ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

COMPUTER SCIENCE AND ENGINEERING

Ciclo 38

Settore Concorsuale: 09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore Scientifico Disciplinare: ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

VARIATIONAL ALGORITHMS FOR QUANTUM MACHINE LEARNING

Presentata da: Filippo Orazi

Coordinatore Dottorato

Paola Salomoni

Supervisore

Stefano Lodi

Co-supervisore

Ugo Dal Lago

Esame finale anno 2026

Abstract

In recent years, we have observed a new revolution in computer science, driven by the increasing availability of computational power and data. This historic event has seen the capillary diffusion of what is commonly known as artificial intelligence (AI). In particular, we have seen a rapid improvement of large language models that now can interact with people on a level that appeared impossible just a few years ago.
Despite this newly found focus on AI and computer science, the technical difficulties that limit the increase in computational power are increasing, and the prediction described by Moore's law no longer holds. For this reason physicists, computer scientists, mathematicians, and engineers are studying another computational paradigm called quantum computing.
Similar to the rise of large language models, quantum computing has gained significant attention in the last ten years due to the development of initial quantum hardware and its rapid advancements.
The quantum computing paradigm offers a fundamentally different set of tools with respect to what is called classical computation. Quantum properties and effects like superposition, entanglement, and interference give us the ability to solve problems that we could not approach classically.
In this thesis, we explore the intersection between artificial intelligence and quantum computing from an experimental point of view. We researched quantum AI models, utilizing both the classical and quantum paradigms using variational architectures.

This thesis is organized as follows: part I contains an introduction to quantum computation, starting from the postulate of quantum mechanics, through an introduction to QML and finally the state of the art.
Part II contains the description of the research we performed. It's divided into two main chapters containing fundamentally different approaches to quantum computing. Each chapter contains section illustrating our research on that approach.

Acknowledgements

The completion of my Ph.D. is not a result of my own doing, but a collective achievement that stems from the efforts of a great number of people.

An obvious but nonetheless incredibly important thanks goes to my family: my parents, my grandmothers, my aunt and uncle, my brother, and my partner. You always supported and believed in me, especially when I could only focus on the negative side of things.

Secondly, I need to thank my friends, near and far, who could always distract me from my problems, giving me time to relax and heal. A special thanks goes to my roommates, Augusto and Arianna, for the patience, the understanding, and the shared lingering anguish caused by life in academia and life in general.

Then a special thanks goes to my supervisors, Professor Lodi and Professor Sartori, who gave me the opportunity to seriously research what was nothing more than a personal fascination. This thanks also extends to Antonio, to whom I owe this journey, and to Johannes.

Here a special thought goes to the people who gave me the tools to start this journey: Mirella, Prof. Montesi, Prof. Marchetti, Prof. Righi, and Prof. Giuliani. You gave me the curiosity that guided me through something far from my comfort zone, and for that, I am extremely thankful.

A thanks goes also to the QML group: Matteo, Simone, and Supreeth, who walked with me the path into quantum computing and perplexity.

Reflecting upon my studies, it is clear that I always wanted to explore something more than just computer science: science in high school, Computer Engineering in my Bachelor's, Statistics in my Master's, and finally Quantum Mechanics in my Ph.D. Now, at the end of my journey, I have approximate knowledge of many things, and even if none of it proves useful, it was surely a fun adventure.

Contents

List of Figures

Chapter 1

Introduction

Quantum mechanics is the branch of physics that studies how atomic and subatomic particles behave. In the early 20th century, following what is known as the ultraviolet catastrophe [3], scientists realized that very small particles do not follow the same physical laws as macroscopic objects. During this time, great minds discovered more and more about how the universe behaves. In particular, it was Max Planck who understood that energy at the atomic level is not a continuous phenomenon but a quantized one. Following this, in 1907, Albert Einstein [4] demonstrated that not only light but also atomic vibrations are quantized as well. These discoveries led to the formulation of two different quantum mechanics theories by Schrödinger [5] and Heisenberg [6]. These parallel formulations, though seemingly different, ultimately described the same phenomena, marking the birth of modern quantum mechanics.

In the early '80s researchers founded the field known as quantum computing by combining quantum mechanics and computer science: Paul Benioff defined in 1980 a quantum Turing machine [7] based on the reversible Turing machine by Bennett [8] and Richard P Feynman [9] that asserted the need for quantum machinery for the study of quantum phenomena in a conference in 1982. The field slowly grew until 1994 when Peter W. Shor (1997) published "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithm on a Quantum Computer" [10], an article where he describes two algorithms that could solve in (error-bounded) polynomial-time problems that belong to the NP complexity class for traditional

methods. The consequences of this paper were important, but the limited technology available at that time reduced the field to theoretical speculation. Nonetheless, the promise of efficiently solving the NP problem sparked significant interest in the scientific community.
Today, different quantum hardware is available in the private sector, but some companies like IBM, IQM, and D'Wave allow researchers to use their quantum computers. The world shows a profound interest in this technology, and while the United Nations celebrated 2025 as the International Year of Quantum Science and Technology, countries worldwide are currently investing more than 55.7 billion dollars in quantum computing technologies [11] (both hardware and software).

In this thesis, we report the research conducted in the past few years about variational quantum machine learning, a research field that merges quantum computation and machine learning and adapts it to the current state of quantum hardware to study algorithms that will run learning techniques in near-term (NISQ) devices.
The thesis is structured as follows: part I contains an introduction to quantum computing in chapter 2, a brief overview of quantum machine learning in chapter 3, and finally, a summary of the state of the art in chapter 4. The content of the first part is thought to ease non-experts into the field of quantum machine learning and to give them the tools to better understand part II. Part II contains the description of the research we performed. This section of the thesis is divided into two parts containing approaches to different technologies: chapter 5 explores four contributions related to the gate-based paradigm, chapter 6 is divided into three sections with related contributions that tackle problems in the quantum annealing paradigm.

Part I

Quantum computing and machine learning

Chapter 2

Quantum computing

This chapter provides an introduction to quantum computing, the creation of quantum circuits, and their application in achieving the desired results. This introduction will start from the four quantum mechanics principles that describe qubits, gates, measurements, and registers, respectively.

2.1 Quantum mechanics' postulates

The aspect of quantum mechanics described from now on is the one that relates to the use of quantum computers, and we will not discuss the broader field. The postulates described here are written as in Nielsen and Chuang's book *Quantum Computation and Quantum Information* [12]

2.1.1 State space postulate

Postulate 1 *Associated with any isolated physical system is a complex vector space with inner product (that is, a Hilbert space) known as the state space of the system. The system is completely described by its state vector, which is a unit vector in the system's state space.*

The simplest Quantum system is called a *qubit* and can be described as a linear combination of two basis states in the two-dimensional Hilbert space. A more in-depth formalization and description of the qubit can be found in section 2.2.1.

This postulate affirms that every quantum state lives inside a complex space and that we can have a complete description of this state using the state's basis. For this research, we treat qubits as pure mathematical concepts, disregarding the complexity of the physical implementation, and we assume that, one day, we will obtain a quantum computer efficient enough that the physical implementation does not influence the execution.

