comparing our approach with the Graph Edit Distance. The

²Retrieving the normalization constant is highly non-trivial and can be done, in principle, with quantum counting.

most computationally challenging aspect of GED was that to have a proper distance, we must compare the representation of one graph against the representation of all possible permutations of vertices of the second graph. Here, we exploit the fact that the eigenvalues of an operator are invariant to the permutation of its columns, leading to the feasibility of the approach (i.e., not having to test all permutations of a graph). Obtaining a permutation-invariant feature map for both graphs and ASCs beyond spectral approaches is a paramount open problem, and until that is solved, it is more computationally beneficial to extract the spectrum and analyze it rather than trying to encode a vector representation of the graph. The downside of such an approach is the existence of isospectral but not isomorphic structures.

A fourth aspect to consider is that we can definitively analyze the two "ends" of a filtration: when the p -simplices are so few they can be classically encoded, or when they are so many that their sampling is efficient. It is an open problem to determine the feasibility in intermediate density regimes.

6.4 A toy example of topological kernel over normalized quasi-Betti numbers

The quantum algorithms for Betti number estimations still have a long path of open problems to be truly considered a successful approach to quantum advantage. On the theoretical side, we are unable to efficiently load p -simplices, limiting us to the two extremes of the filtration. On the applicative side, a class of instances that are classically hard and quantumly easy to analyze has been shown in [134], while in general, early stage experimental evidence suggests the speedup might be around quartic in real-case scenarios. On the practical side, these algorithms require fault-tolerant quantum computation, and it is still unclear whether this hardware might be available or if these algorithms can be adapted to the early-fault tolerant regime [138].

In this section, we propose a minimal toy problem that might be used to benchmark QPUs over our kernel machine. We intend to use this benchmark to test whether substituting components of the above-described quantum algorithm with more NISQ-friendly alternatives might result in an approach runnable on the current generation of quantum devices. We require the benchmark to be feasible to solve by simulating the quantum computation on classical hardware.

The choice of the task has been shape detection, a field that, among others, has

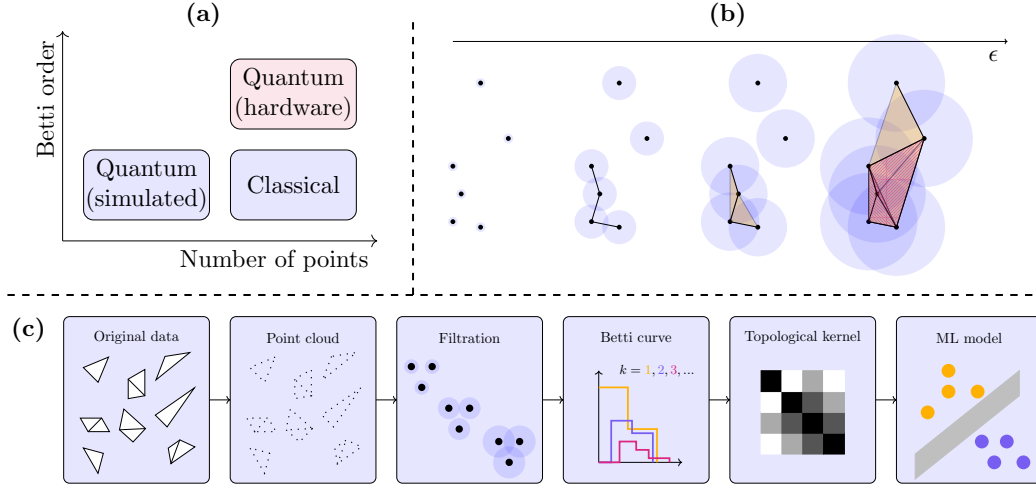


Figure 6.2: Characteristics of Betti number estimation on a quantum device and its relevance to our experimental setup. **(a)** Quantum speedup is anticipated for high Betti numbers, potentially depending on n (e.g., $p \in O(\log n)$), and for structures large enough to pose classical computational challenges. **(b)** Vietoris-Rips Filtration at different threshold values. **(c)** Experimental setup of our toy example: geometric shapes are generated and sampled to form point clouds. The associated Vietoris-Rips (VR) filtration is analyzed using Betti curves to define the topological kernel. Subsequently, the kernel is evaluated using an SVM against the defined dataset.

benefited the most from these approaches. We generate a synthetic dataset of point clouds with a modest number of vertices. We ensure these shapes lead to Vietoris-Rips filtrations with substantially different topological features. The experiment is schematized in Figure 6.2.

6.4.1 Setup

We generated a dataset of geometric shapes in $\mathbb{R}^2$. These shapes can either be the boundary of a triangle or a "sliced" quadrangle, as depicted in Figure 6.2(c). From the geometric description of these shapes, we sampled an arbitrary number of points uniformly around their boundary. The task is to classify the cloud of points according to their original shape. Note that this setting is a simplified version of real-world tasks, such as recognizing digits in the MNIST dataset.

The geometric shapes were defined by applying affine transformations (rotation between 0 and 2π , skew between 0 and $\pi/4$) to a basis shape. For the triangle, the basis shape consists of the segments $\{(x_0, x_1); (x_1, x_2); (x_2, x_0)\}$, and for the

sliced quadrangle, it consists of $\{(x_0, x_1); (x_1, x_2); (x_2, x_0); (x_1, x_3); (x_2, x_3)\}$. Here, $x_0 = (0, 0)$, $x_1 = (1, 0)$, $x_2 = (0, 1)$, $x_3 = (1, 1)$, though any set of non-aligned points would have worked similarly.

We created a balanced dataset of 100 geometric figures, randomly split into training (80%) and testing (20%) sets, ensuring each set is balanced. We generated several derived datasets $\mathcal{D}_5, \dots, \mathcal{D}_{20}$ by sampling $m = 5, \dots, 20$ points from each geometric shape. Additionally, we created a smaller dataset $\mathcal{D}_q$ composed of 5-point clouds derived from 20 geometric shapes. We used these datasets to address two tasks: 1) evaluating the potential of the kernel based on topological features to outperform more traditional, Euclidean-based kernels, and 2) assessing the behavior of the kernel based on topological features when executed or simulated.

Performance evaluation We solved tasks involving datasets $\mathcal{D}_5, \dots, \mathcal{D}_{20}$ using an SVM equipped with the following kernels:

- Gaussian kernel (hyperparameters $\gamma = 1e-4, 1e-3, 1e-2, 1e-1, 1$);
- Laplacian kernel (hyperparameters $\gamma = 1e-4, 1e-3, 1e-2, 1e-1, 1$);
- Polynomial kernel (degrees $d = 1, 2, 3, 4, 5$);
- Topology-based kernel (Equation 6.35, $\gamma = 1.0, p = 2$).

All kernels were calculated classically and exactly.

Behavior in a quantum system We addressed the task involving dataset $\mathcal{D}_q$ by computing it both classically (exactly) and by simulating the quantum algorithm on a classical device. The quantum algorithm for Betti number estimation introduced earlier requires primitives available on fault-tolerant quantum hardware only. To explore whether alternative approaches feasible on NISQ devices exist, we tested various Hamiltonian simulation techniques (with worse asymptotic scaling but better performance with few qubits). The Hamiltonian simulation step is crucial compared to other phases: sampling of p -simplices can be done classically, while quantum phase estimation is best suited for an early fault-tolerant setting using robust quantum phase estimation. We tested Hamiltonian simulation techniques such as Trotter (with a Trotter number spanning 2, 3, 4) and qDrift [139] (with a repetition number of 1, 2, 5, 10), alongside varying numbers of shots $\{1, 2, 4, 8, 16, \infty\} \times 10^2$. The metric used to evaluate the error between

exact and approximated kernel values (due to Hamiltonian simulation and sampling errors) was the root mean square error (RMSE):

$$\text{RMSE}(K_{\text{exact}}, K_{\text{approx}}) = \frac{\sqrt{\sum_{i,j=1}^m (K_{\text{exact}})_{ij} - (K_{\text{approx}})_{ij}}}{m}. \quad (6.36)$$

6.4.2 Results

The comparison of kernel performance accuracy is shown in Figure 6.3(a). As expected, shape recognition is a task where topological features, particularly the kernel based on them, significantly outperform traditional Euclidean-based kernels. The topology-based kernel demonstrated a consistent improvement in performance with an increasing number of sampled points from the point cloud. The second-best performance was achieved by the RBF kernel with $\gamma = 0.1$; however, its oscillating accuracy with respect to the number of sampled points raises questions about its overall efficacy. Notably, the use of normalized quasi-Betti numbers did not penalize the results in this task.

Figure 6.3(b) illustrates the comparison between exact kernel values (calculated classically) and those estimated by simulating the quantum algorithm under different Hamiltonian simulation tasks and numbers of shots. The Trotter-based approach resulted in negligible errors, aligning with considerations that, despite worse asymptotic scaling, Trotter methods are effective with few qubits. In contrast, stochastic Hamiltonian simulation approaches like qDrift showed less success and are not recommended for this task.

6.5 Potential advantage of quantum kernels on combinatorial structures

In the previous and current chapters, we have explored two approaches to defining a quantum kernel over combinatorial structures: one via the solution of an optimization problem and the other by estimating the spectral properties of an operator.

The most challenging aspect of calculating a kernel over combinatorial structures is ensuring permutation invariance: we desire the kernel function to be independent of the permutation of its vertices. For the Graph Edit Distance (GED) kernel, achieving permutation invariance involves an optimization process that finds the minimum value across all possible permutations of vertices. This process is computationally intensive,

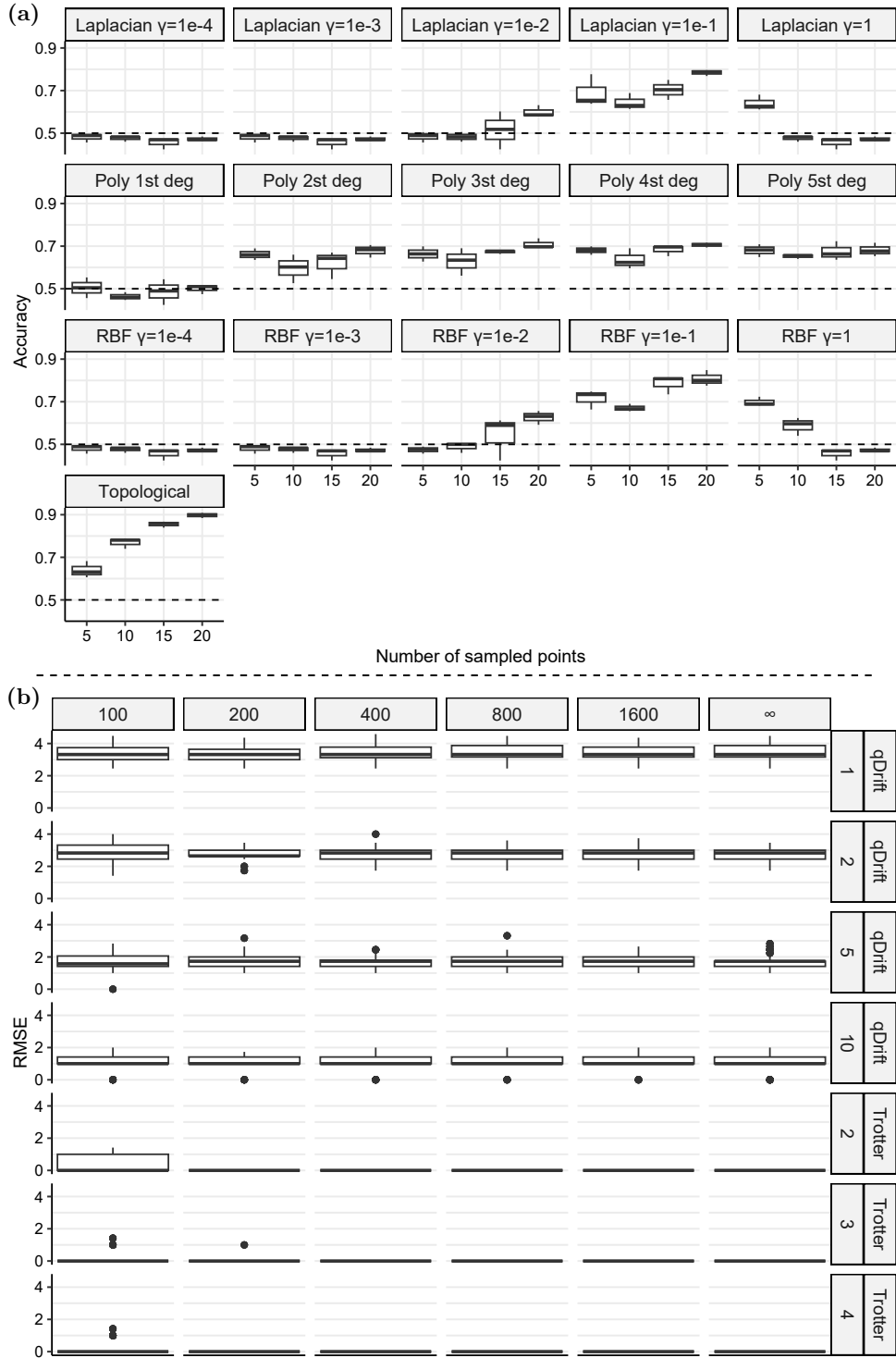


Figure 6.3: Results of the toy example. **(a)** Accuracy of the various kernels (y) with respect to the number of points sampled from the geometric shapes (x). **(b)** Root Mean Square Error (RMSE) between the exactly calculated topological kernel and its approximation using weaker Hamiltonian simulation techniques and quantum circuit shots.

and exact calculation is generally infeasible. The use of quantum algorithms does not currently make this optimization feasible. In contrast, for the "topological" kernel, permutation invariance is inherent because the eigenvalues of an operator are invariant under any permutation of its columns, thus preserving the spectrum's properties. However, this method may fail to distinguish isospectral but not isomorphic objects. Despite this drawback, the computational cost remains feasible.

This latter consideration is crucial for justifying our approach. Although the kernel proposed in Equation 6.35 is not a traditional quantum kernel—in that it extracts information and encodes it into a classical vector rather than exploiting the Hilbert space of the quantum system—this choice is ideal because relying on a traditional quantum kernel (and thus the vectors in the Hilbert space) might compromise permutation invariance.

Furthermore, it's important to note that the advantage of using ASCs is not necessarily tied to their specific structure. Such an approach could potentially be used by substituting ASCs with a large, sparse graph whose sparsity structure is efficiently determined.

These considerations apply to tasks at both the graph-level and ASC-level. For node-level tasks where the combinatorial structure is fixed and permutation invariance is not needed, other approaches may also be feasible.

This page has been intentionally left blank.

Chapter 7

Application of quantum techniques for combinatorial optimization to epistasis detection

A paramount task in systems biology and genetics is performing *genome-wide association studies*. This task aims to identify and statistically correlate genetic mutations, the so-called *single nucleotide polymorphisms* (SNPs), that are associated with a disease or a phenotype, i.e., an observable characteristic of an organism. Many studies have focused on explaining the heritability of phenotypes by associating them with single SNPs; however, such a correlation is not always statistically significant. The detection of *epistasis*, i.e., interactions between genes, is an important research direction. The core idea is to investigate the correlation between SNPs occurring simultaneously in individuals with single phenotypes [140]. No cases of human epistasis have yet been discovered, partly due to the challenging computational problem of finding the correct group of SNPs among the combinatorially many ones over billions of SNPs.

Approaches to tackle epistasis detection require the development of several aspects; these include a statistical model of epistasis, a biologically plausible model for SNP-SNP interaction that is sparse enough to render the task feasible, and an optimization engine that samples the best candidate solutions from the SNP-SNP network. Such approaches are all heuristic in nature. In this chapter, we investigate the role of quantum computing as a tool for improving the optimization part of this task. We propose a hybrid method to navigate the SNP-SNP interaction network: first, a series of 'seeds' or initial SNPs are sampled from the network, and second, a stochastic optimization technique visits

the neighborhood of each seed to sample the best candidate solutions. The core idea is to have a relatively inexpensive approach to sampling a few promising seeds from the network, followed by a much more expensive approach to evaluating the (optimized) candidate solutions in the seed neighborhood against actual data. The first part, the seeding, is a quadratic optimization problem due to navigating a network of pairwise interactions only, while the second part, the search, is expected to be much more complex as we are dealing with larger interaction sizes. We have implemented *quepistasis*, an optimization tool that models the seeding phase via a QUBO and constructs and analyzes instances of epistatic interactions on several kinds of quantum hardware. We can break down large instances using a classical community detection technique. We validate our seeding routine against random seed selection and other state-of-the-art classical optimization approaches. We find that the few satisfactory seeds sampled from the quantum approach have superior performance compared to the many unsatisfactory choices of the random techniques, as well as competitive performance with the other classical techniques. Our scaling analysis suggests a quantum advantage over larger instance sizes and larger quantum hardware under the assumption that computation does not degrade exponentially with the instance size.