2.1.2 Evolution postulate

Postulate 2 *The evolution of a closed quantum system is described by a unitary transformation. That is, the state $|\psi\rangle$ of the system at time t_1 is related to the state $|\psi'\rangle$ of the system at time t_2 by a unitary operator U which depends only on the times t_1 and t_2.*

$$|\psi'\rangle = U\,|\psi\rangle$$

Much like the first postulate didn't describe the state of the system and only declared the space it lives in, the second postulate tells us that any state can be changed based on a unitary operator without telling us which operator this is. In the case of qubits, any unitary operator can be realized in realistic systems.
This postulate tells us what happens at two different timesteps, but a more complete version allows us to describe their behavior in continuous time:

Postulate 2′ *The time evolution of the state of a closed quantum system is described by the Schrödinger equation,*

$$i\hbar\frac{d\,|\psi\rangle}{dt} = H\,|\psi\rangle$$

In this equation, $\hbar$ is a physical constant known as Planck's constant whose value must be experimentally determined. The exact value is not important to us. In practice, it is common to absorb the factor $\hbar$ into H, effectively setting $\hbar = 1$. H is a fixed Hermitian operator known as the Hamiltonian of the closed system.

This second postulate for the evolution of a system will be important to us in a future section when we will introduce and describe quantum annealing. As for now, we will stick to the discrete time transformations since these are the ones

that are used to describe the evolution of a system in the gate-based approach to quantum computation.

2.1.3 Measurement postulate

The evolution of a system, as described in the previous section, works only for a closed system. This is basically impossible to achieve (only the universe as a whole can be considered a closed system), but we can achieve a good enough approximation that allows us to finish our computations. The next problem comes from accessing the results of the computation, since these results are hidden in a 'closed' system that we have no access to. For this reason, postulate 3 describes the measurement operation, where we interact with the system to retrieve some information.

Postulate 3 *Quantum measurements are described by a collection M_m of measurement operators. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement, then the probability that result m occurs is given by*

$$p(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle$$

and the state of the system after the measurement is

$$\frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^\dagger M_m |\psi\rangle}}$$

The measurement operation satisfies the completeness equation

$$\sum_m M_m^\dagger M_m = I$$

The completeness equation expresses the fact that probabilities sum to one:

$$1 = \sum_m p(m) = \sum_m \langle\psi| M_m^\dagger M_m |\psi\rangle$$

The simplest and most used measurement operation is the measurement of a qubit in the computational basis. This measurement has two possible outcomes (the two basis states) and one operator for each result $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$, if a qubit is described by two complex numbers α, β the probability of obtaining the corresponding basis is $|\alpha|^2, |\beta|^2$.

2.1.4 Composite systems postulate

If we want to utilize a more complex system by using more than one qubit at once, we create what is known as a composite system:

Postulate 4 *The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered 1 through n and system number i is prepared in the state $|\psi_i\rangle$, then the joint state of the total system is $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$*

This postulate is at the basis for one of the most important advantages of quantum computing as well as one of its most difficult parts to manage: if we start from a system composed of a single qubit the space it lives in will be a two dimensional complex space but when we try to consider other qubits in the system the space this system live in increases its dimensionality exponentially in the number of qubit.
This means that with a small number of qubits, we can work in a high-dimensional space, possibly achieving an exponential space advantage.

2.2 Gate based computation

This section will discuss the basics of what is known as the gate-based quantum computing paradigm, by far the most well-known and most popular in recent years. This approach to quantum computation is based on the application of operands to a qubit that can be precisely operated on. To introduce this concept, we will first introduce the fundamental unit of quantum computing, the qubit, and then talk about how we perform computation with this element.

2.2.1 The qubit

The quantum bit, or qubit for short, is the mathematical and physical foundation for quantum computing. It can be represented using Dirac notation as the sum of basis states of the complex space it inhabits, known as Hilbert space $\mathcal{H}$. Mathematically we represent the qubit as a 2-element vector of complex numbers, written using Ket (vertical vector), Bra (horizontal vector), or as a linear combination of the basis $\{|0\rangle, |1\rangle\}$

$$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = |v\rangle = v_1 |0\rangle + v_2 |1\rangle \tag{2.1}$$

with $v_1, v_2 \in \mathbb{C}$. To express the conjugate transpose of $|v\rangle$ we utilize the Bra notation:

$$\vec{v}^{\dagger} = \langle v| = \begin{bmatrix} \bar{v_1}, \bar{v_2} \end{bmatrix} \tag{2.2}$$

Given postulate 1, we also know that a qubit needs to be a unit vector, meaning that its elements need to obey the relation

$$\sum_i |v_i|^2 = 1$$

The coefficients for each basis can also be transformed into the probability of finding that base if the system is measured with the right observables. These probabilities are found by squaring the absolute values of the coefficients, e. g., $p(|0\rangle) = |v_1|^2$. Another well-known representation of a qubit can be obtained using a geometrical argument called the Bloch sphere [13].
We can represent a qubit as a point on the surface of a sphere with unitary radius where the poles represent the basis states, using properties of complex numbers: we can represent an imaginary number $z = a + ib$ we can on a Cartesian plane in the coordinate (a, b), we can then switch to a polar representation using the module $r = \sqrt{a^2 + b^2}$ and the angle ϕ. The complex number can now be rewritten as $z = r(\cos(\phi) + i\sin(\phi))$ that per Euler formula is $z = re^{i\phi}$. Using this representation of an imaginary number let's rewrite the qubit

$$|\phi\rangle = r_0 e^{\phi_0 i} |0\rangle + r_1 e^{\phi_1 i} |1\rangle$$

and since $r_0^2 + r_1^2 = 1$ this allows us to use r_0, r_1 as coordinate of a point belonging on the surface of a unitary sphere where $r_0 = \cos(\rho), r_1 = \sin(\rho)$. Here to reduce the modules to be dependent on the same parameter, we utilize $\rho = \theta/2$ with $0 > \theta > \pi$:

$$|\psi\rangle = \cos(\theta/2)e^{\phi_0 i}|0\rangle + \sin(\theta/2)e^{\phi_1 i}|1\rangle$$

by extracting the global phase e^{γ} and assigning $\psi = \phi_1 - \phi_0$ and $\gamma = \phi_0$ we obtain:

$$|\psi\rangle = e^{\gamma}(\cos(\theta/2)e^{i}|0\rangle + \sin(\theta/2)e^{\psi i}|1\rangle)$$

Since the global phase e^{γ} doesn't influence the results of a quantum measurement, we remove it from the formulation. To conclude, we can see that we can represent the qubit using two angles, and if we take one to be the angle of the rotation from the xy plane and the second to be a rotation from the z axis, we now have a Bloch sphere. This representation will be very useful to understand the application of the quantum gates that we will discuss ahead.

2.2.2 Quantum registers

While managing a single qubit might be interesting, the real power of quantum computation comes from utilizing a vast Hilbert space using multiple qubits at the same time in what is called a register. The definition for a quantum register is very broad, and it is typically defined as a collection of qubits. This can at times represent a set of qubits that are utilized during a computation to perform a task in order to differentiate them from another set, or sometimes it might refer to the entire population of qubits present inside a real quantum hardware architecture. While a classical register composed of n bits can contain 2^n different states, a quantum register allows us to explore a 2^n dimensional Hilbert space ($\mathbb{C}^{2^n}$) where every normalized vector is a possible state that our register can be in. A quantum register is generally represented as a vector obtained through the tensor product of single qubits:

$$|\psi\rangle = \bigotimes_{j=1}^{n} \alpha_i |i_j\rangle \iff |i_1, i_2, \ldots, i_n\rangle \tag{2.3}$$

This formulation is a direct application of the fourth postulate of quantum mechanics, and it's subjected to the normalization condition:

$$\sum_{j=1}^{n} |\alpha|^2 = 1 \tag{2.4}$$

The use of quantum registers allows us to exploit many properties specific to quantum states, like superposition and entanglement. The superposition effect can be seen in a single qubit as well, but it becomes really useful only when multiple qubits are involved. In the single qubit case, we can see that it goes into a superposition every time we take the basis coefficients $a, b \neq 0$. Here, the physical qubit exists in a combination (or superposition) of both basis states, and by interfering with the closed system through measurement, we obtain one or the other with a given probability.
Entanglement is instead the ability of two (or more) different qubits to have their state connected in such a way that that measuring (and collapsing) one influences the other's state.

2.2.3 Quantum gates

To change the state of a quantum register, we employ what are called quantum gates. These entities are the quantum equivalent of classical gates (NOT, AND,…) and can be applied to one or more qubits.

From the second postulate, we know that all quantum transformations U can only be unitary, meaning that for every U the relation $U^\dagger U = I$ must be true; they are mathematically represented by a square, unitary matrix. A quantum circuit consists of the application of multiple operations on a register and is often represented graphically like a timeline where the transformations are applied from left to right.

Opposite to classical computations, the quantum operators are infinite in number, and for this reason, some of them are considered the de facto standard. These gates are divided between single-qubit gates (Pauli gates, Hadamard gate, phase gates) and multi-qubit gates (Toffoli gate, controlled gates). The single qubit gates known as Pauli gates are those gates that rotate the state of single qubits: if we

picture the Bloch sphere representation of a qubit, we can imagine the Pauli gate PauliX to apply a π rotation over the x axis, PauliY over the y axis, and PauliZ over the z axis:

$$\sigma_0 = I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \rightarrow \quad \sigma_x = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\sigma_y = Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \sigma_z = Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

We can then exponentiate the Pauli gates using parameters that represent the rotation angle to obtain a set of gates that apply any rotation along each axis:

$$R_x(\theta) = e^{-i\frac{\theta}{2}\sigma_x} = \cos\left(\tfrac{\theta}{2}\right) I - i \sin\left(\tfrac{\theta}{2}\right) \sigma_x = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

$$R_y(\theta) = e^{-i\frac{\theta}{2}\sigma_y} = \cos\left(\tfrac{\theta}{2}\right) I - i \sin\left(\tfrac{\theta}{2}\right) \sigma_y = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

$$R_z(\theta) = e^{-i\frac{\theta}{2}\sigma_z} = \cos\left(\tfrac{\theta}{2}\right) I - i \sin\left(\tfrac{\theta}{2}\right) \sigma_z = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$$

Finally, we have three very important gates that appear very often: Hadamard, S, and T. The Hadamard gate applies two rotations and puts the basis state into states of superposition known as $|+\rangle$ and $|-\rangle$. S and T are special cases of the $R_z(\theta)$ gate that change the phase of the state.

$$H = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

A second class of quantum gates is the ones that are applied to multiple qubits at once. In this category, we can find controlled gates, swap gates, and the Toffoli gate. Controlled gates are a subset of gates that take the single qubit gates and add to them a control operation, with it the gates are applied only if the control qubit is in the state $|1\rangle$. Different from the classical *'if'* statement, here the control qubit can be simultaneously in both state 0 and 1, and the target qubit is

entangled to the control one based on the $|1\rangle$ state.
The Swap gate exchanges the state of two qubits. This operation is particularly important in quantum computation because of the *non cloning theorem* [14] that doesn't allow for copying a quantum state. Furthermore, modern quantum hardware doesn't have an all-to-all connection between qubits, so to perform two-qubit operations, the states need to move around the hardware to be close together.
Lastly, the Toffoli gate is a multi-controlled NOT operation. If the state of the two control qubits contains $|11\rangle$, the Not (σ_x) operation is applied to the target qubit. We can, of course, generalize this gate to apply any unitary operation, like with the controlled gates.

$$\text{CX} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad \text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{Toffoli} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

2.2.4 Entanglement

Quantum entanglement is the physical phenomenon that doesn't allow us to describe the state of a single unit of an entangled system without referencing the rest of the system, even when the single parts are separated. Let's now consider the mental exercise proposed by Einstein, Podolsky, and Rosen[15]:
We have two actors Alice (A) and Bob (B), they have one qubit each and while they are together they entangle the state with a simple circuit: They start from the state $|\psi_0\rangle = |00\rangle$ (and Alice applies an H gate to obtain a full superposition (

$|+\rangle$), the complete state can now be describes as

$$|\psi_1\rangle = |+\rangle |0\rangle = \frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |10\rangle$$

. Now they use a CNOT (Controlled X) gate on the second qubit using the first as control, the state of the second changes only if the state of the first is $|1\rangle$ so they obtain

$$|\psi_2\rangle = \frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle .$$

This particular state $|\psi_2\rangle$ is known as one of the EPR states from the authors of the exercise. Now, once the state is fully entangled, both of them start to travel and move a very long distance from each another. Once the travel is done Bob decides to measure his qubit, based on the formula for projective measurement (the kind of measurement that is most used in quantum computing) and on the operators $M_0 = I \otimes |0\rangle \langle 0| , M_1 = I \otimes |1\rangle \langle 1|$ Bob will obtain 0 with probability $p(0) = |1/\sqrt{2}|^2 = 0.5$. But if Bob obtains a collapse of the state to 0 with his measurement, this also means that Alice's qubit collapsed without any action on her part.

This instant phenomenon that collapses a qubit far away from the one that is measured bothered Einstein, who didn't believe it to be true. This mental experiment was itself a way to discredit the theory, but was later successfully performed experimentally [16].

In this example, we also find one of the practical uses of entanglement: quantum processes are not as straightforward as classical ones; most of the time, the answer we seek is encoded in the amplitude of the states or in some other aspect we cannot really measure. For this reason, some algorithms use an ancilla qubit entangled with the main register. The anchilla qubit is then measured during computation at specific points, and based on this result, we can understand if the process is valid or not, even before the full computation is done.

2.3 Quantum annealing

Quantum computing introduces a radically different way of processing information, based on the principles of quantum mechanics. Among the various models of quantum computation, adiabatic quantum computation (AQC) and quantum annealing (QA) have a unique approach to solving optimization problems. Unlike the gate-based model, which relies on discrete quantum gates applied in sequence following Postulate 2, adiabatic quantum computation (AQC) and quantum annealing (QA) are continuous-time models where the solution emerges from the evolution of a quantum system under a slowly changing Hamiltonian as described in Postulate 2'. These models are particularly interesting for computer scientists because they offer alternative strategies for tackling hard computational problems, especially those in the NP class.