In Section 7.1, we introduce the problem of epistasis detection and how to tackle it using a SNP-SNP interaction network approach. In Section 7.2, we discuss the optimization engine used to solve the task. In Section 7.3, we detail the experimental evaluation setup we have used to validate our approach. In Section 7.4, we show and comment on the results of our investigation. This chapter is based on [8].

7.1 Tackling epistasis detection

Epistasis detection presents several challenges.

1. *Choice of a statistical model*: There is no consensus on the choice of a statistical model that can best characterize epistatic behavior. Some of the choices include:
2. *Interpretability of the results*: It is unclear if, whenever an epistatic interaction is found, it is truly due to a biological process or if it is only a statistical artifact.
3. *Large search space*: Enumerating the combinatorially many groups of SNPs is unfeasible even for small group sizes, as the number of SNPs is in the order of billions.

Nevertheless, some approaches have already tried to solve the problem, most notably the software Potpourri [141], LinDen [142], PoCos [143], MACOED [144], and BiologicalEpistasis [145]. These are all limited to pairwise interactions only. Our proposal aims to exhaustively try all the statistical models known in the literature so far and is based on an algorithmic approach that navigates an SNP-SNP network, guaranteeing interpretability as built upon relevant biological information (e.g., protein interactions associated with such SNPs) while being able to manage much larger interaction sizes than pairwise ones.

Our approach has led to the software NeEDL, an acronym for *Network-based Epistasis Detection via Local search*. We can exploit statistical epistasis models; see the ones in [146]. The interpretability of the results is obtained by prioritizing solutions derived from the biologically informed SNP-SNP network. Within such a network, we use a two-phase algorithm: first, we look for cheap initial points in this network; secondly, we use a simulated annealing-based algorithm in the neighborhood of such solutions to find the locally optimal group of SNPs according to the fixed statistical model. Looking uniquely at the dense subgraph in the SNP-SNP network, which has also been sparsified as much as possible thanks to the encoding of biological information, renders the search space much more feasible to explore despite still being unable to enumerate all possible candidates. We find the first phase crucial, benefiting from returning only a few initial candidates of good quality. This is because the first seeding phase is meant to be computationally cheaper, informed by pairwise interactions only, and thus much faster than the test in the subsequent phase. Therefore, having only a few good initial candidates substantially influences the runtime. For this reason, we have employed a quantum algorithm to improve the first phase of the algorithm.

7.2 Functioning of NeEDL

In this section, we detail the functioning of the many components of our software, NeEDL.

7.2.1 Epistasis models

We model epistasis using the framework in Blumenthal et al. [146]. Let S be the set of SNPs with cardinality $|S| = s$, P the set of phenotypes with cardinality $|P| = p$, and I the set of individuals or patients with cardinality $|I| = n$. The dataset is defined

by a pair $(G, \mathbf{y})$, where G is the genotype matrix and $\mathbf{y}$ is the phenotype vector. The genotype matrix $G \in \{0, 1, 2\}^{n \times s}$ has entries valued 0, 1, 2 representing the minor alleles frequency. The phenotype vector $\mathbf{y} \in P^n$ has entries valued as any element in the discrete set of possible phenotypes, although usually it is the set $\{0, 1\}$ representing case or control data.

The *epistatic model* is a pair $(f_{G,\mathbf{y}}, \sigma)$, where $f_{G,\mathbf{y}} : \wp(S) \rightarrow \mathbb{R}$ maps a subset of SNPs to a real value, and $\sigma \in \{\text{MAX}, \text{MIN}\}$ indicates for $\sigma = \text{MAX}$ that $f_{G,\mathbf{y}}$ is a fitness function to be maximized to obtain the optimal result and indicates for $\sigma = \text{MIN}$ that $f_{G,\mathbf{y}}$ is a cost function to be minimized. Intuitively, $f_{G,\mathbf{y}}$ quantifies the statistical evidence for a group of SNPs to experience epistatic interaction. Such a model has been proposed as a generalization of the many approaches tested in the literature, including those based on the p-value of the chi-square test [147], Bayesian score (e.g., K2 [147]), multi-factor reduction [148], polynomial regression [149], and maximum likelihood model [146]. For simplicity, we assume throughout the document that $\sigma = \text{MIN}$.

The problem of epistasis detection can be formalized as finding the group of SNPs that minimize the cost of f ,

$$S^* = \arg \min_{\Sigma \subseteq S} f_{G,\mathbf{y}}(\Sigma), \quad (7.1)$$

possibly under the constraint $l \leq |\Sigma| \leq u$, l, u being the lower and upper bounds on the dimensionality of the resulting SNPs group. This rather general optimization problem is unfeasible to work with; therefore, we insert additional constraints that will drastically reduce the search space.

Let $\mathcal{N} = (S, E)$ be the SNP-SNP network, a graph defined over S as the set of vertices and E as the set of edges, where $(s_1, s_2) \in S \times S$ is an edge whenever we have prior biological knowledge that the two SNPs can interact. The details regarding the construction of this network are shown in the subsequent section. The optimization problem can be rephrased by enforcing that each group of SNPs Σ induces a connected subgraph $\mathcal{N}[\Sigma]$ of $\mathcal{N}$,

$$S^* = \arg \min_{\Sigma \subseteq S, \mathcal{N}[\Sigma] \text{ connected}} f_{G,\mathbf{y}}(\Sigma). \quad (7.2)$$

7.2.2 Construction of the SNP-SNP interaction network

The SNP-SNP network is constructed according to the relationships between SNPs, genes, and proteins. Let S be the set of SNPs and G the set of genes. We introduce a many-to-many SNPs-to-genes mapping $\pi \subseteq S \times G$, which is obtained from the free archive *dbSNP* [150]. We also introduce a network of genes $\mathcal{N}_G = (G, E_G)$ such that two genes are connected whenever the proteins that encode such genes can experience physical interaction, i.e., no fewer than two studies have concluded as such experimentally via, e.g., affinity purification-mass spectrometry or yeast-2-hybrid. The physical interaction between proteins has been obtained from the software *BioGRID* [151]. Our software allows us to easily replace *dbSNP* and *BioGRID* with other archives.

The network of SNPs $\mathcal{N}$ is built as follows. Let $s \in S$ be a SNP and $\pi[s] = \{g \mid (s, g) \in \pi\}$ the genes associated with s according to the relation π . Then $(s_1, s_2) \in S \times S$ form an edge of $\mathcal{N}$ if any of these conditions hold:

- they share a common gene, $\pi[s_1] \cap \pi[s_2] \neq \emptyset$;
- they are related to a common SNP s' with which they both share a common gene, $\pi[s_1] \cap \pi[s'] \neq \emptyset$ and $\pi[s_2] \cap \pi[s'] \neq \emptyset$;
- they are mapped to interacting genes, $\exists g_1 \in \pi[s_1], g_2 \in \pi[s_2]$ such that there is a path from g_1 to g_2 in $\mathcal{N}_G$.

Note that SNPs having $\pi[s] = \emptyset$ will not be present in any solution, further reducing the search space.

7.2.3 First phase: seeding

The first phase of the detection of epistatic interaction requires us to decide where to start navigating the SNP-SNP network; this phase will be followed by a local search in the neighborhood of the initial vertex or group of vertices. We call the first phase *seeding*. The trivial way of performing the seeding is to choose many randomly sampled vertices in the network and pass them to the second, '*searching*' phase. Clearly, to have a meaningful searching phase, we will need a lot of seeds. Such a technique is the default choice in NeEDL.

The searching phase will be a rather computationally expensive optimization problem. This is because we will take into account interactions between vertices of arbitrary degree, and not pairwise ones only (this characteristic most differentiates the algorithm

in NeEDL from alternative software). Therefore, it is reasonable to investigate whether sampling for free many seeds that have to undergo an expensive optimization process is more or less convenient than sampling fewer, 'heuristically good' seeds obtained via a cheap optimization process and passing these fewer ones to the expensive optimization process.

To address this, we prepare a simpler optimization process that samples from the SNP-SNP network's densely connected groups of SNPs. Such an optimization is based on pairwise interactions only and is thus intrinsically cheaper. We model this problem as a QUBO optimization. This formulation is native for many classical and quantum algorithms, making it a natural choice for analyzing the impact of using quantum hardware.

Informed sampling of seeds via QUBO optimization

We model sampling informed seeds as a variant of max-clique that heuristically samples densely connected groups of SNPs in $\mathcal{N}$ and reduce such a problem to a QUBO formulation. Let $\mathcal{N}_S = (S, S \times S, w)$ be a newly constructed SNP-SNP weighted network (different from $\mathcal{N}$) together with a weighting function $w : S \times S \rightarrow \mathbb{R}_{\geq 0}$ assigning a positive score to each edge (the larger the score, the stronger the interaction between the corresponding SNPs of the edge). Such a metric is usually a linear combination of statistical information and biological prior,

$$w(s_1, s_2) = \nu \times f_{\text{stat}}(s_1, s_2) + (1 - \nu) \times \delta_{(s_1, s_2) \in \mathcal{N}}. \quad (7.3)$$

The statistical information f_{stat} can be any of the metrics defined in the previous section, e.g., the p-value of the chi-square test.

The seeding problem is then formulated as the N *max-weighted k-cliques*, i.e., the problem of finding N different SNP groups whose pairwise weights within the same group are maximized, under the constraint of each solution having cardinality K . Such a formulation is defined over a set of $N \times |S|$ binary variables $x_{i\ell}$, to be interpreted as $x_{i\ell} = 1$ for the i -th SNP present in the ℓ -th solution, $i \in \{1, \dots, |S|\}$ and $\ell \in \{1, \dots, N\}$. The formulation itself is:

$$Q = \sum_{\ell=1}^N \left[\lambda_0 \left(\sum_i x_{i\ell} - K \right) - \lambda_1 \sum_{(i,j) \notin E} w_{i,j} x_{i\ell} x_{j\ell} \right] + \lambda_2 \sum_{\ell=1}^N \sum_{m=\ell+1}^N \sum_{i=1}^{|V|} x_{i\ell} x_{im} \quad (7.4)$$

where

- the term $\lambda_0(\sum_i x_{i\ell} - K)$ penalizes groups of SNPs of the wrong size, i.e., different from K . The penalty is weighted by the hyper-parameter $\lambda_0 > 0$ that can be chosen to allow or prohibit small deviations in the group cardinality of feasible solutions;
- the term $-\lambda_1 \sum_{1 \leq i, j \leq |V|} w_{i,j} x_{i\ell} x_{j\ell}$ penalizes groups of SNPs with the largest weight. The term is controlled by the hyper-parameter $\lambda_1 > 1$;
- the term $\lambda_2 \sum_{\ell=1}^N \sum_{m=\ell+1}^N \sum_{i=1}^{|V|} x_{i\ell} x_{im}$ penalizes largely overlapping solutions. The term is controlled by the hyper-parameter $\lambda_2 > 2$.

In the simplest case, we can set $N = 1$ and the optimization will return a single candidate solution. This is ideal when computational resources are scarce.

We have designed a separate module named *quepistasis* that is in charge of constructing the QUBO instances and running them over classical and quantum algorithms. The input of the module is a pairwise correlation matrix between SNPs (matricial form of Equation 7.3) and the output is one or more sets of SNPs. The classical solvers implemented are 1) thermal/simulated annealing and 2) parallel tempering; the quantum approaches implemented are 3) quantum annealing on D-Wave devices and 4) QAOA on IBM devices. The structure of the module is schematized in Figure 7.1. Note that the module is supported by a community detection tool that splits any arbitrarily large instance of SNPs into smaller chunks whenever necessary, e.g., when dealing with smaller quantum devices. The nature of the community detection tool is heuristic, as the algorithm we have implemented is Leiden [152]. This further step can be thought of as a further layer of initial discovery of a cheap solution to warm start the seeding process, rendering the overall approach a pipeline of algorithms that, at each phase, discover fewer and better solutions based on an increasingly higher order of interaction between SNPs.

7.2.4 Second phase: searching

The searching phase aims to find a solution for Equation 7.2 (any epistatic model in [146] is supported). Such a problem is solved heuristically using the simulated annealing algorithm with the seeds from the previous phase as initial solutions. The actual algorithm is a stochastic adaptation of a graph edit distance [153] implemented in GEDLIB [126, 154]. The main advantage of using simulated annealing in this phase is

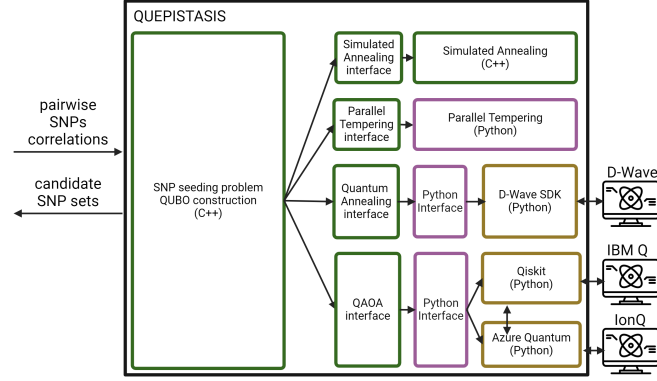


Figure 7.1: Schematization of *quepistasis* module. The module receives a matrix that contains pairwise relations between SNPs. This matrix is then processed by a preliminary software layer to create a QUBO formulation using the input data. Subsequently, this QUBO instance can be resolved either through the utilization of a thermal annealing algorithm on a classical device or by employing one of the existing quantum devices such as D-Wave, IBM Quantum devices, or the devices accessible via the Azure Quantum cloud, which encompasses IonQ devices. The result consists of one or more sets of Single Nucleotide Polymorphisms (SNPs) that will serve as starting points for the local search process.

its trivial parallelization and the possibility to seamlessly interrupt the process before convergence.

We start from the candidate SNP set S_0 , which is one of the initial solutions returned by the seeding phase. Iteratively, we calculate S'_{j+1} as the group of SNPs locally reachable from S_j that differs from the latter candidate by exactly one SNP. We enumerate all the possible subsets of cardinality between a lower bound l and an upper bound u , and keep the set with the highest score. The new solution S_{j+1} will correspond to either S'_{j+1} or S_j ; the former is accepted with certainty if it has a higher score, and accepted probabilistically if it has a lower score, with probability depending on the difference between scores. The process continues until convergence or until a threshold number of iterations is reached.

7.3 Experimental evaluation

In this section, we show the details of our benchmark. The purpose of the current and subsequent section is to identify whether the 'informed' seeding method leads to an end-to-end speedup compared to the randomly chosen seeds and whether the informed

seeding, when solved using quantum algorithms as a heuristic, will lead to a speedup or a projection of speedup in the future.

Datasets

The benchmark of NeEDL is based on eight datasets, but only a few of them have been used to evaluate the quantum algorithms. The datasets are associated with eight different diseases: Late-Onset Alzheimer’s Disease (LOAD), Bipolar Disorder (BD), Coronary Artery Disease (CAD), Diabetes Type 1 (T1D), Diabetes Type 2 (T2D), Hypertension (HT), Inflammatory Bowel Disease (IBD), and Rheumatoid Arthritis (RA) (see Section Data Availability for links to the databases). These were provided by either the TGen consortium [155, 156] or the WTCCC consortium [157].

For the benchmark of the quantum algorithms, we have focused on the datasets BD, RA, and T2D only. The limited quantum resources prohibit us from directly analyzing the whole dataset. Therefore, we have subsampled for each disease, three sub-datasets of 100, 500, and 1000 SNPs, respectively.

Configuration of the optimization algorithms for the seeding phase

Regarding the optimization algorithms, the configurations are the following:

- *Thermal/simulated annealing*: We have used the implementation in Ocean SDK. The number of samples is 10^3 , the number of sweeps (steps) is 10^5 , and the annealing schedule is geometric, $T(i) = T(0) \cdot \alpha^i$. The initial values of the optimization are chosen automatically according to the procedure implemented in Ocean SDK.
- *Parallel tempering*: We have implemented the algorithm in [158, Section 2]. We initialize $M = 8$ chains in parallel, and used for the temperature a geometric annealing schedule of $\{\beta_i\}_{i=1}^8 = \{1.000, 0.518, 0.268, 0.139, 0.072, 0.037, 0.019, 0.01\}$.
- *Quantum annealing*: We have used D-Wave Ocean SDK as a framework to communicate with the quantum hardware, which is D-Wave Advantage 6.1. The number of samples is 10^3 , and the annealing schedule is divided between a forward annealing (proceed linearly for $1ms$ to 50% of the process, pause $1ms$, proceed to completion for $1ms$) and a backward annealing (proceed linearly for $1ms$ to 50% of the process, pause $1ms$, proceed to completion for $1ms$).