The foundation of adiabatic quantum computation lies in the adiabatic theorem of quantum mechanics. This theorem states that if a quantum system is initially in the ground state of a Hamiltonian H_0, and the Hamiltonian is changed slowly enough to a final Hamiltonian H_1, then the system will remain in the ground state throughout the evolution, assuming the energy gap between the ground and excited states does not vanish. The evolution is governed by the time-dependent Schrödinger equation [17]. Here, the problem is encoded in the final Hamiltonian H_1, and its ground state represents the solution. The initial Hamiltonian H_0 is chosen to have a known and easy-to-prepare ground state. The system is then evolved from H_0 to H_1 over a time interval T, where at each moment

$$H(t) = (1 - s(t))H_0 + s(t)H_1$$

with s being monotonic functions from 0 to 1. If the evolution is slow enough, the system ends in the ground state of H_1.

This model is theoretically universal, meaning it can simulate any quantum circuit with polynomial overhead. However, its practical implementation is challenging due to the need for precise control over the Hamiltonian and the requirement of maintaining coherence over long time scales.

2.3.1 Basic principles of Quantum annealing

Quantum annealing is a practical approach that follows the properties of adiabatic quantum computation but is optimized for solving combinatorial optimization problems. The history of the quantum annealing process starts from metallurgy, where a material is heated and then cooled slowly to remove defects. From this process and with the inspiration of quantum behaviors, the classical algorithm of simulated annealing was created. Here, thermal fluctuations help the system escape local minima in the loss landscape. Finally, quantum annealing was formulated inspired by simulated annealing, but using quantum fluctuations introduced by a transverse field that plays a similar role to temperature, allowing the system to tunnel through energy barriers.

The typical form of the Hamiltonian used in QA is:

$$H(t) = A(t)H_{\text{init}} + B(t)H_{\text{final}}$$

Here, H_{init} is usually a transverse field Hamiltonian, such as $H_{\text{init}} = -\sum_i \sigma_i^x$, and H_{final} encodes the cost function to be minimized, often represented as an Ising model:

$$H_{\text{final}} = \sum_i h_i \sigma_i^x + \sum_{i<j} J_{ij} \sigma_i^z \sigma_j^z$$

The functions $A(t)$ and $B(t)$ control the annealing schedule, typically starting with $A(0) \gg B(0)$ and ending with $A(T) \ll B(T)$. The system begins in the ground state of H_{init}, which is easy to prepare, and evolves toward the ground state of H_{final}, which corresponds to the optimal solution.

2.3.2 Problem mapping

From a computer science perspective, the key step in using QA is to map a computational problem to a form suitable for quantum annealing. This usually involves expressing the problem as a Quadratic Unconstrained Binary Optimization (QUBO) problem or an Ising model. All NP-hard problems, such as Max-Cut, graph coloring, and satisfiability, can be formulated in this way since the QUBO

is an NP-hard problem itself.

For example, in the Max-Cut problem, the goal is to partition the vertices of a graph into two sets such that the number of edges between the sets is maximized. This can be encoded in an Ising Hamiltonian where each vertex is represented by a spin variable, and the couplings J_{ij} are derived from the adjacency matrix of the graph.

The advantage of QA is that it can explore the solution space using quantum tunneling, potentially escaping local minima more efficiently than classical algorithms. However, the success of this approach depends on the structure of the energy landscape and the annealing schedule.

Practically, this computation has a critical factor in the minimum energy gap between the ground state and the first excited state during the evolution. If this gap becomes too small, the system may undergo non-adiabatic transitions, jumping to excited states and failing to find the correct solution. The runtime required for adiabatic evolution scales inversely with the square of the minimum gap:

$$T \propto \frac{1}{\Delta^2}$$

where Δ is the minimum gap. For hard problems, this gap can shrink exponentially with system size, making the adiabatic evolution impractical. Understanding and estimating the gap is a major challenge in designing effective quantum annealing.

Furthermore, in real quantum annealers, such as those built by D-Wave Systems [18], the presence of noise and decoherence affects the performance. These systems operate at very low temperatures and use superconducting qubits, but they are still subject to environmental interactions that can cause the system to deviate from the ideal quantum evolution. As a result, the actual dynamics may include thermal relaxation and classical effects, complicating the interpretation of results.

Despite these limitations, quantum annealing remains a valuable tool for exploring quantum optimization. It provides insights into how quantum systems behave under complex energy landscapes and offers a platform for testing quantum algorithms in practice.

Chapter 3

Quantum Machine Learning

This chapter is dedicated to a brief introduction to classical and quantum machine learning. We will begin by describing various types of machine learning algorithm and their application, and then move to the three main parts of a quantum variational machine learning algorithm.

3.1 Classical machine learning

Machine learning is part of the big field of Artificial Intelligence (AI), where we consider an agent that can improve its performance on a task based on previous attempts. Traditionally, machine learning algorithms are divided into three different classes based on the updating techniques:

- *Unsupervised learning*: The agent understands the quality of its results without explicit feedback from the user or from the environment, by evaluating the solution on metrics that do not depend on the real distribution. The most well-known example is clustering.
- *Reinforcement learning*: The agent has a direct connection to the environment, and the choices it takes affect the surroundings, and the change in the environment functions as feedback to the model. One well-known example is models that learn to play a game by means of repeated failure.
- *Supervised learning*: This is the most common approach; the model compares

its output with the expected (true) output for that input. The errors guide the model through the training. Most deep learning models are trained with variations of this technique.

To better introduce the content of the next chapters, we will briefly give an explanation of how a machine learning model trains using a supervised approach.
First of all, let's give a formal definition of supervised learning, and then we will analyze the steps of this algorithm:

Definition 1 ***Supervised learning***
Given a set T of N example input-output pairs:

$$T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

where each y_j was generated by an unknown target function $f(x_j) = y_j$ the algorithm aims to discover a function $\hat{h}(x_j) = \bar{y_j}$ that approximates the function f, by minimizing a loss function $\mathcal{L}(y_j, \bar{y_j})$.

Here we call h a hypothesis function ($\hat{h}$ refers to the best hypothesis) and we measure its performance based on data not seen during training, called the test set.
As stated above, the aim of a supervised technique is to discover a function $\hat{h}$ that approximates the target. To do so the model starts from a parametric function $h(\theta, x)$ and updates the parameters θ until $h(\theta_{final}) = \hat{h} \approx f$.
To improve on its hypothesis and move from $h(\theta_0)$ to $\hat{h}$ models assume a relationship between x_j and y_j and uses the loss function $\mathcal{L}()$ to tailor the parameters to the problem.
Choosing the loss function is an arduous task in the more complex models since it is highly connected to the kind of problem we want to solve. For example, if we are solving a regression problem, the loss function might compute a metric based on the Euclidean distance between the predicted output and the true one. Instead, if we aim to solve a classification problem (a problem where $y \in Y$ with Y being a finite set of possible results), the loss function will take a different form since now the Euclidean distance loses its meaning.

The concept of loss function is not exclusive to supervised learning techniques, but it changes form based on the approach: Supervised techniques have a loss func-

tion that depend on the output and on the true result ($\mathcal{L}(y_j, h(x_j))$, Unsupervised techniques might want to minimize something that depends on the output and the input ($\mathcal{L}(x_j, h(x_j))$) and Reinforced learning algorithm will take in consideration the environment instead of input or output ($\mathcal{L}(env, h(x_j))$). Note that here we are using simplified definitions that cannot comprehend all the nuances of the field. That is because, unfortunately, a comprehensive and thorough description of classical machine learning algorithms would be too long and fall outside the scope of this work.

The process described above can be generalized for every type of machine learning approach. We start from a random parametric function and update the parameter using the value of the right loss function iteratively until we find a set of parameters that works well enough. Then, depending on the problem we aim to solve, different loss functions as well as different update rules (gradient-based, Bayesian, evolutionary,...) can be used.

3.2 Variational algorithms

The term *variational* can assume different meanings depending on the context it is used in. In our case, when we use the term variational algorithm, we refer to a hybrid algorithm made of both quantum and classical parts [19, 20]. This algorithm is among the most promising computing paradigms of the noisy intermediate scale quantum (NISQ) era, where we find ourselves today. The use of classical parts in the algorithms reduces the computational resources required from the quantum hardware as well as the precision required from the operations.
In this document, when discussing a quantum variational machine learning model, we refer to architectures divided into three main parts: *State preparation*, *Ansatz*, and *Parameter optimization*.

3.2.1 State preparation

We refer to the act of setting up the inputs of a model as the state preparation. This is similar to the preparation of the data into a fast sector of the memory before a classical model is run. Unlike a classical input quantum state, it has

some unique properties that limit quantum models. As an example, the non-cloning theorem tells us that a quantum state cannot be copied, meaning that we cannot run multiple times on the same input, but instead we need to recreate the state from the beginning, making the creation procedure part of the model. Furthermore, current hardware technologies don't allow for a long-term storage solution like the theoretical QRAM [21] and also require the computation to be incredibly fast to avoid the 'corruption' of the states during operations.

Here, it's worth specifying that the term state preparation encapsulates the strategies to obtain a state at a specific point of the circuit (traditionally, the starting point), without specifying whether the original data was quantum in nature or not. From this point forward, we will move the discussion to the problem of how to encode a classical data point in a quantum system, since this is the discussion that is most interesting from a computer science point of view. To analyze the advantages and disadvantages that quantum properties bring to the table, we will explore three of the most important state preparation techniques: Basis encoding, Angle encoding, and Amplitude encoding.

Basis encoding is the encoding strategy that most resembles a classical technique: If we start from a state $|\psi\rangle = |0\rangle^n$, we want to change the state of each qubit based on the binary representation of the number we want to encode. As an example, the number 13 is written in binary as 1101, then we will apply (quantum) NOT gates to the respective qubits to obtain the state $|1101\rangle$.
The resource cost of this technique in terms of the number of qubits increases with the $\log_2(K)$ where K is the number we want to encode. On the other hand, this approach has a depth cost in terms of the maximum number of gates applied to a qubit of 1.
This encoding strategy is not widely used in quantum machine learning, except for maybe the encoding of black and white images that can be represented as a binary number, but we can find it in what is the most influential quantum algorithm ever made, Shor's algorithm for discrete logarithm and prime factorization [10].

Angle encoding is the most used state preparation strategy because it balances the trade-off between resource and complexity. It utilizes rotation gates over the three main axes to rotate the qubit by an angle equal to the value we want to embed. These techniques allow us to encode up to three values for each qubit,

given values between 0 and $\pi/2$.

Amplitude encoding is the most famous approach that gives the high return that it promises. Using amplitude embedding we use N qubits to embed a data-point $X = \{x_1, \ldots, x_n\}$ with $x_j \in \mathbb{C}$ and $n = 2^N$. This is done by constructing a state $|\psi\rangle = \sum_{i=1}^{n} x_u |i\rangle$ where each x_i becomes the amplitude of the basis state $|i\rangle$. In theory, this strategy allows us to store an exponential amount of information in a given number of qubits. Practically, there are two big limitations: first of all, $|\psi\rangle$ has to adhere to the normalization rule, meaning that $\sum_{i=1}^{n} |x_i|^2 = 1$. When the number of qubits increases, the values of the amplitude we store diminish exponentially in magnitude.
Secondly, the practical implementation of this encoding strategy is not fixed and needs to be computed for each data point we want to embed. Furthermore, the depth of the encoding circuit tends to grow exponentially with the number of qubits used, countering the advantages we were able to get in terms of memory.

3.2.2 Ansatz

The term Ansatz in quantum machine learning refers to a circuit or sub-circuit made partially or entirely by parametric quantum gates. The ansatz is the quantum counterpart to the classical neural network architecture since the parameters are optimized during the training phase in order to minimize a loss function.
Ansatz architectures can be grouped based on the basic structures of the gates, taking different names: Layered gate, alternating operator, tensor networks.
besides this classification, we can also distinguish the architectures based on the aim of the structure: Hardware efficient, qubit efficient, problem-inspired, and classically inspired.

Layered gate: Since every operation in quantum computing is a unitary operation, we can group many operators to describe a bigger one. Following this idea, we can devise composite gates that will be applied multiple times. In the layered gate architectures, two different composite gates are created and applied alternatively. The first one is a composition of single-qubit operations that aim to rotate the state to a more favorable position, while the second one is a collection of multi-qubit gates that entangle the state. At least one of the two needs to contain

parametric gates.
Multiple repetition of this composite gate creates the most commonly used structure for basic experiments in quantum machine learning.

Alternating operators Similar to the previous one, the alternating operators architecture is composed of two different gates that are repeated one after the other. The difference is that, while in the layered gates we had a composite gate made of smaller ones that depend on the specific model, here the gates represent a parametric Hamiltonian that describes specific aspects of the problem [22].
The most famous architecture that belongs to this group is the Quantum approximate optimization algorithm that solves a QUBO problem by repeatedly applying two Hamiltonians that represent a cost Hamiltonian H_c and a mixer Hamiltonian H_m.

Tensor networks This type of architecture also belongs to the classically inspired architecture since it draws inspiration from (classical) Tensor Networks. It consists of a fixed structure with efficient local entanglement and a low-depth circuit. It follows a different optimization approach, and instead of optimizing on the entire state vector, these models work on a network of smaller state vectors. This allows a better expressibility for low entangled states, an easier training process, and an increased interpretability.

We will not focus on the other definitions for the ansatz, except that, for the most part, they are not relevant to this work, except 'Classically inspired':

Classically inspired: Architectures belonging to this class do not share a common circuit structure of gates, but instead are all created to replicate and improve upon existing classical architectures. In this work, many proposals will have a classical-inspired formulation that translates into many different quantum circuit structures. This approach aims to find models in classical machine learning algorithms that can gain some advantages when the architecture or part of it is reformulated into the quantum counterpart. This advantage can be in terms of space, time, or complexity, and highly depends on the formulation of the quantum version.