- *QAOA (noiseless)*: We have used a noiseless optimizer in the IBM Qiskit framework with 1024 shots drawn per each execution of the quantum circuit. The QAOA ansatz has 10 layers, with random initialization of each parameter. The algorithm has been iterated 10 times using the stochastic gradient descent algorithm ADAM (the low number of iterations matches the ones done on the IBM QPU, whose availability of hardware is limited).
- *QAOA (IBM QPU)*: We have used IBM Perth and Lagos devices. In such cases, we have repeated the QAOA ansatz once, and trained for 10, 5, and 1 number of epochs for datasets of 100, 500, and 1000 SNPs, respectively. The number of shots is 128.
- *QAOA (IonQ QPU)*: NeEDL allows for the use of any backend from the Microsoft Azure Quantum platform and has been tested to work with IonQ QPUs. However, due to the limited access to such QPUs, we have not benchmarked this device against our dataset.

Configuration of the QUBO for the seeding phase

Whenever we set NeEDL to draw random seeds, no configuration is needed. In contrast, when we want to use the optimization procedure, we have to first configure the creation of the QUBO instances. In particular the choice of the hyper-parameters K (desired SNP group size, in our case always set to 5), λ_0 , λ_1 , λ_2 (weights of the terms in the QUBO), and ν (weight of statistical vs biological prior). We also need to specify the maximum allowed dimensionality of the subproblem to be fed directly to the optimization engine, while larger instances have to be split using the community detection tool.

We remark that the number of variables, $N \times |S|$ (or simply $|S|$ when sampling a single k -clique), also corresponds to the number of qubits. However, this value is decoupled from the size of the full instance under consideration, which can be substantially larger than what a quantum computer can handle. This is achieved by selecting a sufficiently large portion of the dataset—specifically, a subset of the SNP-SNP network—and decomposing it into smaller instances that are suitable for the chosen QUBO solver. The decomposition is performed using a community detection algorithm, resulting in sub-instances that can be tackled independently as separate QUBO problems. The size of each sub-instance depends solely on the capabilities of the selected QUBO solver. When executed on classical hardware or quantum annealers, relatively large sub-instances can

be handled. However, when using QAOA on IBM quantum hardware, the instance sizes must be significantly smaller, and in some cases, can even be solved exactly via exhaustive search, given sufficient time. An extensive discussion of this decomposition approach is provided in Hoffmann et al. [8, Supplementary Material 3].

7.4 Results

We show that, when using quantum computing, fewer but higher-quality candidate SNP sets are returned. This implies that speeding up these high-quality initial solutions results in a more efficient searching phase, which outweighs the overhead introduced by the quantum-computing-based seeding. Thanks to the community detection tool, we can break any dataset into SNP sets of arbitrary size, allowing *quepistasis* to be tested on any kind of dataset. However, we still have to rely on smaller, sub-sampled datasets due to the inability to process large amounts of data on quantum hardware yet, due to, for example, having to send jobs through a large queue and the economical costs involved. As the quantum-based techniques outperform the classical baseline, we can anticipate a potential scaling advantage.

Here we show the results of our experiments. Note that we have run our algorithm for a number of iterations or for a time just enough to match (what we think are) the best results (ever obtained on these datasets), with the exception of the RA dataset with 100 SNPs.

In Figure 7.2, we illustrate the trade-off between using ‘linear NeEDL’ seeding (randomly selected seeds) and informed seeding strategies across various algorithms. Along the x -axis—with the arrow pointing from the origin to the left—we report the seeding time, i.e., the time required to perform community detection on the arbitrarily large SNP instance, solve all resulting QUBO sub-instances independently, and aggregate their outputs in preparation for the next phase. Conversely, along the x -axis with the arrow pointing from the origin to the right, we show the time taken to execute the searching phase, which analyzes higher-order interactions within the neighborhood of each seed in the SNP-SNP network. On the y -axis, we plot the quality of the best solution found – both at the end of the seeding phase (horizontal line to the left of the origin) and at the end of the searching phase (to the right). Solution quality is measured in terms of maximum likelihood score, where lower values indicate better results.

As anticipated, the generation of random seeds is practically free but has the drawback

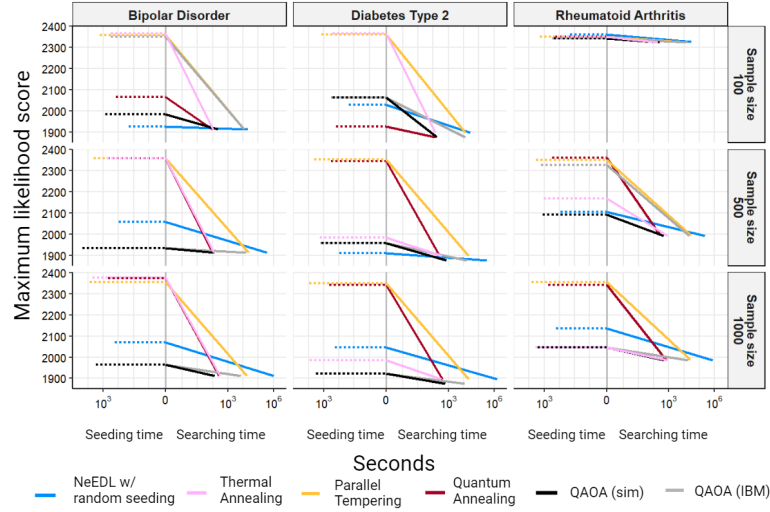


Figure 7.2: Timing performance comparison between Linear NeEDL (randomly chosen seeds) and informed seeding NeEDL with various classical and quantum-based optimization techniques. The negative time represents the duration of the seeding process, while the positive time corresponds to the searching process. Quantum computing enables the generation of a reduced number of high-quality seeds, thereby accelerating the local search phase.

of needing many more seeds in the search phase to obtain meaningful results. This advantage is clearly lost in the subsequent phase, with a $\times 10^3$ factor compared to the informed seeding. For informed seeding, the thermal annealing works best with smaller datasets; QAOA produces very high-quality solutions, but that is mainly due to the community detection tool (the largest instances of QAOA consist of sizes in the order of 1-10 variables, which can even be enumerated). However, the simulated QAOA is much slower than the one executed on IBM devices. For execution on QAOA IBM, we have subtracted the portion spent in the queue from the runtime. The quantum annealing struggles to find high-quality solutions in the seeding phase; however, in the search phase, these are recovered and furthermore, this technique shows the best performances.

In Figure 7.3, we present a potential scaling analysis of performance across all informed seeding algorithms. The x -axis represents the time required for the seeding phase, while the y -axis shows the sample size—i.e., the input size provided to the community detection tool—which remains decoupled from the sizes of the resulting QUBO instances. A complete list of the tested configurations, including the sizes of the QUBO sub-instances, is provided in Hoffmann et al. [8, Supplementary Table 12], and

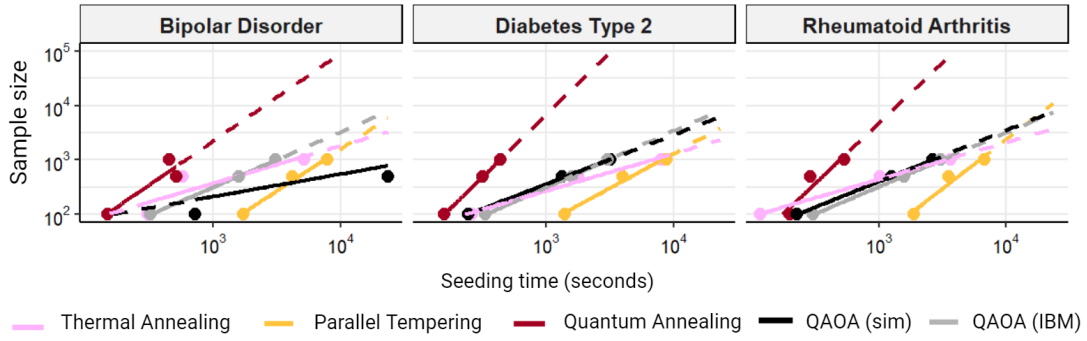


Figure 7.3: (Projected) scaling of the quantum hardware. The various quantum and classical devices utilized for optimization exhibit distinct scaling patterns, with quantum devices (such as Quantum Annealing and QAOA) demonstrating a steeper slope compared to classical hardware (such as thermal annealing and parallel tempering). This suggests a potential advantage in terms of scaling for quantum hardware.

the best-performing values are reported in Hoffmann et al. [8, Supplementary Table 24].

The lines in the plot are obtained via linear interpolation of the runtimes measured for datasets containing 100, 500, and 1000 SNPs. Steeper slopes indicate better scalability. Among the tested approaches, quantum annealing shows the most favorable scaling behavior, owing in part to its ability to handle larger instances directly. In contrast, QAOA is more constrained, as it must rely extensively on community detection to break down larger instances into solvable sub-problems.

Overall, these results suggest a potential scaling advantage for certain quantum approaches. However, we note that while this scaling model appears reasonable for small to moderate dataset sizes, it remains an open question whether the observed trends persist for substantially larger datasets.

This page has been intentionally left blank.

Chapter 8

Open source software in quantum kernels

The last chapter discusses the software tools available for quantum machine learning research, focusing on the software *quask*, explicitly designed for quantum kernels.

Various tools have been made available starting with QuTIP [159], a software dedicated to studying quantum information. A significant breakthrough was achieved with Qiskit [160], which is used to design quantum circuits specifically for IBM quantum hardware. One of the first applicative experiments ever run on a quantum computer and developed with Qiskit has been performed by Havlíček et al. [21], which featured a quantum kernel machine. A series of similar software from significant hardware vendors followed Qiskit. Recently, most of these software tools have been updated to introduce higher-level functionalities aimed explicitly at defining parameterized quantum circuits and variational algorithms in general while offering little support for quantum kernels.

As a result, researchers often need to use quantum software to define quantum circuits and then build the necessary architecture to handle kernels on top of that. This results in a lack of productivity and the introduction of *bugs* [161], which can lead to erroneous results being published in research papers [162, 163, 164]. In contrast, short, high-level, and well-organized code facilitates reproducibility and code reuse while minimizing the time needed for experiment development and bug fixing [165, 166].

To address this gap, we have developed *quask*. Our software is based on Python and built on top of existing quantum libraries to ensure interoperability with existing software. This design facilitates and accelerates the work of quantum kernel researchers through high-level APIs. *Quask* is developed following the “make your code count”

guidelines (Shammah [167]), which have driven the success of some of the most popular open-source packages. Additionally, we provide extensive resources for newcomers to the field. This chapter is based on [3].

8.1 The jungle of open source software in quantum machine learning

The rise in popularity of quantum computing has highlighted the need for more sophisticated programming tools. To address these needs, various programming languages, compilers, and frameworks have been developed. Most of these tools are based on the quantum circuit model and emphasize aspects such as modularity, symbolic manipulation, control, and uncomputation of circuits. Some also allow interfacing with quantum hardware from various vendors via the cloud, facilitating research. The interoperability between these frameworks has also underscored the need for a common intermediate representation. The most popular choice is the OpenQASM format [168], an open-source, assembly-like specification language. Various programming languages, compilers, and frameworks have been developed to address these needs.

In addition to the quantum circuit model, there are several alternative models for quantum computation. These include quantum lambda calculus [169], the quantum Turing machine [170], the adiabatic quantum model [171], measurement-based quantum computation [172], topological quantum computation [173], and the ZX calculus [174].

Table 8.1 compares frameworks that use the quantum circuit model. This section briefly overviews some of these software tools, the ones offering the most advanced machine learning-related features.

8.1.1 PennyLane

PennyLane has been the first framework to offer Quantum Machine Learning capabilities, pioneering a new approach to integrating quantum computing with machine learning. These capabilities include training a parametric quantum circuit, with gradients calculated using advanced techniques such as the parameter-shift rule [192] or the finite difference method. PennyLane facilitates the seamless integration of quantum transformations as layers within neural network models defined in popular machine learning libraries like Keras [193] and PyTorch [194].

Additionally, PennyLane provides various facilities related to quantum kernels, such as automatically creating a fidelity circuit based on a given quantum embedding circuit and offering metrics such as kernel-target alignment.

Remarkably, PennyLane’s sister software, Strawberry Fields, offers similar features tailored for the Continuous Variables model of quantum computing, particularly suited for applications in photonic quantum computing.

8.1.2 Qiskit Machine Learning

Qiskit has designed a sub-package dedicated to machine learning. It contains a class Quantum Kernel that integrates seamlessly with the framework’s lower-level API, enabling users to specify parameters such as the number of samples and error mitigation techniques. The same level of control is provided for Quantum Neural Networks, including gradient calculation.

Certain classes are based on and specialize in the API of the Python library *sklearn*, which can facilitate the development whenever we already use such a library in our pipeline.

8.1.3 QiBO/Qibolab

Qibo [177] is an open-source middleware that offers an API for defining quantum circuits and provides several simulators and plugins to communicate with various types of quantum hardware. It is designed similarly to PennyLane, with some unique features that distinguish it, such as the ability to perform symbolic computations.

The recent development of the sub-module QiboLab [178] has introduced the capability of training variational algorithms. However, as of the time of writing, no features specifically tailored explicitly to quantum kernels are available.

8.1.4 TensorFlow Quantum

TensorFlow Quantum [185] is a library specifically designed to create hybrid quantum-classical ML models. It integrates Google’s quantum computing framework Cirq within TensorFlow [195].

At its core, it focuses on two points: the generation and processing of quantum data, for example, as in the experiment of [44] on the geometric test, and constructing hybrid quantum-classical models. It integrates quantum computing algorithms and logic,

initially developed in Cirq, into the TensorFlow environment. Like *Qiskit Machine Learning* uses Qiskit as its backend, and TensorFlow Quantum uses Cirq for its backend operations while relying on TensorFlow to support higher-level APIs, which is analogous to Qiskit ML's reliance on Scikit-learn.

8.2 “Make your code count” guidelines

When designing new open-source software in quantum computing, careful attention must be paid to several key aspects. Given that researchers will likely be the primary users of the tool, the code must be reproducible. Moreover, it is essential to note that many researchers may only have a basic knowledge of the most popular programming languages. To address these considerations, a set of guidelines known as “Make your code count” [167] has been made available, specifically tailored for the quantum computing community. We briefly recall these guidelines as they have been used to design our tool *quask*.

1. **Choice of the programming language:** A popular choice for the programming language is Python, which has an intuitive syntax, is stable and well-regarded within the scientific community, and features a powerful ecosystem of scientific libraries.
2. **Code version control and distribution:** Maintain an upstream, i.e., the official copy of the code. Then, maintain a copy of such repository on your personal GitHub profile, online, and an offline copy locally on your device. Updating the official copy should be done via the Pull Requests, while bugs can be traced via the New Issue tools in GitHub.
3. **Code conventions:** Write your code using a consistent style in terms of indentation, naming conventions, programming style, and code documentation. The official standards are PEP 8 for code and PEP 257 for documentation strings, with tools available to automatically format the code. Documentation can be paired with notebooks to illustrate the main functionalities.
4. **Testing:** Write unit tests for different software components. Each function should have its testing file. A test script can be created using frameworks like ‘nose’ or ‘pytest’. Pair GitHub with tools such as Travis CI to ensure that every new submission runs the tests from scratch on a clean machine.

5. **Packaging:** The library should be configured as a proper Python package, which requires creating folders and files according to certain conventions. Python allows you to install the library locally, manage external dependencies, and modify the source code on the fly while debugging.
6. **Documentation:** Every function and class should be documented directly within the source code using Python docstrings, following the mentioned format. A documentation website can be generated automatically using the software 'Sphinx'. If present, notebooks detailing some tool functionalities will be part of the documentation. The built website can be hosted on a proprietary site or, for free, on the platform *Read the Docs*.
7. **Distribution:** Once designed as a package, the library can be distributed to all Python users via the 'pip' package manager through the online platform *PyPI*. Before applying to the platform, use the *Test PyPI* platform to ensure everything works correctly. The *Anaconda* package manager offers similar functionalities.
8. **Interactive usage:** The notebooks composing the documentation can be run online, allowing users to test your platform without installing anything or for use in workshops and lectures. These services are usually free, up to limited resources (e.g., Google Colab).