3.2.3 Parameter optimization

The last aspect that we need to describe to conclude an introduction to quantum machine learning is the idea of parameter optimization. As we discussed above, a quantum machine learning model makes use of parametrized gates, mainly for two purposes: Classical data encoding in the state preparation phase, and parametric-based operations in the ansatz. While in the state preparation, the parameters depend only on the architecture and on the data we want to encode in the ansatz; these parameters are guided to transform the input they receive into the desired output.
Exactly like in classical machine learning, we utilize a cost function and a loss function to change the parametric rotations. The loss function, in particular, is computed and used in an optimizer. Two aspects of this approach are important: firstly, the optimization part and the optimizer are fully classical approaches; the only difference is that for a gradient-based optimizer, the computation of the gradient is relatively complex and needs to be computed using the gradient shift rule [23]. Non-gradient-based approaches work exactly the same in classical and quantum.
The second aspect we need to keep in mind is that the loss function can be computed in numerous ways since it also encapsulates the measurement.
Quantum measurement, as discussed above, is a very complex issue that literally destroys the state computed with a nonlinear operation. This must be taken into account when deciding which loss function is correct for the problem. On the other hand, we could decide not to use the measurement of the model but its state vector for some problem that can be simulated and for which the state is known. In other instances, multiple measurements could be used to approximate the state, even for real quantum architectures where a complete state read might be impossible. Many other approaches exist, like quantum Earth Mover distance [24] or classical shadows [25] (two promising approaches to quantum loss functions), but a complete description of the use of measurement as a quantum machine learning loss function would be too extensive and would fall out of the scope of this work.

In general, we can confidently say that the optimization of the parameters for a quantum machine learning architecture is as complex to set up and run as it is on

classical architectures. All classical optimizers are available and the loss function needs to take into consideration measurement techniques, but as it happens in classical machine learning, the right choices might drastically change the results of the training and the overall ability of the model to perform the task it was assigned to do.

Chapter 4

State of the art

In this chapter, we will present an analysis of the state of the art for quantum variational algorithms. Here, we will describe the environment, the subjects, and the findings of the most relevant research connected to the topic of this work. We will start with the gate-based quantum neural network field, talking about how modern quantum machine learning approaches often come from classical methods. In the second part, we will describe the field of quantum annealing and adiabatic-based optimization, along with a discussion of the current technology.

4.1 Gate based quantum computation

In this section, we will focus our attention on a literature review on the topic of quantum gate-based architectures. We will first discuss basic neural networks and later three different state-of-the-art architectures that are closely related to the work presented in Chapter 5.

4.1.1 Quantum neural networks

The first and basic architecture for QML is called the Quantum neural network (QNN). As discussed above, these circuits can be considered as the composition of multiple layers of connected computational units controlled by trainable parameters. This definition is similar to the characterization of neural networks, but the comparison between the two poses several challenges. In fact, classical neural

networks require a non-linear activation function, which is a limit for quantum computation since quantum operations are unitary and linear. Furthermore, it is impossible to access the quantum state at intermediate points during computation. Thus, backpropagation [26], which is the standard to work on large-scale models, is harder to use for quantum circuits training.

In practice, several attempts to build a quantum neural network are discussed in the literature. Although a potential quantum advantage of quantum neural networks over classical ones has been investigated [27], to the best of our knowledge, there are no trainable quantum algorithms that can effectively implement a complete neural network in a quantum setting. A concrete implementation of a quantum circuit to solve a typical ML task using a near-term processor is illustrated in [28], where the authors introduced a model for binary classification using a modified version of the perceptron updating rule. Another approach is represented by Tensor Networks, inspired by quantum many-body physics. Tensor networks [29] are methods to represent quantum states whose entanglement is constrained by local interactions. This approach enables the numerical treatment of systems through layers of abstraction, as for deep neural networks. Some of the most studied tensor networks have been adopted for classification and generative modeling [30, 31]. In the past years, the idea of building a parametrized quantum circuit to generate a quantum state equivalent to the output of a two-layer neural network has been proposed [1]. However, the generalization to more than two neurons is not provided.

In general, the standard approach for quantum neural networks consists of building a quantum circuit on top of existing approaches for variational circuits, which solve a supervised learning task [32]. Methods within this approach take care of the supervised task end-to-end, where the input quantum state is fed through a parametrized unitary operator and the final measurement provides the estimate of the target variable of interest.

Alternatively, quantum support vector machines (QSVM) [33] use a quantum circuit to map classical data into a quantum feature space. The resulting features are then fed into a classical linear model, which calculates the final output. A more in-depth analysis of this architecture will be presented later in this chapter.

4.1.2 Activation function for quantum neural network

Regarding the main obstacle to quantum neural networks, the curse of linearity, recently, several attempts have been made to provide a routine for quantum activation functions to overcome the constraint of the unitarity of quantum operations. The quantum Splines (QSplines) [34] rely on classical B-Spline regression models that aim to find the optimal set of parameters to minimize the Residual Sum of Squares in a ridge regression problem where observed variables are augmented with polynomials. However, the QSplines suffer from several drawbacks and limitations: the use of the HHL [35] as a subroutine imposes running the algorithm on a perfectly error-corrected quantum computer, and it is not suitable to be executed on near-term quantum devices. Moreover, The output of the QSplines requires a post-processing step to obtain the value of the non-linear function, which is stored in the quantum state. Furthermore, QSplines require ad-hoc formulation for the basis expansion matrix in terms of a diagonal block matrix that is hardly generalizable.

Other works on quantum activation functions rely on repeat-until-success technique [36]. In this case, the most significant limitation is that the input must be in the range $\left[0, \frac{\pi}{2}\right]$, which is a severe constraint for real-world problems. A step forward was made in [37], where domain restrictions were removed, and the activation gate parameters were learned during training. Recently, the problem of non-linear approximation has been considered by means of a Quantum Perceptron [38]. The proposed quantum algorithm produces the output of a non-linear activation function, leveraging an iterative computation of all the powers of the inner product up to an order d. The main drawback of this approach is the number of qubits required, which depends linearly on the number of input features and the order of the polynomial. in particular, given an n-dimensional feature vector and a degree of the polynomial d, the idea of the Quantum Perceptron requires $n + d$ qubits. Experimentally, this approach shows good results with a degree of polynomial $3 \leq d \leq 10$ for 1 1-dimensional feature vector. Finally, parametrized quantum circuits have been used to achieve non-linear approximation by means of hybrid quantum-classical optimization [39]. However, this approach relies on utilizing multiple copies of variational quantum states to treat nonlinearities, and its

use in the context of quantum neural networks is unclear. Furthermore, the experiments approximate Schrödinger equation only and do not address the estimation of typical activation functions.

4.1.3 Quantum approximate algorithm and quantum hardware benchmarking

QAOA The Quantum Approximate Optimization Algorithm (QAOA) [22] is a variational algorithm designed to solve combinatorial optimization problems. It is particularly suited for Noisy Intermediate-Scale Quantum (NISQ) devices, and has become a central focus in the development of near-term quantum algorithms.

QAOA's ansatz is part of the alternating operators class, and it doesn't present a state preparation phase; it operates by alternating between two unitary operators: one derived from the cost Hamiltonian, which encodes the objective function of the optimization problem, and another from the mixer Hamiltonian, which promotes exploration of the solution space. These operators are applied in layers, with the number of layers p controlling the depth and expressiveness of the quantum circuit. The quantum state produced by the circuit is measured, and the resulting bitstrings are evaluated classically to guide parameter optimization.