8.3 Overview of *quask*

Existing software fails to provide facilities specifically thought with quantum kernels in mind. We have introduced an open-source library named *quask* to fill this gap.

Quask is an actively maintained library designed for constructing, evaluating, and benchmarking quantum kernel methods. It simplifies selecting a quantum kernel and assessing its suitability for specific tasks while offering a pedagogical platform for newcomers to learn the topic quickly. *Quask* promotes reusable code, as its API seamlessly integrates quantum kernels into existing codebases. It is written in Python 3, easily installed via *pip*, and available on *PyPI*.

Our software fulfills three missions.

Implement complex features with few lines of code Among the limitations of existing software is the inability to offer complex features, requiring them to be rewritten from

scratch. This results in a large waste of time and the potential introduction of errors in the code. Examples of such features include evaluating the spectral bias of a quantum kernel with the Task Model alignment (cf. Section 3.1.2) and perturbing the quantum circuit implementing the feature map using a reinforcement learning agent (cf. Section 3.4). To address this, *quask* provides a high-level API for evaluating and optimizing quantum kernels.

Provide a reliable reference implementation The ever-growing body of literature regarding quantum kernels, often based on complicated mathematical backgrounds, requires significant knowledge and care to be fully understood. Transitioning from theory in journals to actual implementation demands meticulous attention and a considerable investment of time. For some of the major theoretical works in the field, *quask* provides a reliable reference implementation that can be used to verify against their more theoretical definitions. Additionally, tutorials explaining each of these implementations serve as commentary on these works. Examples of these include the tutorials on projected quantum kernels (https://quask.readthedocs.io/en/latest/tutorials_quantum/quantum_2_projected.html) and spectral bias (https://quask.readthedocs.io/en/latest/tutorials_quantum/quantum_3_spectralbias.html).

Enable a pedagogical discovery of the topic The scarcity of resources tailored specifically for newcomers to the field of quantum kernels exacerbates the already steep learning curve inherent in such a theoretical domain. To facilitate a quicker understanding of the topic, we have developed a series of tutorials covering various aspects, including the basics of kernel machines (such as linear regression, kernel regression, and SVM), quantum kernel machines (including parametric unitaries as feature maps, Haar-based expressivity, connections to DLA, projected quantum kernel, and spectral bias), advanced features (such as evaluation and automatic optimization of quantum circuits for feature maps), and applications (such as beyond standard model event detection).

8.3.1 A short example

To illustrate the functioning of *quask*, we present a simple example. We instantiate an SVM (using the SVC class from the `scikit-learn` library) whose kernel function is a quantum kernel. First, its latest version will define the quantum kernel using *Qiskit*.

For simplicity, we consider the dataset x to be a single real value, the number of qubits to be two, and the number of gates to be a single XX gate (which in Qiskit must be decomposed into multiple gates).

```
theta = Parameter('theta')
qreg = QuantumRegister(2, 'q')
creg = ClassicalRegister(2, 'c')
qc = QuantumCircuit(qreg, creg)
qc.h(qreg[0])
qc.h(qreg[1])
qc.u1(theta, qreg[0])
qc.u1(theta, qreg[1])
qc.cx(qreg[0], qreg[1])
sampler = Sampler()
fidelity = ComputeUncompute(sampler=sampler)
adhoc_kernel = FidelityQuantumKernel(fidelity=fidelity, feature_map=qc)
model = SVC(kernel=adhoc_kernel.evaluate)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

The equivalent code in *quask* is much shorter. This is primarily because *quask* natively supports the XX rotation (whereas for quantum circuits defined using the more common U_3 and CNOT gates, the opposite might occur). Additionally, the *QiskitKernel* class abstracts away the complexities of the library, providing a much higher-level interface.

```
ansatz = Ansatz(n_features=1, n_qubits=2, n_operations=1)
ansatz.change_operation(0, new_feature=0, new_wires=[0, 1], \
    new_generator="XX", new_bandwidth=0.9)
kernel = QiskitKernel(ansatz, "ZZ", KernelType.FIDELITY)
model = SVC(kernel='precomputed')
K_train = kernel.build_kernel(X_train, X_train)
model.fit(K_train, y_train)
K_test = kernel.build_kernel(X_test, X_train)
y_pred = model.predict(K_test)
```

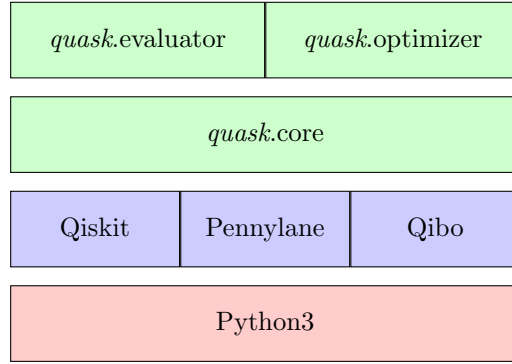


Figure 8.1: Structure of *quask*: The software is written in Python 3 and is built on top of existing quantum frameworks such as Qiskit, PennyLane, and Qibo. The core module of *quask* implements main objects, such as the ansatz and the kernel, which are directly mapped onto quantum circuits and explicitly embed all the features necessary for expressivity control, such as bandwidth. An even higher-level API is also designed to evaluate and optimize these kernels.

However, this example does not fully demonstrate the library’s potential, as *quask* can do much more. For instance, setting up a Bayesian optimizer to automatically select the best quantum circuit for a task, according to the Kernel Model Alignment (Equation 3.2), requires just three lines of code in *quask*. In contrast, achieving the same functionality with Qiskit would require significantly more code.