However, the practical implementation of QAOA faces challenges. Chief among them is the optimization of variational parameters, which can be hindered by barren plateaus—regions in the cost landscape where gradients vanish exponentially with system size [40]. Recent work has shown that the presence of barren plateaus depends on the system's architecture, which can be analyzed using the dynamical Lie algebra generated by the ansatz operators. Several strategies have been proposed to enhance QAOA's performance, like combinations between QAOA and the Generator Coordinate Method (GCM)[41] or Layer-wise training and warm-starting [42], and customizing the mixer Hamiltonian to better reflect the structure of the problem [43]

In this work, we utilize the QAOA algorithm to perform a comparative analysis on different architectures. This comparison will be presented in terms of the ability and the time the machines take to solve the same problems.
Benchmarking quantum devices is essential for evaluating their capabilities and

understanding their limitations, particularly in solving complex computational problems. Numerous frameworks and methodologies have been developed over the years to assess the performance of both gate-based quantum computers and quantum annealers. Early contributions to benchmarking came from Knill [44] and Magesan [45], who developed scalable and robust randomized benchmarking techniques that allowed for a comprehensive evaluation of quantum gate fidelity. Gambetta introduced simultaneous randomized benchmarking to assess qubit addressability, a crucial aspect for large-scale quantum computing [46].

In the context of specific tasks, such as NP-complete problems, Boixo characterized quantum supremacy in near-term devices, providing critical insights into their capabilities for solving computationally hard problems [47]. Cross et al. advanced the field by validating quantum computers using randomized model circuits, ensuring that performance benchmarks could be applied to real-world tasks like graph coloring [48]. Erhard further contributed with cycle benchmarking, aimed at characterizing large-scale quantum computers [49].

More recent works have shifted toward application-oriented benchmarks. Lubinski [50] and Martiel et al. [51] focused on evaluating quantum devices based on their performance on practical problems, such as optimization tasks, which directly apply to graph coloring. Similarly, Pelofske compared three generations of D-Wave's quantum annealers, providing valuable insights into their evolution and performance on combinatorial optimization problems [52].

The introduction of comprehensive benchmarking frameworks has also played a pivotal role. Blume-Kohout and Young proposed a volumetric framework that evaluates quantum computers based on the volume of circuits they can process, making it applicable to devices like IBM's gate-based systems [53]. Finzgar et al. [54] introduced the QUARK framework for benchmarking quantum computing applications, while Quetschlich [55] developed MQT Bench, focusing on benchmarking quantum software and design automation tools. Barbaresco introduced the BACQ framework, an application-oriented benchmarking system, which provides a comprehensive evaluation for a range of quantum computing tasks, including NP-complete problems [56].

In parallel, domain-specific quantum algorithms related to graph coloring have also emerged. More recent studies, such as Clerc [57], proposed a quantum method

for solving the graph k-coloring problem.

For gate-based quantum computers such as IBM's general-purpose quantum devices, the methodology involves more flexible circuit design techniques. Quantum circuits can be explicitly constructed to address the graph coloring problem by encoding the solution space into qubit states and using quantum gates to represent constraints and objective functions. Additionally, the QAOA [22] is employed as a key optimization method. QAOA is used to optimize the quantum circuits, refining the solution quality by variationally adjusting the parameters of the circuit to minimize the cost function associated with the problem.

4.1.4 Swap test and quantum dimensionality reduction

In the field of quantum computing, the SWAP test is a foundational component for various tasks, including the estimation of quantities like $\mathrm{tr}(\rho^n\sigma)$, where ρ is a density matrix and σ is an observable [58, 59]. These circuits are a multipartite generalization of the SWAP test [60]. A key development in the SWAP test was the discovery of a method to reformulate the circuit without using an ancilla qubit, which is particularly beneficial for applications like cross-platform verification [61]. This reformulation allows for parallelization and constant circuit depth. This simplified circuit can be obtained using known circuit identities or through more exhaustive search methods [62]. A fundamental principle behind the SWAP test is its equivalence to the Hong-Ou-Mandel effect in quantum optics [63].

A related approach to reducing the cost of such operations is to first perform a quantum dimensionality reduction on the states. One proposed method for this is the use of a Quantum Convolutional Neural Network (QCNN) to compress each quantum state ρ. This reduction in dimensionality allows the use of a permutation test (gate D) with a lower resource cost [64, 65].

An alternative and increasingly researched solution for dimensionality reduction is the use of a quantum autoencoder (QAE). Inspired by their classical counterparts, QAEs are a type of variational quantum algorithm that learns an efficient, compressed representation of quantum data by encoding and then attempting to decode it.

The primary objective in a QAE is to find a unitary transformation that com-

presses a quantum state into a smaller subspace while leaving errors and redundant information in a separate trash space [66, 67, 68]. This trash space, which can have a known form like $|0\rangle^{n-m}$, is then effectively discarded. This approach is most effective when the family of states can be compressed efficiently, possibly with minimal or no information loss [69, 70, 71]. The training process, which seeks to minimize the information lost, can be complex, but recent research has focused on optimizing this.

Most recent literature highlights that QAEs are moving beyond just data compression to serve as powerful tools for error mitigation and noise-reduction [72, 73]. This is especially critical for NISQ (Noisy Intermediate-Scale Quantum) devices, where noise is a significant challenge. By training a QAE on noisy data in an unsupervised manner, it can learn to produce ideal, noise-free output states [74]. This capability to denoise complex entangled states, such as GHZ and Dicke states, is a significant advancement [74]. Some studies have also explored more complex "brainbox" architectures within the QAE's bottleneck to improve denoising performance on stronger noise channels [73].

Furthermore, recent work has explored more sophisticated QAE applications, such as their use in quantum machine learning for image classification and even for learning quantum channel codes [75, 76]. The ability of a QAE to capture complex correlations and reduce resource requirements makes it a promising component for more complex quantum applications. The minimum loss function for training can be computed following established methods [70], and new work continues to explore novel training objective functions and circuit structures to enhance the model's robustness and efficiency [77].

4.1.5 Quantum kernel methods

In the field of QML, one of the foundational models is the so-called Quantum support vector machines [78] or more correctly quantum kernel methods. This model is based on the idea of using the exponential dimensionality of the Hilbert space in order to create a linear separation between classical data in the same way as Kernel methods are used. More specifically, Schuld and Killoran [79] showed the equivalence of this phase to a quantum kernel and proposed one of the initial

gate-based Support Vector Machine models (QgSVM). Their work outlines two approaches: firstly, they introduce the quantum kernel as a means to process classical data, transforming it into new data with different dimensionality analogous to a classical kernel, which is then utilized in classical algorithms such as SVM. Secondly, they propose a parametric circuit for classifying input in a quantum environment. This feature map/ansatz approach has become standard for Quantum Neural Networks (QNNs).

Research by Maria Schuld [80] aggregates results from various sources and concludes that all supervised quantum machine learning models (excluding generative ones) operate as kernel methods.