```
ce = CenteredKernelAlignmentEvaluator()
optimizer = BayesianOptimizer(kernel, X_train, y_train, ce)
optimized_kernel = optimizer.optimize(n_epochs=100, n_points=4, n_jobs=4)
```

8.4 Functioning of *quask*: API and architecture

8.4.1 The *quask.core* module

The *quask.core* module implements the fundamental classes forming the feature map. Its design follows the model described in Section 3.4.1. The classes of this module are schematized in Figure 8.2.

The Operator class Each gate in the underlying quantum circuit is represented by an *Operator*. Any operation corresponds to a two-qubit gate in the form of Equation 3.29,

which is defined as:

$$G(\theta) = \exp\left(-i\frac{\beta\theta}{2}\sigma_1^{(p)}\sigma_2^{(q)}\right) \quad (8.1)$$

where $\theta \in \mathbb{R}$ is the single real parameter that defines the unitary rotation. The operation is characterized by its generators, σ_1 and σ_2 , the qubits on which the operation is applied, p and q , and the bandwidth parameter $\beta \in [0, 1]$. The parameter θ corresponds to one of the components of $\mathbf{x} \in \mathbb{R}^d$ and can be identified with the index $i \in \{0, \dots, d-1\}$, with a special value $i = d$ indicating a constant gate with $\theta = 1$, independent of the input. The qubit indices p and q range in $\{0, \dots, n-1\}$, with n being the number of qubits, and they must be distinct. Finally, $\beta \in [0, 1]$ must belong to a discretized, relatively small range.

The Ansatz class An Ansatz is a sequence of parameterized quantum gates. This class wraps individual operations and performs consistency checks to ensure the validity of each operation. The Ansatz object contains shared information about the quantum circuit to avoid repeating checks for each operation. This shared information includes the number of features d in the classical data vector, the total number of operations in the quantum circuit, and the number of qubits n in the quantum system.

Arbitrary quantum circuits to generators-only quantum circuits We specify operations in terms of Pauli matrix generators instead of more commonly used quantum gates like U_3 and CNOT. This approach aids in analyzing the Lie algebra associated with the parameterized quantum circuit, providing insights into the quantum kernel's functioning while allowing the implementation of all usual quantum gates. For example, any U_3 quantum gate can be defined in terms of three rotations R_Z , R_X , R_Z . Another important example is the CNOT gate, which can be similarly decomposed using an R_{XX} gate and some single-qubit gates.

The Kernel class The Kernel class is a wrapper for the ansatz and the quantum circuit's architecture performing the inner product. We can set the kernel to perform an overlap test, measuring all or just some qubits (in the latter case, we have an instance of projected quantum kernels), or we can measure the feature map according to one or more fixed observables and perform the classical inner product between vectors of estimated quantities.

The KernelFactory class One of the main advantages of *quask* is its compatibility with various frameworks. We work with Kernel objects, high-level descriptions of the operations we want to perform, which are then compiled into low-level objects via different quantum SDKs.

The KernelFactory object manages the different implementations. We cannot directly instantiate Kernel objects (as the class is abstract). Instead, we use `KernelFactory.create_kernel`, which has the same arguments as `Kernel.__init__`.

The role of KernelFactory is to choose the subclass of Kernel that implements the methods on a specific backend and instantiates the object. To do this, we first list all available implementations.

Serialization Both Operator and Ansatz can be serialized into NumPy arrays. The Operator is serialized into a vector of five elements, where each pair of generators corresponds to a single element. The vector includes the two qubits, the feature, and the bandwidth. The elements in the vector are of type float, so certain elements need to be transformed from float encoding back to integer encoding during deserialization.

The Ansatz is serialized into a matrix of $p \times 5$ elements, where p is the number of operations, representing each operation individually. It's important to note that serializing an Ansatz object at this level results in the loss of information regarding the number of features, operations, and qubits.

The kernel object can be serialized into a NumPy array. When deserializing a kernel object, the `KernelFactory.create_kernel` method is invoked, and the default backend of KernelFactory is chosen. The default behavior of the KernelFactory class can be changed via its API.

Backends The `core.implementation` package contains the low-level implementation of the Kernel class on various quantum programming frameworks.

The `core.KernelFactory` class is used to instantiate Kernel objects on the chosen, pre-configured backend. The researcher does not need to install all backends locally; they only need to install the one they plan to use.

The available backends include the Qiskit Backend (upgraded to version 1.0), which supports simulation and execution on IBM hardware. It allows for fine-tuning various configurations on the IBM platform, such as the quantum circuit's optimization level and the use of error-mitigation protocols. The PennyLane Backend functions similarly, allowing the selection of the preferred simulator. The plugin structure of PennyLane

enables *quask* to support many other platforms, such as Braket. The backend for Qibo is under active development.

It is possible to write custom backends if it is necessary to target a particular platform or feature. In fact, *quask* is designed modularly, allowing researchers to modify it to suit their needs with minimal effort. To create a new backend, a class that extends `Kernel` must be created and the following abstract methods must be implemented,

- **kappa**: Calculates the inner product between a pair of data points.
- **phi**: Calculates the feature map corresponding to a single data point if the `kernel_type` field is observable; otherwise, it throws an error.

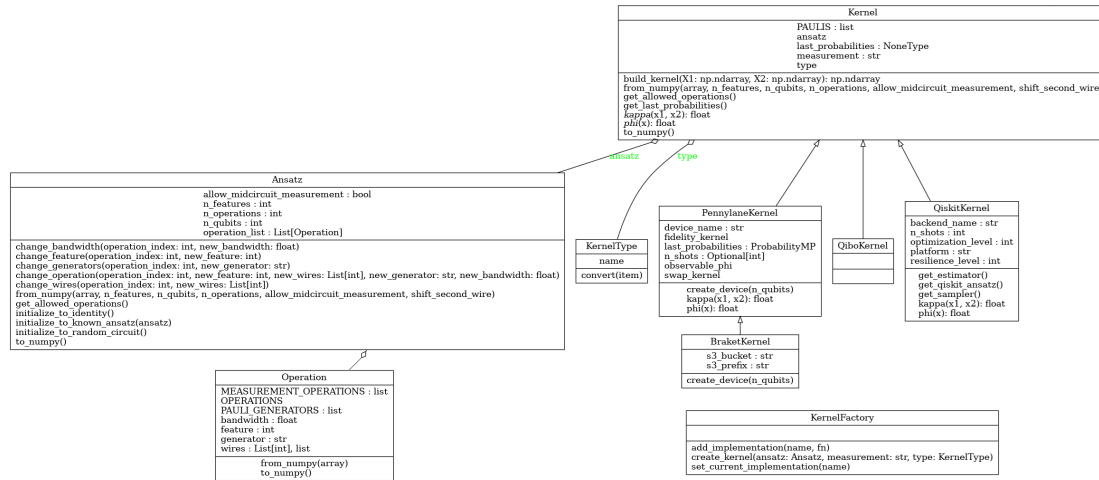


Figure 8.2: Core module of *quask*. Its classes map directly to the components a quantum circuit is made of. The `Operation` class describes a quantum gate, the `Ansatz` class describes a parametric quantum circuit, and the `Kernel` class describes the kernel object, which consists of an ansatz and the type of measurement involved. The `Kernel` class is abstract, with concrete classes being `[PennyLane, Qiskit, Qibo, Braket]Kernel`, each implementing the object on a different backend. The `KernelFactory` class determines the actual backend used by providing a `create_kernel` method that relies on the configured backend.

8.4.2 The *quask.evaluator* module

A key feature of *quask* is its capability to evaluate quantum kernels using various established criteria from the literature. These criteria are crucial when striving for quantum advantage, where the goal is for a quantum model to surpass classical models.

Each criterion is implemented as a class derived from the abstract `KernelEvaluator` class. This class defines a single abstract method, `evaluate`, which takes four parameters:

- The kernel object to be evaluated.
- The training dataset X , which some criteria might require.
- The labels y , which may be used alongside the training data for certain criteria.
- The kernel Gram matrix K , which is optional and can be generated from `kappa` and X .

The optional parameter K is included for scenarios where the Gram matrix has already been computed, allowing faster evaluations. Any evaluator has to implement the `evaluate` methods uniquely. The class diagram for this module is shown in Figure 8.3.

The built-in evaluators are the following ones:

- **HaarEvaluator**: A criterion inspired by the definition of expressivity given in [41]. A discretized, approximated version of this metric is given, comparing the histogram of inner products between Haar random vectors with the inner product of vectors generated with the given kernel `kappa`. It implements the formula in Equation 2.68.
- **LieRankEvaluator**: A criterion inspired by the work in [56]. The rank of the Lie algebra associated with the ansatz is computed and truncated to a maximum threshold value T . In this case, the criterion can be associated both with expressivity (higher rank leads to higher expressivity) and with the efficiency of simulation on a classical device (higher rank leads to harder-to-simulate unitaries).
- **CoveringNumberEvaluator**: A criterion inspired by the work in [196]. The expressivity is upper bounded by a quantity exponential in the number of trainable gates. In our context, it is quite a loose bound, but the original article allows us to consider more precise bounds in some cases.
- **GeometricDifferenceEvaluator**: A criterion inspired by the work in [44]. It quantifies whenever the quantum kernel diverges largely enough from a list of simple classical kernels fixed a priori. It is thoroughly described in Section 3.3.1.
- **KernelAlignmentEvaluator**: A criterion inspired by the work in [50]. It implements the formula in Equation 3.2.

- **CenteredKernelAlignmentEvaluator**: A criterion inspired by the work in [51]. It is similar to the kernel alignment, but the mean of both matrices is subtracted beforehand.
- **RidgeGeneralizationEvaluator**: Accuracy of the model trained using the given kernel over a validation set.
- **EssModelComplexityEvaluator**: A criterion inspired by the work in [44] quantifying the complexity of the function. It is thoroughly described in Section 3.3.1. It implements the formula in Equation 3.18.
- **SpectralBiasEvaluator**: A criterion inspired by the work in [52]. It implements the formula in Equation 3.8.

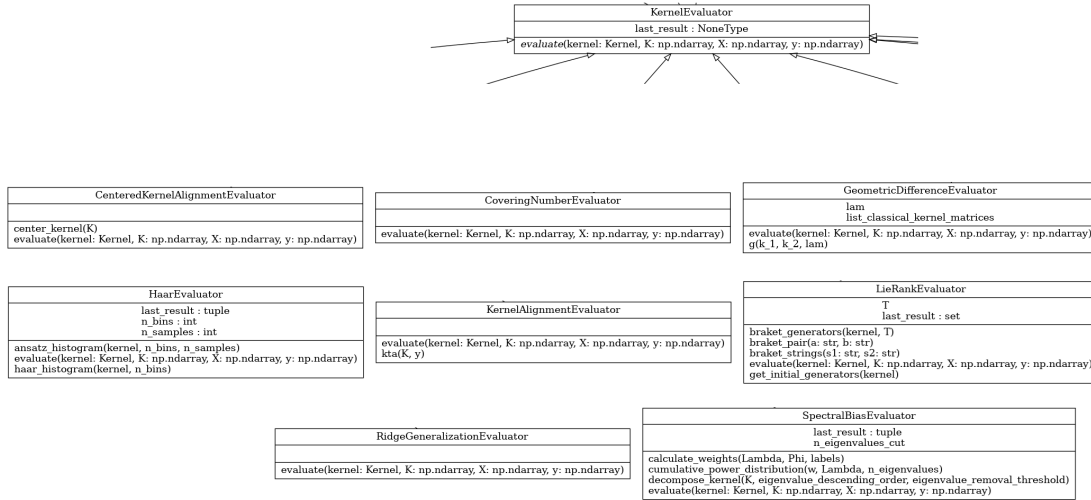


Figure 8.3: Evaluator module of *quask*. The **KernelEvaluator** interface provides a simple API to quantify the fitness of the kernel for the task at hand. The other classes extend this interface. Notably, it is possible to define ad-hoc criteria by extending this interface and possibly making them depend on the already defined criteria.

8.4.3 The *quask.optimizer* module

One of the main challenges regarding quantum kernels is selecting the appropriate ansatz for each specific problem. When there is little-to-no information about a given task, techniques can be used to construct a suitable quantum kernel by solving an optimization problem. Two main approaches are possible:

- Choosing an ansatz where some parameters are the usual features, while others are freely tunable parameters optimized according to a cost function via a stochastic gradient descent-based algorithm.
- Avoiding any initial choice and letting an optimization algorithm determine the entire quantum circuit.

The `quask.optimizer` module demonstrates how to handle both cases. The class diagram for this module is shown in Figure 8.4.

The BaseKernelOptimizer class The `quask.optimizer` package allows defining an optimization procedure for quantum kernels. The main interface is the `BaseKernelOptimizer` class, which requires:

- A kernel object is the starting candidate solution for the optimization routine.
- A kernel evaluator object is the cost function guiding the optimization.
- Potentially some input data, if needed by the kernel evaluator.

The optimizer method then initiates the optimization and returns a new instance of `quask.core.Kernel` for use. Researchers will use `BaseKernelOptimizer` directly when creating a new optimization method. For example, a simple gradient-descent-based scheme can be implemented by calculating the gradient of the optimizer function (e.g., via finite difference) and using this information to minimize the cost.

The BayesianOptimizer class Bayesian optimization is the simplest and usually the most effective technique. It is known to work best for low-dimensional problems where the function to optimize is a costly-to-evaluate black box, which is often the case in our context (although the optimization might not be so low-dimensional). This approach is based on the *scikit-optimize* library.

The ReinforcementLearningOptimizer The `ReinforcementLearningOptimizer` class relies on the *mushroom rl* library [197]. Its simple structure requires defining an environment (in our case, the `WideKernelEnvironment` class) that tells the agent what the possible moves are and how the state is updated and rewarded. These pieces of information are derived directly from the evaluator component. The library then implements state-of-the-art policies.

The GreedyOptimizer The GreedyOptimizer has been implemented from scratch, although it relies on the WideKernelEnvironment class defined for the reinforcement learning optimizer for convenience. The best value is chosen locally for each kernel component, and each component is determined sequentially.

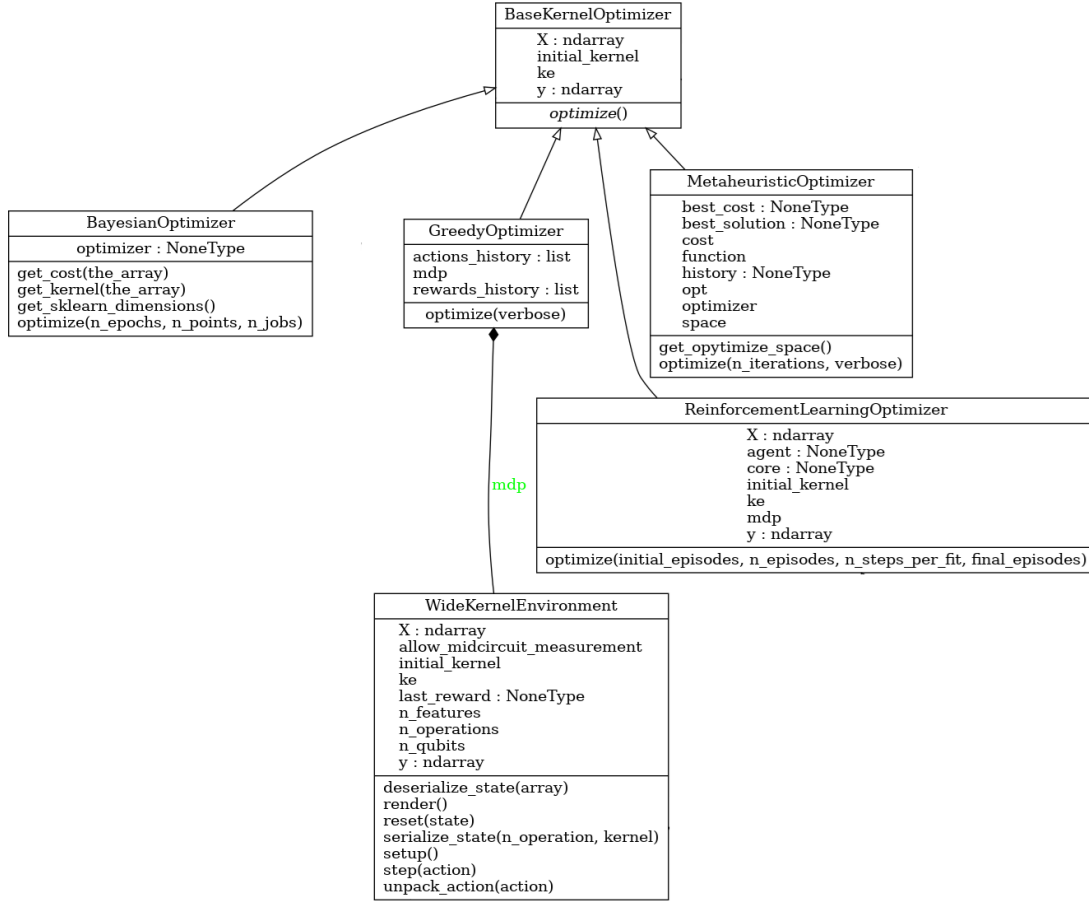


Figure 8.4: Optimizer module of *quask*. Each optimizer provides a simple API with a single method, as they inherit from the **BaseKernelOptimizer** interface. This interface’s single instantiation can be much more complex and is often based on other Python3 libraries.

8.5 Resources of *quask* for the community

Software in the open-source community is only as good as its resources, which ensure stable and continuous support for the platform. We briefly review the tools we use and the resources we have created to support our software and the researchers using it.

8.5.1 GitHub

GitHub is an online platform based on the open source software Git, allowing for the complete management of a project, from code to deployment to maintenance. Among the variety of features offered by this platform, we rely on the following ones:

- **Version control:** The main feature, inherited from Git, allows the versioning of the code for each commit while enabling several branches.
- **Collaboration management:** Features such as Pull Requests allow seamless control over code uploaded by each contributor, ensuring it is checked before being merged with the existing code base. The Issue functionality keeps track of bugs in the code and can also be used for feature requests, while the project board shows and tracks progress.
- **Code review:** Before each update is merged into the existing codebase, the code can be reviewed, and contributors can be asked to modify the update before merging. Other facilities supported include branch protection and fine-grained branch ownership and access.
- **Continuous deployment:** Every time you commit a change, a clean virtual machine is instantiated, the library is installed, and the tests are run. After the tests run successfully, the commit is merged, ensuring each modification does not break the software.
- **Visibility:** The star system of GitHub rewards repositories with more contributions and researchers interested in tracking updates on the software. This way, users remain closely updated with respect to the library.

In contrast, we do not use the documentation features offered by GitHub, relying instead on the more sophisticated choice of ReadTheDocs. It is worth noting that GitHub offers space to host wiki-like documentation written in Markdown, which meets only the basic needs. Furthermore, GitHub can be integrated with the most popular IDEs, a feature that has not influenced our choice. The GitHub page of the documentation is pictured in Figure 8.5. It can be accessed at:

<https://github.com/CERN-IT-INNOVATION/QuASK>

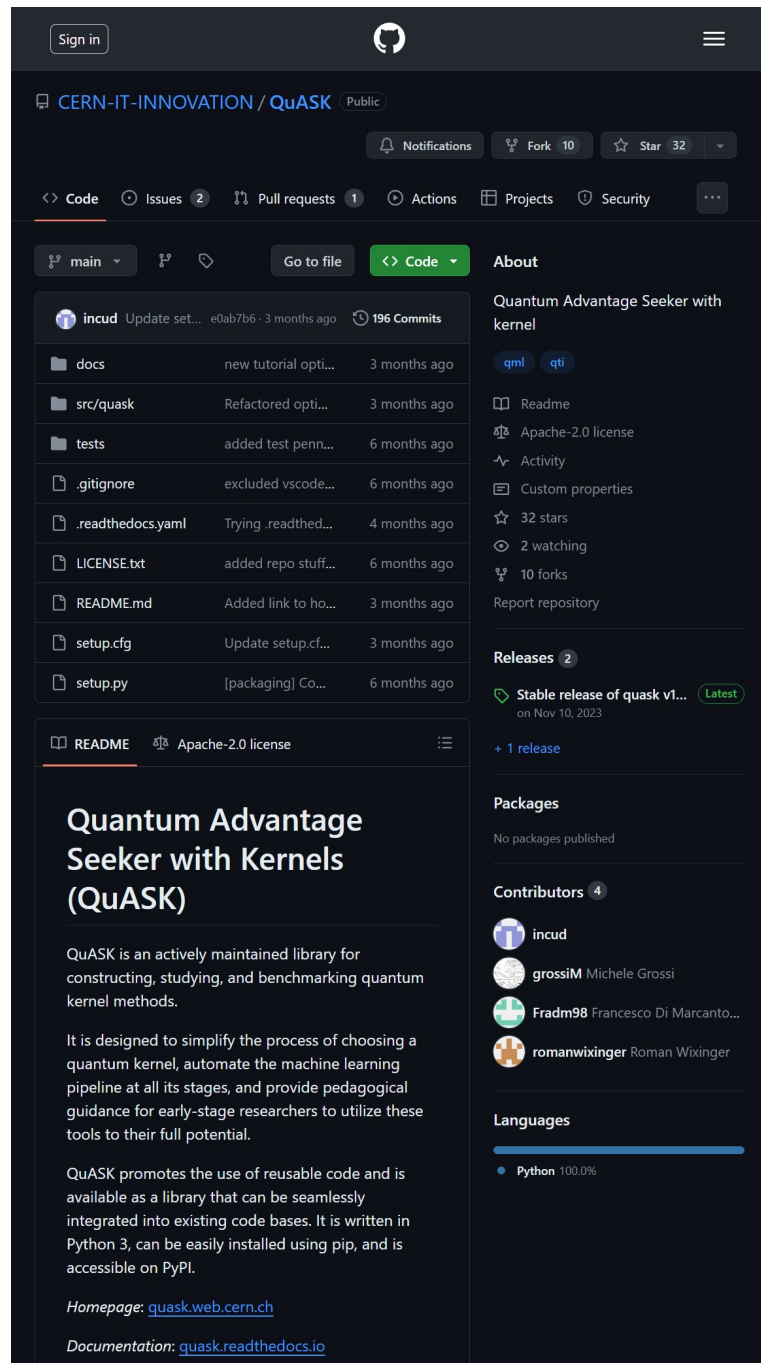


Figure 8.5: Main page of the code repository hosted on GitHub. It shows the stable branch of the code and the latest contributions with dates and authors. A brief summary regarding the management of the repository, including how to install, how to contribute, and how to access other resources, is listed below the code.

8.5.2 Documentation and tutorials

Documentation for open-source software is of paramount importance. Among several libraries created for the same purpose, the main criterion for selection is often the quality of the documentation. For this reason, the online platform *ReadTheDocs* is a good choice.

The documentation itself can be embedded within the source code as Python docstrings, in Jupyter Notebooks for tutorials, or in Markdown files for static content. The software Sphinx can generate an HTML website containing all these resources. ReadTheDocs enhances this functionality: by connecting your GitHub repository to ReadTheDocs, it will recognize all the documentation sources with the help of a few configuration files in the root directory of the repository and autonomously generate the documentation online using its own instance of Sphinx. This enables continuous integration in the documentation process: each commit in GitHub will trigger ReadTheDocs to regenerate the documentation from scratch, including the tutorials. This is crucial because if a code modification is not reflected in the tutorials, their execution will fail, resulting in an error and allowing developers to fix the tutorials immediately.

Among the features offered is automatic versioning of the documentation, which allows us to maintain documentation for different software versions for free. It also supports the automatic management of multiple languages, a feature we do not currently use but plan to support based on community demand, and offers a visually appealing theme.

This platform includes, in the form of Jupyter Notebooks, a series of lectures introducing classical kernel machines and quantum kernels developed using *quask*. The front page of the documentation is pictured in Figure 8.6. It can be accessed at:

`https://quask.readthedocs.io`

8.5.3 Homepage and visibility

The final resource offered is a webpage. Its purpose is to highlight the features and benefits of using *quask* compared to other platforms, and to provide links to various resources such as the Getting Started tutorial, installation instructions, and documentation. The collaborative nature of *quask* leads us to recognize the support of each contributor, who is listed directly on this homepage. Furthermore, users of *quask* can apply to be

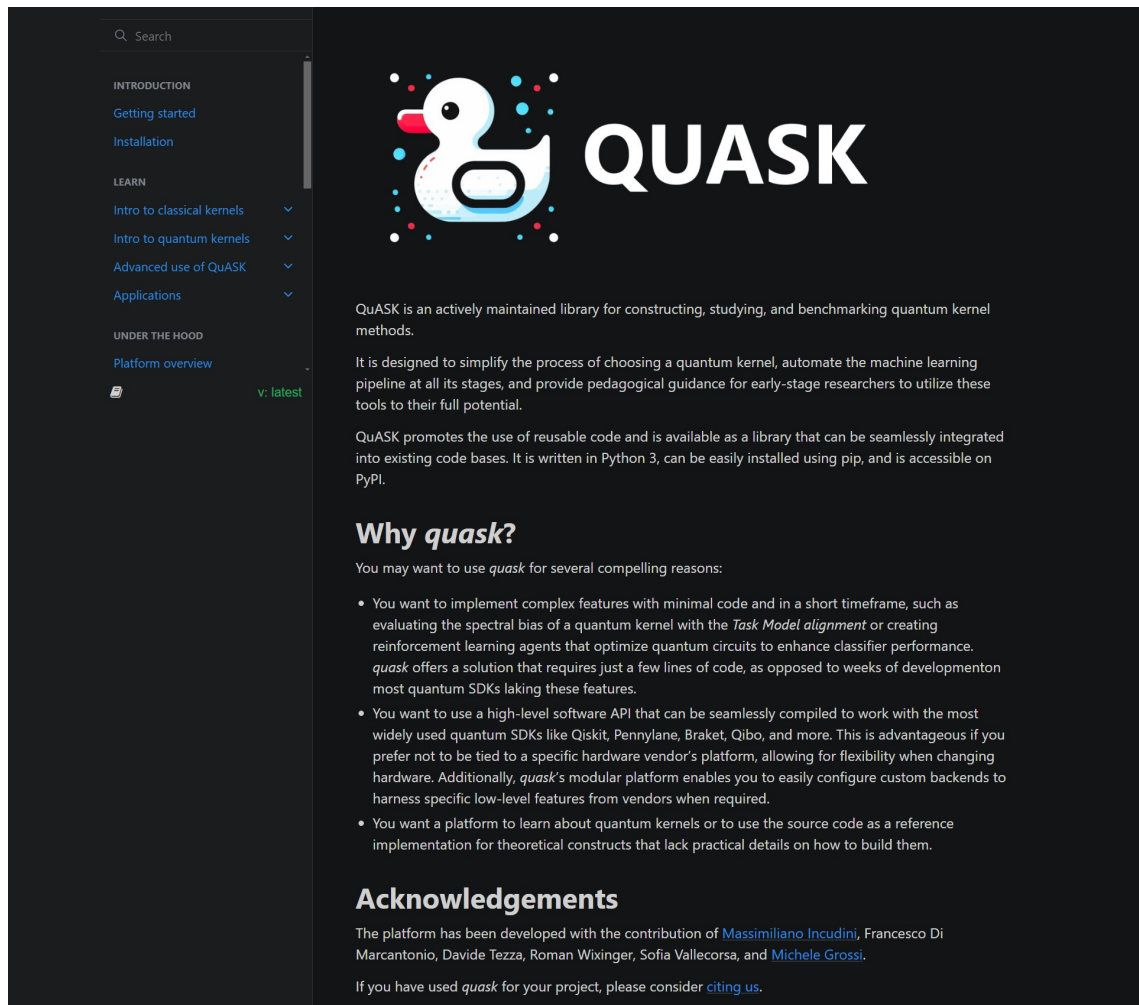


Figure 8.6: Front page of the documentation hosted on ReadTheDocs. It consists of a brief presentation of the library, the motivation behind its design, and its features. On the side, the Getting Started and Installation tutorials are shown along with the pedagogical material.

sponsored to showcase their scientific works. The home page is pictured in Figure 8.7. It can be accessed at:

<https://quask.web.cern.ch>

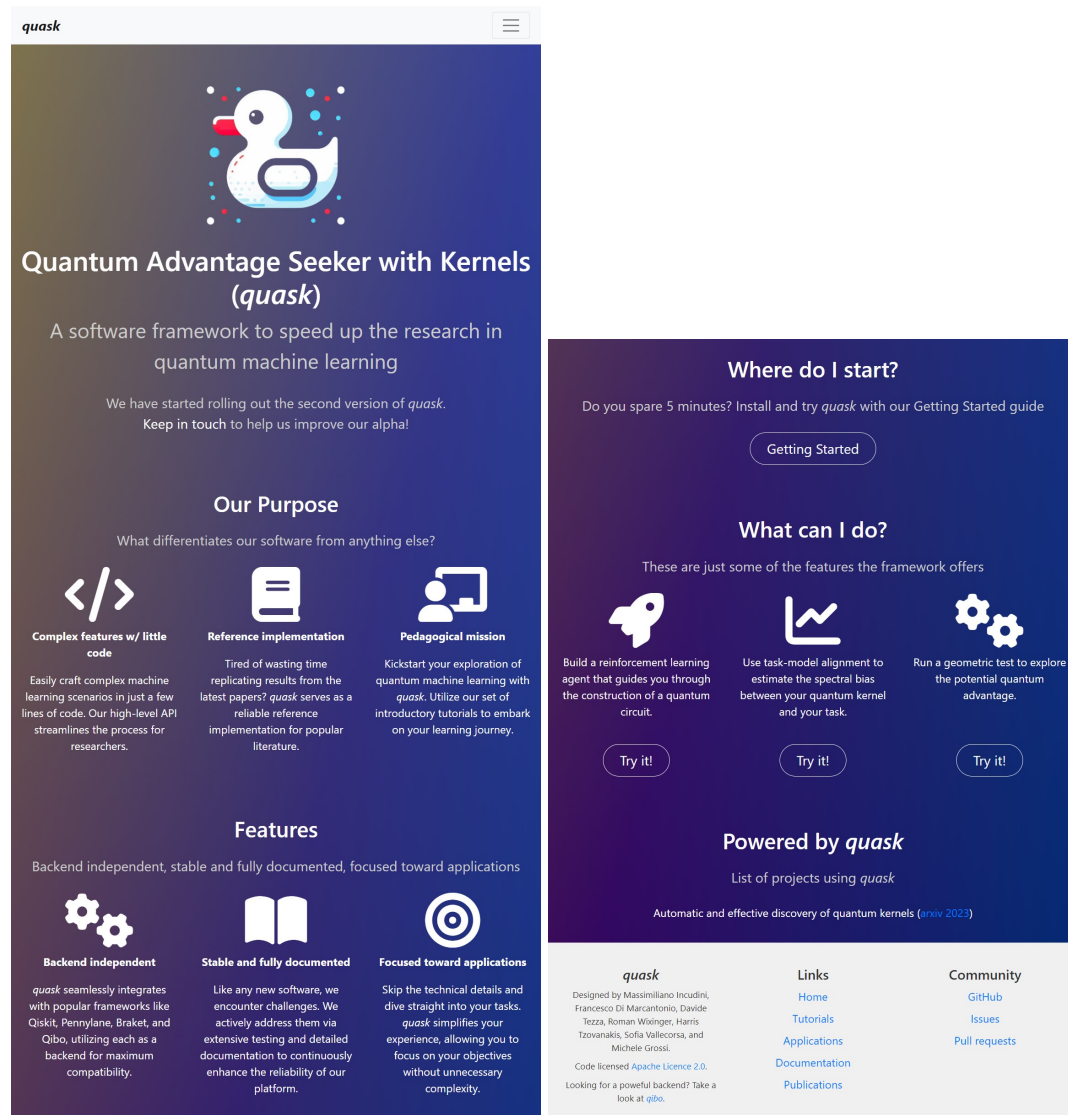


Figure 8.7: Home page of *quask*. The top section of the home page introduces the tool, the motivation behind its implementation, and its features. The bottom section presents tutorials, showcases works based on *quask*, and credits the contributors.

Vendor	Name	Ref	Language	S	E	QML
-	ProjectQ	[175]	Imperative	Yes	No	No
-	QCL	[176]	Imperative	Yes	No	No
-	QiBO/Qibolab	[177, 178]	Python	Yes	No	No
-	Quipper	[179]	Domain-specific (Haskell)	Yes	No	No
-	Quirk	[180]	Drag-and-drop	Yes	No	No
-	SilQ	[181]	Imperative	Yes	No	No
Amazon	Braket	[182]	Python	Yes	Yes	No
Baidu	Paddle Quantum	[183]	Python	Yes	No	Yes
Google	Cirq	[184]	Python	Yes	Yes	No
Google	TensorFlow Quantum	[185]	Python	Yes	Yes	Yes
IBM	Qiskit	[160]	Python	Yes	Yes	Yes
Microsoft	Azure Quantum	[186]	Python, Q#	Yes	Yes	No
Microsoft	LIQUi ⟩	[187]	Domain-specific (F#)	Yes	No	No
Quantinuum	t ket⟩	[188]	Python	Yes	Yes	No
Rigetti	Forest (pyQuil)	[189]	Python	Yes	Yes	No
Xanadu	PennyLane	[190]	Python	Yes	Yes	Yes
Xanadu	Strawberry Field	[191]	Python	Yes	Yes	Yes

Table 8.1: Some popular quantum computing frameworks and their characteristics. Legend: S: Supports simulation on CPU, E: Supports execution on quantum devices, QML: Includes quantum machine learning capabilities. Domain-specific languages are embedded in the language indicated in parentheses.

This page has been intentionally left blank.

Chapter 9

Conclusion

Quantum machine learning plays a paramount role in delineating the frontier of what can be computed on quantum devices, both with and without the aid of data. Beginning with the work of Huang et al. [44], learning properties of physical systems have proven challenging for classical devices but naturally more feasible for quantum devices. However, with the availability of data, the task can be simplified to the extent that it becomes trivial for classical devices as well. To address the central theme and research question of this thesis, *How can we determine tasks where the quantum approach truly disrupts other methods?* we investigate this topic through the lens of the powerful mathematical framework of kernel methods.

Focusing on NISQ devices, experimental evidence of a potential advantage dates **Chapter 3** back to the work of Havlíček et al. [21], based on the assumption that evolving the quantum system according to complex enough unitaries makes the computation hard for a classical device. However, several theoretical and practical drawbacks have since arisen: the statistically unfavorable properties make expressible unitaries unfit for all machine learning tasks, while hands-on benchmarking has highlighted the failure of such techniques for all but artificially crafted tasks. Despite these issues, the practical applicability remains unresolved, leaving open the question: *Is it possible to design an ansatz that is tailored for the task at hand, has favorable statistical properties, and retains a computational advantage compared to classical devices?* We address this by proposing an automatic approach to construct such a unitary from the data. Our approach differs from others by rendering optimization dependent on expressivity and the hardness of classical simulability of the resulting unitary, guiding the method directly toward a set of provably interesting solutions. We tested the framework on the classification of

beyond-standard model events. We chose this problem because it is physically motivated, with an underlying target function likely closer to the model we are developing, and it has been statistically proven to be effectively solved by a quantum device. We found that our approach is able to outperform or at least match the performance of previous works without external insight into the data, which we consider a success. However, the main limitation of our approach lies in its strong experimental dependence on the task at hand. While we specifically chose this problem to allow a comparison with preceding techniques, it does not fully address whether these types of models make a good general-purpose approach or if they will be limited to a few physically motivated tasks or even none.

Chapter 4 In investigating the role of NISQ-ready quantum kernels as general-purpose machine learning models, one might question, *what are the characteristics unique to the most popular classical models, and how can I translate them to their quantum analogue?* We identify three characteristics typical of deep learning models: feature learning, i.e., evolving features during training, hierarchical dependencies between features, and over-parameterization, i.e., having more parameters than necessary to interpolate all data while still having the freedom to choose a solution with minimum norm. The feature learning capabilities of quantum neural networks (which share the structure of quantum kernels but provide a more convenient interface for this specific task) have been studied by Liu et al. [198]. Defining a hierarchy of representations is much more complex due to the unitarity of quantum evolution effectively ‘collapsing’ the layers of the transformation. We propose the use of the Quantum Path Kernel, which introduces a temporal dependency on the parameters (compared to a spatial dependency in the chain rule of neural networks) and test its efficacy against the Gaussian XOR Mixture—a synthetic benchmark. We found that the quantum path kernel outperforms the quantum NTK in a layer-wise context: the layer-wise growth of performance is larger for the QPK, indicating some advantage beyond the increase in expressivity. Nevertheless, this improvement is modest compared to the large computational cost of such a tool, suggesting the need for continued exploration. Possible directions include the use of measurements, structured networks of quantum neurons, or QSP-based approaches. Regarding over-parameterization, the challenge is that the large resource requirements contrast with the scarcity of those available on NISQ devices, making the approach theoretically feasible but not practical. Inspired by the work of Geiger et al. [24], we investigate reaching the over-parameterization regime via a small ensemble of models. We test ensemble approaches on real-world, generic datasets with no physical

motivation; the results suggest a decent performance improvement along with significant resource savings. However, a limitation of our results is not reaching zero loss during training, which would classify our model as over-parameterized. Instead, we observe an improvement in performance (which is the goal) but with a weak relation to the deep learning model characteristics we aim to study. Future efforts might be directed toward entering this regime.

Up to this point, traditional, NISQ-ready quantum kernels on Euclidean data have been the only paradigm considered. In search of a more pragmatic approach and some guarantees, we now turn to the realm of fault-tolerant quantum computing and its tools. These have been applied to combinatorial optimization tasks, not with the pretense of solving NP-hard optimization tasks efficiently but rather suggesting potential advantages in heuristic contexts. Therefore, we ask, *is there an advantage in using quantum computing as a heuristic for machine learning models over highly structured data?* We focus on the calculation of the graph edit distance, a dissimilarity measure widely used in chemistry and biology tasks; calculating this metric exactly is NP-hard. We formulate its calculation as a QUBO optimization problem and solve it exactly on a classical device, heuristically on a D-Wave quantum annealer, and using QAOA simulated noiselessly on a classical device, against a small synthetic dataset. As these experiments were run on the first generations of quantum annealers, the results were severely influenced by noise, and the classical baseline outperformed the quantum approach. Similarly, QAOA was found to be a weak heuristic for reasons unrelated to noise, as the circuits were simulated. Despite the negative experimental results, a positive note is found in the formulation itself, which consists mostly of quadratic terms and does not rely on the 'quadratrization' of higher-order terms, adding a large overhead to the computation. This characteristic suggests that an improvement in hardware could lead to better performance, as the optimization problem fundamentally accommodates the QUBO formulation. Future work can characterize which classes of graphs lead to QUBO formulations best adapted to the topology of various versions of quantum annealers. **Chapter 5**

The more structured but still highly flexible nature of graphs has led to a kernel function that is NP-hard to compute, thus forcing us to rely on heuristics. *Can we design quantum kernels on even more structured data and benefit from strong theoretical guarantees of speedup in processing such data on a quantum device?* To investigate this, we turn to fault-tolerant quantum hardware. We focus on tasks involving abstract simplicial complexes, a generalization of graphs that greatly accommodates the framework of quantum computing: the elements in an abstract simplicial complex **Chapter 6**

can be combinatorially many, but their sparsity structure can still be easily encoded on a quantum computer, which is not feasible on classical hardware. As these objects play an important role in topology, we can exploit this aspect to define a quantum kernel based on topological invariants of these objects. These topological invariants are estimated as spectral properties of Hamiltonians associated with the object, which is naturally suited for quantum algorithms and can be performed efficiently under some mild assumptions. We present a toy example to illustrate the applicability of our approach in shape recognition tasks. This model accommodates the need for practical applicability and guaranteed speedup, making it a more promising path toward general-purpose machine learning. Even though topological features are still under active research and may be considered niche (in contrast to more popular approaches), their use is rapidly rising and might become standard in many scenarios. One main limitation is the reliance on fault tolerance. More work is needed to test the applicability of the algorithm in the early fault-tolerant regime.

Chapter 7 Finally, we turn our attention to the software portion of the machine learning pipeline. Despite the rapid evolution of the quantum software landscape, especially within the machine learning context, a substantial development effort is still required to develop and use quantum kernel-based approaches. Simplifying the development process can lead to faster and less error-prone research. *Can we design an architecture flexible enough to work with many types of quantum kernels but with a minimal amount of error-free code?* To answer this, we have designed *quask*, a quantum framework built on top of existing libraries. It allows for the efficient implementation of quantum kernels while providing maximum flexibility in terms of expressivity control. Automatic approaches shown in Chapter 3 are allowed. We demonstrate that a reinforcement learning agent learning the quantum kernel for an arbitrary task can be implemented in less than 20 lines of code. This framework, along with the pedagogical material offered, is expected to kickstart the work of early-stage quantum researchers and speed up the work of existing ones. Future work can focus on extending the framework to allow for more general, non-NISQ-ready approaches to quantum kernels.

Chapter 10

Bibliography

- [1] Massimiliano Incudini et al. “Computing graph edit distance on quantum devices”. In: *Quantum Machine Intelligence* 4.2 (2022), p. 24.
- [2] Massimiliano Incudini et al. “Resource saving via ensemble techniques for quantum neural networks”. In: *Quantum Machine Intelligence* 5.2 (2023), p. 39.
- [3] Francesco Di Marcantonio et al. “Quantum Advantage Seeker with Kernels (QuASK): a software framework to speed up the research in quantum machine learning”. In: *Quantum Machine Intelligence* 5.1 (2023), p. 20.
- [4] Massimiliano Incudini et al. “The Quantum Path Kernel: a Generalized Neural Tangent Kernel for Deep Quantum Machine Learning”. In: *IEEE Transactions on Quantum Engineering* (2023).
- [5] Massimiliano Incudini, Francesco Martini, and Alessandra Di Pierro. “Higher-order topological kernels via quantum computation”. In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 1. IEEE. 2023, pp. 621–629.
- [6] Massimiliano Incudini, Francesco Martini, and Alessandra Di Pierro. “Toward Useful Quantum Kernels”. In: *Advanced Quantum Technologies* (2024), p. 2300298.
- [7] Massimiliano Incudini et al. “Automatic and effective discovery of quantum kernels”. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* (2024).

- [8] Markus Hoffmann et al. “Network medicine-based epistasis detection in complex diseases: ready for quantum computing”. In: *Nucleic Acids Research* 52.17 (2024), pp. 10144–10160.
- [9] Riccardo Mengoni, Massimiliano Incudini, and Alessandra Di Pierro. “Facial expression recognition on a quantum computer”. In: *Quantum Machine Intelligence* 3 (2021), pp. 1–11.
- [10] Alessandra Di Pierro and Massimiliano Incudini. “Quantum Machine Learning and Fraud Detection”. In: *Protocols, Strands, and Logic*. Cham, Germany: Springer, 2021, pp. 139–155.
- [11] Michele Grossi et al. “Amplitude-assisted tagging of longitudinally polarised bosons using wide neural networks”. In: *The European Physical Journal C* 83.8 (2023), p. 759.
- [12] Marco Maronese et al. “The maximum-likelihood quantum amplitude estimation algorithm provides the best tradeoff between accuracy and circuit depth among quantum solutions for integral estimation”. In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 1. IEEE. 2023, pp. 554–559.
- [13] Marco Maronese et al. “The Quantum Amplitude Estimation Algorithms on Near-Term Devices: A Practical Guide”. In: *Quantum Reports* 6.1 (2023), pp. 1–13.
- [14] Riccardo Manenti and Mario Motta. *Quantum Information Science*. Oxford: Oxford University Press, 2023. ISBN: 9780198787488.
- [15] Richard P Feynman et al. “Simulating physics with computers”. In: *Int. J. Theor. phys* (1982).
- [16] Jacob Biamonte et al. “Quantum machine learning”. In: *Nature* 549.7671 (2017), pp. 195–202.
- [17] Carlo Ciliberto et al. “Quantum machine learning: a classical perspective”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 474.2209 (2018), p. 20170551.
- [18] Oriel Kiss et al. “Quantum neural networks force field generation”. In: *Mach. Learn.: Sci. Technol.* 3 (2022), p. 035004. doi: <https://doi.org/10.1088/2632-2153/ac7d3c>.

- [19] Oriel Kiss et al. “Quantum computing of the ${}^6\text{Li}$ nucleus via ordered unitary coupled clusters”. In: *Phys. Rev. C* 106 (3 Oct. 2022), p. 034325. DOI: 10.1103/PhysRevC.106.034325. URL: <https://link.aps.org/doi/10.1103/PhysRevC.106.034325>.
- [20] Maria Schuld and Nathan Killoran. “Quantum Machine Learning in Feature Hilbert Spaces”. In: *Physical Review Letters* 122.4 (Feb. 2019). DOI: 10.1103/physrevlett.122.040504. URL: <https://doi.org/10.1103/PhysRevLett.122.040504>.
- [21] Vojtěch Havlíček et al. “Supervised learning with quantum-enhanced feature spaces”. In: *Nature* 567.7747 (2019), pp. 209–212.
- [22] Maria Schuld and Nathan Killoran. “Is Quantum Advantage the Right Goal for Quantum Machine Learning?” In: *PRX Quantum* 3.3 (3 July 2022), p. 030101. DOI: 10.1103/PRXQuantum.3.030101. URL: <https://link.aps.org/doi/10.1103/PRXQuantum.3.030101>.
- [23] Joseph Bowles, Shah Nawaz Ahmed, and Maria Schuld. “Better than classical? the subtle art of benchmarking quantum machine learning models”. In: *arXiv preprint arXiv:2403.07059* (2024).
- [24] Mario Geiger et al. “Scaling description of generalization with number of parameters in deep learning”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2020.2 (2020), p. 023401.
- [25] Arthur L Samuel. “Some studies in machine learning using the game of checkers”. In: *IBM Journal of research and development* 3.3 (1959), pp. 210–229.
- [26] Ali Rahimi and Benjamin Recht. “Random features for large-scale kernel machines”. In: *Advances in neural information processing systems* 20 (2007).
- [27] Arthur Jacot, Franck Gabriel, and Clément Hongler. “Neural tangent kernel: Convergence and generalization in neural networks”. In: *Advances in neural information processing systems* 31 (2018).
- [28] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. “Quantum algorithm for linear systems of equations”. In: *Physical review letters* 103.15 (Oct. 2009), p. 150502. ISSN: 1079-7114. DOI: 10.1103/physrevlett.103.150502. URL: <http://dx.doi.org/10.1103/PhysRevLett.103.150502>.

- [29] Scott Aaronson. “Read the fine print”. In: *Nature Physics* 11.4 (2015), pp. 291–293.
- [30] Nai-Hui Chia et al. “Sampling-based sublinear low-rank matrix arithmetic framework for dequantizing quantum machine learning”. In: *Journal of the ACM* 69.5 (2022), pp. 1–72.
- [31] Seth Lloyd, Silvano Garnerone, and Paolo Zanardi. “Quantum algorithms for topological and geometric analysis of data”. In: *Nature communications* 7.1 (2016), p. 10138.
- [32] Jin-Peng Liu et al. “Efficient quantum algorithm for dissipative nonlinear differential equations”. In: *Proceedings of the National Academy of Sciences* 118.35 (2021), e2026805118.
- [33] Dylan Lewis et al. “Limitations for Quantum Algorithms to Solve Turbulent and Chaotic Systems”. In: *arXiv preprint arXiv:2307.09593* (2023).
- [34] Junyu Liu et al. “Towards provably efficient quantum algorithms for large-scale machine-learning models”. In: *arXiv preprint arXiv:2303.03428* (2023).
- [35] John Preskill. “Quantum Computing in the NISQ era and beyond”. In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. URL: <http://dx.doi.org/10.22331/q-2018-08-06-79>.
- [36] Marco Cerezo et al. “Variational quantum algorithms”. In: *Nature Reviews Physics* 3.9 (Aug. 2021), pp. 625–644. ISSN: 2522-5820. DOI: 10.1038/s42254-021-00348-9. URL: <http://dx.doi.org/10.1038/s42254-021-00348-9>.
- [37] Maria Schuld et al. “Evaluating analytic gradients on quantum hardware”. In: *Physical Review A* 99.3 (2019), p. 032331.
- [38] Marco Cerezo et al. “Cost function dependent barren plateaus in shallow parametrized quantum circuits”. In: *Nature communications* 12.1 (2021), pp. 1–12.
- [39] Zoë Holmes et al. “Connecting ansatz expressibility to gradient magnitudes and barren plateaus”. In: *PRX Quantum* 3.1 (2022), p. 010313.
- [40] Samson Wang et al. “Noise-induced barren plateaus in variational quantum algorithms”. In: *Nature communications* 12.1 (2021), pp. 1–11.

- [41] Sukin Sim, Peter D Johnson, and Alán Aspuru-Guzik. “Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms”. In: *Advanced Quantum Technologies* 2.12 (2019), p. 1900070.
- [42] Maria Schuld and Francesco Petruccione. “Quantum ensembles of quantum classifiers”. In: *Scientific reports* 8.1 (2018), pp. 1–12.
- [43] Simon Marshall, Casper Gyurik, and Vedran Dunjko. “On Bounded Advice Classes”. In: *arXiv preprint arXiv:2405.18155* (2024).
- [44] Hsin-Yuan Huang et al. “Power of data in quantum machine learning”. In: *Nature communications* 12.1 (2021), pp. 1–9.
- [45] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. “A rigorous and robust quantum speed-up in supervised machine learning”. In: *Nature Physics* 17.9 (2021), pp. 1013–1017.
- [46] Jonas Jäger and Roman V Krems. “Universal expressiveness of variational quantum classifiers and quantum kernels for support vector machines”. In: *Nature Communications* 14.1 (2023), p. 576.
- [47] Casper Gyurik and Vedran Dunjko. “Exponential separations between classical and quantum learners”. In: *arXiv preprint arXiv:2306.16028* (2023).
- [48] Riccardo Molteni, Casper Gyurik, and Vedran Dunjko. “Exponential quantum advantages in learning quantum observables from classical data”. In: *arXiv preprint arXiv:2405.02027* (2024).
- [49] Hayata Yamasaki, Natsuto Isogai, and Mio Murao. “Advantage of Quantum Machine Learning from General Computational Advantages”. In: *arXiv preprint arXiv:2312.03057* (2023).
- [50] Nello Cristianini et al. “On kernel-target alignment”. In: *Advances in neural information processing systems* 14 (2001).
- [51] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. “Algorithms for learning kernels based on centered alignment”. In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 795–828.
- [52] Abdulkadir Canatar, Blake Bordelon, and Cengiz Pehlevan. “Spectral bias and task-model alignment explain generalization in kernel regression and infinitely wide neural networks”. In: *Nature communications* 12.1 (2021), p. 2914.

- [53] Jonas Kübler, Simon Buchholz, and Bernhard Schölkopf. “The inductive bias of quantum kernels”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 12661–12673.
- [54] Supanut Thanasilp et al. “Exponential concentration in quantum kernel methods”. In: *Nature communications* 15.1 (2024), p. 5200.
- [55] Rolando Somma et al. “Efficient solvability of Hamiltonians and limits on the power of some quantum computational models”. In: *Physical review letters* 97.19 (2006), p. 190501.
- [56] Martin Larocca et al. “Diagnosing barren plateaus with tools from quantum optimal control”. In: *Quantum* 6 (2022), p. 824.
- [57] Domenico D’Alessandro. *Introduction to quantum control and dynamics*. CRC press, 2021.
- [58] M Cerezo et al. “Challenges and opportunities in quantum machine learning”. In: *Nature Computational Science* 2.9 (2022), pp. 567–576.
- [59] Abdulkadir Canatar et al. “Bandwidth enables generalization in quantum kernel models”. In: *arXiv preprint arXiv:2206.06686* (2022).
- [60] Johannes Jakob Meyer et al. “Exploiting symmetry in variational quantum machine learning”. In: *PRX Quantum* 4.1 (2023), p. 010328.
- [61] Cenk Tüysüz et al. “Symmetry breaking in geometric quantum machine learning in the presence of noise”. In: *arXiv preprint arXiv:2401.10293* (2024).
- [62] Seth Lloyd et al. *Quantum embeddings for machine learning*. 2020. arXiv: 2001.03622 [quant-ph].
- [63] Jennifer R Glick et al. “Covariant quantum kernels for data with group structure”. In: *arXiv preprint arXiv:2105.03406* (2021).
- [64] Mateusz Ostaszewski et al. “Reinforcement learning for optimization of variational quantum circuit architectures”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 18182–18194.
- [65] Fan-Xu Meng et al. “Quantum Circuit Architecture Optimization for Variational Quantum Eigensolver via Monte Carlo Tree Search”. In: *IEEE Transactions on Quantum Engineering* 2 (2021), pp. 1–10.
- [66] Yuhan Huang et al. “Robust resource-efficient quantum variational ansatz through an evolutionary algorithm”. In: *Physical Review A* 105.5 (2022), p. 052414.

- [67] Yuxuan Du et al. “Quantum circuit architecture search for variational quantum algorithms”. In: *npj Quantum Information* 8.1 (2022), pp. 1–8.
- [68] Sergio Altares-López, Angela Ribeiro, and Juan José García-Ripoll. “Automatic design of quantum feature maps”. In: *Quantum Science and Technology* 6.4 (2021), p. 045015.
- [69] Bang-Shien Chen and Jann-Long Chern. “Genetically auto-generated quantum feature maps”. In: *arXiv preprint arXiv:2207.11449* (2022).
- [70] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. “A review on genetic algorithm: past, present, and future”. In: *Multimedia tools and applications* 80.5 (Feb. 2021), pp. 8091–8126. ISSN: 1573-7721. DOI: 10.1007/s11042-020-10139-6. URL: <https://doi.org/10.1007/s11042-020-10139-6>.
- [71] Yutaka Matsuo et al. “Deep learning, reinforcement learning, and world models”. In: *Neural Networks* 152 (2022), pp. 267–275.
- [72] Kinga Anna Woźniak et al. “Quantum anomaly detection in the latent space of proton collision events at the LHC”. In: *arXiv preprint arXiv:2301.10780* (2023). Unpublished, available as preprint arXiv:2301.10780.
- [73] Julian Schuhmacher et al. “Unravelling physics beyond the standard model with classical and quantum anomaly detection”. In: *arXiv preprint arXiv:2301.10787* (2023). DOI: 10.48550/ARXIV.2301.10787. arXiv: 2301.10787. URL: <https://arxiv.org/abs/2301.10787>.
- [74] Maurizio Pierini and Kinga Anna Wozniak. *Dataset for Quantum anomaly detection in the latent space of proton collision events at the LHC*. Zenodo, Feb. 2023. DOI: 10.5281/zenodo.7673769. URL: <https://doi.org/10.5281/zenodo.7673769>.
- [75] Torbjörn Sjöstrand et al. “An Introduction to PYTHIA 8.2”. In: *Comput. Phys. Commun.* 191 (2015), pp. 159–177. DOI: 10.1016/j.cpc.2015.01.024. arXiv: 1410.3012 [hep-ph].
- [76] J. de Favereau et al. “DELPHES 3, A modular framework for fast simulation of a generic collider experiment”. In: *JHEP* 02 (2014), p. 057. DOI: 10.1007/JHEP02(2014)057. arXiv: 1307.6346 [hep-ex].
- [77] Daniel A. Roberts, Sho Yaida, and Boris Hanin. *The Principles of Deep Learning Theory*. Cambridge University Press, 2022.

- [78] David Meltzer and Junyu Liu. “Catapult Dynamics and Phase Transitions in Quadratic Nets”. In: *arXiv preprint arXiv:2301.07737* (2023).
- [79] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444.
- [80] Minshuo Chen et al. “Towards understanding hierarchical learning: Benefits of neural representations”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 22134–22145.
- [81] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part I* 13. Springer. 2014, pp. 818–833.
- [82] Mikhail Belkin et al. “Reconciling modern machine-learning practice and the classical bias–variance trade-off”. In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854.
- [83] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. “To understand deep learning we need to understand kernel learning”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 541–549.
- [84] Trevor Hastie et al. “Surprises in high-dimensional ridgeless least squares interpolation”. In: *Annals of statistics* 50.2 (2022), p. 949.
- [85] Lenaïc Chizat, Edouard Oyallon, and Francis Bach. “On lazy training in differentiable programming”. In: *Advances in Neural Information Processing Systems* 32 (2019).
- [86] Behrooz Ghorbani et al. “When do neural networks outperform kernel methods?”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2021.12 (2021), p. 124009.
- [87] Maria Refinetti et al. “Classifying high-dimensional Gaussian mixtures: Where kernel methods fail and neural networks succeed”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8936–8947.
- [88] Yu Bai and Jason D Lee. “Beyond linearization: On quadratic and higher-order approximation of wide neural networks”. In: *arXiv preprint arXiv:1910.01619* (2019).

- [89] Yu Bai et al. “Taylorized training: Towards better approximation of neural network training at finite width”. In: *arXiv preprint arXiv:2002.04010* (2020).
- [90] Pedro Domingos. “Every model learned by gradient descent is approximately a kernel machine”. In: *arXiv preprint arXiv:2012.00152* (2020).
- [91] Yilan Chen et al. “On the Equivalence between Neural Network and Support Vector Machine”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 23478–23490.
- [92] Maria Schuld. *Supervised quantum machine learning models are kernel methods*. 2021. arXiv: 2101.11020 [quant-ph].
- [93] Zeyu Deng, Abba Kammoun, and Christos Thrampoulidis. “A model of double descent for high-dimensional binary linear classification”. In: *arXiv preprint arXiv:1911.05822* (2019).
- [94] Xiaoyi Mai and Zhenyu Liao. “High dimensional classification via regularized and unregularized empirical risk minimization: Precise error and optimal loss”. In: *arXiv preprint arXiv:1905.13742* (2019).
- [95] Marc Lelarge and Léo Miolane. “Asymptotic bayes risk for gaussian mixture in a semi-supervised setting”. In: *2019 IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*. IEEE, 2019, pp. 639–643.
- [96] Evan Peters and Maria Schuld. “Generalization despite overfitting in quantum machine learning models”. In: *arXiv preprint arXiv:2209.05523* (2022).
- [97] Martin Larocca et al. “Theory of overparametrization in quantum neural networks”. In: *Nature Computational Science* 3.6 (2023), pp. 542–551.
- [98] Samet Oymak and Mahdi Soltanolkotabi. “Toward moderate overparameterization: Global convergence guarantees for training shallow neural networks”. In: *IEEE Journal on Selected Areas in Information Theory* 1.1 (2020), pp. 84–105.
- [99] Angus Lowe et al. “Fast quantum circuit cutting with randomized measurements”. In: *arXiv preprint arXiv:2207.14734* (2022).
- [100] Peng Zhang et al. “Robust ensemble learning for mining noisy data streams”. In: *Decision Support Systems* 50.2 (2011), pp. 469–479.
- [101] Cha Zhang and Yunqian Ma. *Ensemble machine learning: methods and applications*. New York, NY: Springer US, 2012.

- [102] Nicolas De Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. Cambridge Library Collection - Mathematics. Cambridge: Cambridge University Press, 2014. DOI: 10.1017/CB09781139923972.
- [103] Yoav Freund and Robert E Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.
- [104] Torgyn Shaikhina and Natalia A Khovanova. "Handling limited datasets with neural networks in medical applications: A small-data approach". In: *Artificial intelligence in medicine* 75 (2017), pp. 51–63.
- [105] Yogesh Gupta. "Selection of important features and predicting wine quality using machine learning techniques". In: *Procedia Computer Science* 125 (2018), pp. 305–312.
- [106] Fabio Nelli and Fabio Nelli. "Machine Learning with scikit-learn". In: *Python Data Analytics: With Pandas, NumPy, and Matplotlib* (2018), pp. 313–347.
- [107] Harris Drucker. "Improving regressors using boosting techniques". In: *ICML*. Vol. 97. Citeseer. 1997, pp. 107–115.
- [108] Trevor Hastie et al. "Multi-class adaboost". In: *Statistics and its Interface* 2.3 (2009), pp. 349–360.
- [109] Yeeun Yoo, Jinho Shin, and Sunghyon Kyeong. "Medicare fraud detection using graph analysis: a comparative study of machine learning and graph neural networks". In: *IEEE Access* (2023).
- [110] Rui Yan et al. "Citation count prediction: learning to estimate future citations for literature". In: *Proceedings of the 20th ACM international conference on Information and knowledge management*. 2011, pp. 1247–1252.
- [111] Zahra Gharaee et al. "Graph representation learning for road type classification". In: *Pattern Recognition* 120 (2021), p. 108174.
- [112] Carlos Garcia-Hernandez, Alberto Fernandez, and Francesc Serratosa. "Ligand-based virtual screening using graph edit distance as molecular similarity measure". In: *Journal of chemical information and modeling* 59.4 (2019), pp. 1410–1421.
- [113] Jie Zhou et al. "Graph neural networks: A review of methods and applications". In: *AI open* 1 (2020), pp. 57–81.

- [114] Xiaowen Dong et al. “Graph signal processing for machine learning: A review and new perspectives”. In: *IEEE Signal processing magazine* 37.6 (2020), pp. 117–127.
- [115] László Babai. “Graph isomorphism in quasipolynomial time”. In: *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. 2016, pp. 684–697.
- [116] Nino Shervashidze et al. “Weisfeiler-lehman graph kernels.” In: *Journal of Machine Learning Research* 12.9 (2011).
- [117] David B Blumenthal and Johann Gamper. “On the exact computation of the graph edit distance”. In: *Pattern Recognition Letters* 134 (2020), pp. 46–57.
- [118] Bas Lodewijks. “Mapping NP-hard and NP-complete optimisation problems to Quadratic Unconstrained Binary Optimisation problems”. In: *arXiv preprint arXiv:1911.08043* (2019).
- [119] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. “Optimization by simulated annealing”. In: *science* 220.4598 (1983), pp. 671–680.
- [120] Bruno Apolloni, C Carvalho, and Diego De Falco. “Quantum stochastic optimization”. In: *Stochastic Processes and their Applications* 33.2 (1989), pp. 233–244.
- [121] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. *A Quantum Approximate Optimization Algorithm*. 2014. arXiv: 1411.4028 [quant-ph].
- [122] D-Wave Systems Inc. *QPU-Specific Physical Properties: DW_2000Q_6*. Tech. rep. D-Wave Systems Inc., May 2020. URL: https://docs.dwavesys.com/docs/latest/_downloads/92a6dea183a90d7d4f6d305c94812c8d/09-1215A-B_QPU_Properties_DW_2000Q_6.pdf.
- [123] D-Wave Systems Inc. *QPU-Specific Physical Properties: Advantage_system1.1*. Tech. rep. D-Wave Systems Inc., Sept. 2020. URL: https://docs.dwavesys.com/docs/latest/_downloads/9a1b594d84370df6c0a09d00a5b72661/09-1237A-A_QPU_Properties_Advantage_system1_1.pdf.
- [124] Jeffrey Marshall et al. “Power of pausing: Advancing understanding of thermalization in experimental quantum annealers”. In: *Physical Review Applied* 11.4 (2019), p. 044083.

- [125] Michael JD Powell. “A direct search optimization method that models the objective and constraint functions by linear interpolation”. In: *Advances in optimization and numerical analysis*. Springer, 1994, pp. 51–67.
- [126] David B Blumenthal et al. “GEDLIB: A C++ Library for Graph Edit Distance Computation”. In: *GbRPR 2019*. Ed. by Donatello Conte, Jean-Yves Ramel, and Pasquale Foggia. Vol. 11510. LNCS. Cham: Springer, 2019, pp. 14–24. DOI: 10.1007/978-3-030-20081-7_2.
- [127] Felix Hensel, Michael Moor, and Bastian Rieck. “A survey of topological machine learning methods”. In: *Frontiers in Artificial Intelligence* 4 (2021), p. 681108.
- [128] Peter Bubenik et al. “Statistical topological data analysis using persistence landscapes.” In: *J. Mach. Learn. Res.* 16.1 (2015), pp. 77–102.
- [129] Alexander Schmidhuber and Seth Lloyd. “Complexity-theoretic limitations on quantum algorithms for topological data analysis”. In: *PRX Quantum* 4.4 (2023), p. 040349.
- [130] Casper Gyurik, Chris Cade, and Vedran Dunjko. “Towards quantum advantage via topological data analysis”. In: *Quantum* 6 (Nov. 2022), p. 855.
- [131] Sam McArdle, András Gilyén, and Mario Berta. “A streamlined quantum algorithm for topological data analysis with exponentially fewer qubits”. Unpublished, available as preprint arXiv:2209.12887. 2022.
- [132] Lin Lin. “Lecture notes on quantum algorithms for scientific computation”. In: *arXiv preprint arXiv:2201.08309* (2022).
- [133] Guang Hao Low and Isaac L Chuang. “Hamiltonian simulation by qubitization”. In: *Quantum* 3 (2019), p. 163.
- [134] Dominic W. Berry et al. “Analyzing Prospects for Quantum Advantage in Topological Data Analysis”. In: *PRX Quantum* 5 (1 Feb. 2024), p. 010319. DOI: 10.1103/PRXQuantum.5.010319. URL: <https://link.aps.org/doi/10.1103/PRXQuantum.5.010319>.
- [135] Simon Apers et al. “A (simple) classical algorithm for estimating Betti numbers”. In: *Quantum* 7 (2023), p. 1202.
- [136] Yuhei Umeda. “Time series classification via topological data analysis”. In: *Information and Media Technologies* 12 (2017), pp. 228–239.

- [137] Ryu Hayakawa. “Quantum algorithm for persistent betti numbers and topological data analysis”. In: *Quantum* 6 (2022), p. 873.
- [138] Amara Katabarwa et al. “Early fault-tolerant quantum computing”. In: *PRX Quantum* 5.2 (2024), p. 020101.
- [139] Oriel Kiss, Michele Grossi, and Alessandro Roggero. “Importance sampling for stochastic quantum simulations”. In: *Quantum* 7 (2023), p. 977.
- [140] Teri A Manolio et al. “Finding the missing heritability of complex diseases”. en. In: *Nature* 461.7265 (Oct. 2009), pp. 747–753.
- [141] Gizem Caylak, Oznur Tastan, and A Ercument Cicek. “Potpourri: An Epistasis Test Prioritization Algorithm via Diverse SNP Selection”. In: *J. Comput. Biol.* (Dec. 2020).
- [142] Tyler Cowman and Mehmet Koyutürk. “Prioritizing tests of epistasis through hierarchical representation of genomic redundancies”. en. In: *Nucleic Acids Res.* 45.14 (Aug. 2017), e131.
- [143] Marzieh Ayati and Mehmet Koyutürk. “Prioritization of genomic locus pairs for testing epistasis”. In: *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics. BCB '14*. Newport Beach, California: Association for Computing Machinery, Sept. 2014, pp. 240–248.
- [144] Peng-Jie Jing and Hong-Bin Shen. “MACOED: a multi-objective ant colony optimization algorithm for SNP epistasis detection in genome-wide association studies”. en. In: *Bioinformatics* 31.5 (Mar. 2015), pp. 634–641.
- [145] Diane Duroux et al. “Interpretable network-guided epistasis detection”. en. In: *Gigascience* 11 (Feb. 2022).
- [146] David B Blumenthal et al. “A framework for modeling epistatic interaction”. en. In: *Bioinformatics* (Nov. 2020).
- [147] Gizem Caylak, Oznur Tastan, and A. Ercument Cicek. “Potpourri: An Epistasis Test Prioritization Algorithm via Diverse SNP Selection”. In: *Journal of Computational Biology* 28.4 (2021), pp. 365–377. DOI: 10.1089/cmb.2020.0429. URL: <https://doi.org/10.1089/cmb.2020.0429>.

- [148] Xia Cao et al. “DualWMDR: Detecting epistatic interaction with dual screening and multifactor dimensionality reduction”. In: *Human Mutation* 41.3 (Nov. 2019), pp. 719–734. DOI: 10.1002/humu.23951. URL: <https://doi.org/10.1002/humu.23951>.
- [149] Javad Ansarifard and Lizhi Wang. “New algorithms for detecting multi-effect and multi-way epistatic interactions”. In: *Bioinformatics* 35.24 (June 2019). Ed. by John Hancock, pp. 5078–5085. DOI: 10.1093/bioinformatics/btz463. URL: <https://doi.org/10.1093/bioinformatics/btz463>.
- [150] S T Sherry et al. “dbSNP: the NCBI database of genetic variation”. en. In: *Nucleic Acids Res.* 29.1 (Jan. 2001), pp. 308–311.
- [151] Rose Oughtred et al. “The BioGRID interaction database: 2019 update”. en. In: *Nucleic Acids Res.* 47.D1 (Jan. 2019), pp. D529–D541.
- [152] V A Traag, L Waltman, and N J van Eck. “From Louvain to Leiden: guaranteeing well-connected communities”. en. In: *Sci. Rep.* 9.1 (Mar. 2019), pp. 1–12.
- [153] Kaspar Riesen, Andreas Fischer, and Horst Bunke. “Improved graph edit distance approximation with simulated annealing”. In: *GbRPR 2017*. Ed. by Pasquale Foggia, Cheng-Lin Liu, and Mario Vento. Vol. 10310. LNCS. Cham: Springer, 2017, pp. 222–231. DOI: 10.1007/978-3-319-58961-9_20.
- [154] David B Blumenthal et al. “Comparing heuristics for graph edit distance computation”. In: *VLDB J.* 29.1 (Jan. 2020), pp. 419–458.
- [155] Brian W Kunkle et al. “Genetic meta-analysis of diagnosed Alzheimer’s disease identifies new risk loci and implicates A β , tau, immunity and lipid processing”. en. In: *Nat. Genet.* 51.3 (Feb. 2019), pp. 414–430.
- [156] Eric M Reiman et al. “GAB2 alleles modify Alzheimer’s risk in APOE epsilon4 carriers”. en. In: *Neuron* 54.5 (June 2007), pp. 713–720.
- [157] Wellcome Trust Case Control Consortium. “Genome-wide association study of CNVs in 16,000 cases of eight common diseases and 3,000 shared controls”. en. In: *Nature* 464.7289 (Apr. 2010), pp. 713–720.
- [158] David J Earl and Michael W Deem. “Parallel tempering: Theory, applications, and new perspectives”. In: *Physical Chemistry Chemical Physics* 7.23 (2005), pp. 3910–3916.

- [159] J Robert Johansson, Paul D Nation, and Franco Nori. “QuTiP: An open-source Python framework for the dynamics of open quantum systems”. In: *Computer Physics Communications* 183.8 (2012), pp. 1760–1772.
- [160] MD Sajid Anis, Abby-Mitchell, Héctor Abraham, et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2021. DOI: 10.5281/zenodo.2573505.
- [161] Myron Lipow. “Number of faults per line of code”. In: *IEEE Transactions on software Engineering* 4 (1982), pp. 437–439.
- [162] Fiona Fidler et al. “Meta-research for evaluating reproducibility in ecology and evolution”. In: *BioScience* 67.3 (2017), pp. 282–289.
- [163] Rotem Botvinik-Nezer et al. “Variability in the analysis of a single neuroimaging dataset by many teams”. In: *Nature* 582.7810 (2020), pp. 84–88.
- [164] José Campos and André Souto. “Qbugs: A collection of reproducible bugs in quantum algorithms and a supporting infrastructure to enable controlled quantum software testing and debugging experiments”. In: *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*. IEEE. 2021, pp. 28–32.
- [165] Ana Trisovic et al. “A large-scale study on research code quality and execution”. In: *Scientific Data* 9.1 (2022), pp. 1–16.
- [166] Patrick Mineault and Kento Nozawa. *patrickmineault/codebook: 1.0.0*. Version releases. Dec. 2021. DOI: 10.5281/zenodo.5796873. URL: <https://doi.org/10.5281/zenodo.5796873>.
- [167] Nathan Shammah. *Make Your Code Count*. 2021. URL: <https://github.com/nathanshammah/make-your-code-count> (visited on 07/22/2021).
- [168] Andrew W Cross et al. “Open quantum assembly language”. In: *arXiv preprint arXiv:1707.03429* (2017).
- [169] André Van Tonder. “A lambda calculus for quantum computation”. In: *SIAM Journal on Computing* 33.5 (2004), pp. 1109–1135.
- [170] David Deutsch. “Quantum theory, the Church–Turing principle and the universal quantum computer”. In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (1985), pp. 97–117.
- [171] Edward Farhi et al. “Quantum computation by adiabatic evolution”. In: *arXiv preprint quant-ph/0001106* (2000).

- [172] Robert Raussendorf and Hans J Briegel. “A one-way quantum computer”. In: *Physical review letters* 86.22 (2001), p. 5188.
- [173] A Yu Kitaev. “Fault-tolerant quantum computation by anyons”. In: *Annals of Physics* 303.1 (2003), pp. 2–30.
- [174] Bob Coecke and Ross Duncan. “Interacting quantum observables: categorical algebra and diagrammatics”. In: *New Journal of Physics* 13.4 (2011), p. 043016.
- [175] Damian S Steiger, Thomas Häner, and Matthias Troyer. “ProjectQ: an open source software framework for quantum computing”. In: *Quantum* 2 (2018), p. 49.
- [176] Bernhard Ömer. “Classical concepts in quantum programming”. In: *International Journal of Theoretical Physics* 44.7 (2005), pp. 943–955.
- [177] Stavros Efthymiou et al. “Qibo: a framework for quantum simulation with hardware acceleration”. In: *Quantum Science and Technology* 7.1 (2021), p. 015018.
- [178] Stavros Efthymiou et al. “Qibolab: an open-source hybrid quantum operating system”. In: *Quantum* 8 (2024), p. 1247.
- [179] Alexander S Green et al. “Quipper: a scalable quantum programming language”. In: *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*. 2013, pp. 333–342.
- [180] Craig Gidney. *Quirk*. Available at: <https://github.com/Strilanc/Quirk>. 2014.
- [181] Benjamin Bichsel et al. “Silq: A High-Level Quantum Language with Safe Uncomputation and Intuitive Semantics”. In: *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2020. London, UK: Association for Computing Machinery, 2020, pp. 286–300. ISBN: 9781450376136. DOI: 10.1145/3385412.3386007. URL: <https://doi.org/10.1145/3385412.3386007>.
- [182] Amazon Web Services. *Amazon Braket*. 2020. URL: <https://aws.amazon.com/braket/>.
- [183] Baidu. *Paddle Quantum*. 2020. URL: <https://github.com/PaddlePaddle/Quantum>.
- [184] Cirq Developers. *Cirq*. Apr. 2022. DOI: 10.5281/zenodo.6599601.

- [185] Michael Broughton et al. “Tensorflow quantum: A software framework for quantum machine learning”. In: *arXiv preprint arXiv:2003.02989* (2020).
- [186] Microsoft. *Azure Quantum SKD*. 2020. URL: <https://learn.microsoft.com/en-us/azure/quantum>.
- [187] Dave Wecker and Krysta M Svore. “LIQUi: A software design architecture and domain-specific language for quantum computing”. In: *arXiv preprint arXiv:1402.4467* (2014).
- [188] Seyon Sivarajah et al. “tket: a retargetable compiler for NISQ devices”. In: *Quantum Science and Technology* 6.1 (2020), p. 014003.
- [189] Rigetti. *Pyquil*. 2019. URL: <http://pyquil.readthedocs.io/en/latest>.
- [190] Ville Bergholm et al. “PennyLane: Automatic differentiation of hybrid quantum-classical computations”. In: *arXiv preprint arXiv:1811.04968* (2018).
- [191] Nathan Killoran et al. “Strawberry Fields: A Software Platform for Photonic Quantum Computing”. In: *Quantum* 3 (2019), p. 129. DOI: 10.22331/q-2019-03-11-129. arXiv: 1804.03159.
- [192] David Wierichs et al. “General parameter-shift rules for quantum gradients”. In: *Quantum* 6 (2022), p. 677.
- [193] Francois Chollet et al. *Keras*. 2015. URL: <https://github.com/fchollet/keras>.
- [194] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: (2019). Ed. by H. Wallach et al., pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [195] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <https://www.tensorflow.org/>.
- [196] Yuxuan Du et al. “Efficient measure for the expressivity of variational quantum algorithms”. In: *Physical Review Letters* 128.8 (2022), p. 080506.
- [197] Carlo D’Eramo et al. “MushroomRL: Simplifying Reinforcement Learning Research”. In: *Journal of Machine Learning Research* 22.131 (2021), pp. 1–5. URL: <http://jmlr.org/papers/v22/18-056.html>.

- [198] Junyu Liu et al. *An analytic theory for the dynamics of wide quantum neural networks*. 2022. DOI: 10.48550/ARXIV.2203.16711. URL: <https://arxiv.org/abs/2203.16711>.