Another noteworthy contribution to the state of the art in gate-based QSVMs is the work by Havlicek et al. [33]. They introduce two SVM classifiers optimizing classically over data obtained using a quantum kernel trick to achieve a theoretical quantum advantage. Similar to previous approaches, this involves non-linearly mapping data to a quantum state.

The authors also highlight a crucial requirement for applying QgSVM: if the feature vector kernel $K(\tilde{x}, \tilde{z}) = |\langle\Phi(\vec{x})|\Phi(\vec{z})\rangle|^2$ is overly simplistic, such as generating only product states, it can be easily computed classically and loses its benefit. The advantage lies in leveraging the high dimensionality of the Hilbert space, necessitating a kernel that is impossible to simulate to surpass classical approaches.

4.2 Quantum annealing

Here we will present works of literature related to chapter 6. While key concepts from the quantum annealing optimization field are presented in section 6.3.2, this section will present a literature review of the topics related to chapter 6 and how this has been explored in the context of quantum annealing.
Unfortunately, while this field of research looks very promising, quantum to machine learning approaches are way less explored in the field of quantum annealing with respect to their gate-based counterparts.

4.2.1 Quantum annealing support vector machines

In section 4.1.5, we discussed how gate-based computation can be utilized to perform the kernel trick for a support vector machine, but we also saw how the main part of the algorithm (the computation of the separating hyperplane) remains a fully classical procedure. With a quantum annealer, we possess the ability to perform this optimization in a quantum fashion.

Quantum annealers efficiently solve problems formulated in either Quadratic Unconstrained Binary Optimization (QUBO) or Ising formulations.

Since we know how to formulate Support Vector Machines as convex quadratic optimization problems, a simple transformation is enough to achieve a QUBO formulation suitable for quantum annealing.

In a study by Willsch et al. [81], the authors introduced and explored the application of kernel-based Support Vector Machines on a DW2000Q quantum annealer [82]. From here on out, we refer to this methodology as Quantum Annealer Support Vector Machines (QaSVM). This approach offers distinct mathematical advantages, generating a spectrum of different classifiers with each iteration of the training process.

The study's findings demonstrate that the ensemble of classifiers produced by the quantum annealer can surpass the single classifier derived from classical SVMs when addressing the same computational problem. Performance is assessed through metrics such as Area Under the Receiver Operating Characteristic curve (AUROC), Area Under the Precision–Recall curve (AUPRC) [83], and accuracy. This advantage is attributed to the quantum annealer's ability to yield not only the global optimum for the training dataset but also a distribution of solutions close to optimality for the given optimization problem. The potential to combine these solutions enhances generalization to the test dataset.

Additional studies [84, 85] corroborate the efficacy of QaSVMs and extend their promise to diverse problems, including multiclass classification.

4.2.2 K-means and annealing

Clustering is an old but incredibly pervasive problem in computer science. In this section 6.1, we will discuss a variational algorithm that improves upon the classical

K-means. K-means tries to solve the clustering problem by placing the centroids and iteratively improving their positions. The algorithm depends a lot on how the initial centroids are selected, and if the positions are not good, the final clusters may be wrong. Quantum K-means uses quantum superposition and parallelism to test many centroid positions at the same time. This helps the algorithm to find better solutions and avoid local minima.

One clear difference in the approach is the use of quantum similarity metrics. In classical clustering, we use distances like Euclidean or cosine. But in quantum clustering, we can use quantum fidelity, which measures how similar two quantum states are. A hybrid method that combines quantum and classical metrics was proposed [86]. This method gives more stable results, especially when the data is noisy. Quantum K-means also benefits from faster sampling techniques. For example, Chen et al. [87] showed that using uniform quantum sampling can reduce the time complexity from linear to logarithmic. This means the algorithm can work faster with large datasets.

Another interesting approach is using adiabatic quantum computing (AQC) and quantum annealing. Zaech et al. [88] used AQC to solve balanced K-means problems. Their method uses probabilistic sampling to estimate the best cluster assignments and also gives uncertainty estimates. In general, studies like Jain et al. [89] show that QK-means can be faster and more accurate than classical K-means, especially with high-dimensional data. Reviews [90] confirm that clustering is one of the areas where quantum machine learning can give real advantages. Even if the results are promising, there are still challenges. Quantum hardware is limited, and algorithms are complex. But hybrid models and quantum-inspired methods are good options for now and can be used in real applications.

4.2.3 Unit commitment problem and annealing

The Unit Commitment problem (UC) is a well-known problem in the context of energy engineering and optimization [91]. Generally described as a family of optimization problems where the energy production of some generators is coordinated based on an objective function, typically the minimization of costs or maximization of revenue. There exist many different versions that take into consideration

one or multiple specifics, depending on the practical case study considered. The problems in this family present some common elements such as generators, consumption forecast, rules or laws (physical or legislative) [92, 93, 94], error tolerance, and a temporal horizon. In literature, there are many classical and quantum techniques that solve this problem with varying degrees of success [95, 96, 97].
Given the high importance of this topic and given that this problem is categorized as an NP-hard mixed-integer optimization problem, different approaches have been proposed to improve in solution quality or computational time, like exhaustive enumeration (brute force), priority listing dynamic programming, branch and bound, and many others [98].
Since the problem is vast and we focused only on some facets, a complete description of the instance and its formulation will be discussed in the related section 6.2

Part II

Contributions

Chapter 5

Gate based approaches

5.1 Quantum Single layer perceptron

5.1.1 Preliminaries

Neural Network as Universal Approximator

A Single Layer Perceptron (SLP) [99] is a single-layer neural network suitable for classification and regression problems. Given a training point $(\mathbf{x}, y)$, the output of a SLP containing H neurons can be expressed as:

$$f(\mathbf{x}; \beta, \theta) = \sigma_{\text{out}} \left(\sum_{j=1}^{H} \beta_j \sigma_{\text{hid}} \left[L(\mathbf{x}; \boldsymbol{\theta}_j) \right] \right) = \sum_{j=1}^{H} \beta_j g(\mathbf{x}; \theta_j). \tag{5.1}$$

where σ_{out} is the identify function in case of a continuous target variable y. Each hidden neuron j computes a linear combination, $L(\cdot)$, of the input features $\mathbf{x} \in \mathbb{R}^p$ with coefficients given by the p-dimensional vector $\boldsymbol{\theta}_j$. This operation is performed for all neurons, and the results are individually fed into the inner activation function σ_{hid}. The outputs of the previous operation are then linearly combined with coefficients β_j. Finally, a task-dependent outer activation function, σ_{out}, is applied.

Despite being less utilized than the deep architectures, the SLP model can be very expressive. According to the *universal approximation theorem* [100], a SLP with an arbitrarily large number of hidden neurons and a non-constant, bounded and continuous activation function can approximate any continuous function on a

closed and bounded subset of $\mathbb{R}^p$. In spite of this crucial theoretical result, the SLP are rarely adopted in practice due to the infeasibility of training large amounts of hidden neurons on classical devices. Quantum computers, however, could leverage the superposition of states to scale the number of hidden neurons exponentially with the number of available qubits. Starting from these considerations, cleverly implementing a quantum SLP would therefore enable a real chance to benefit from the universal approximation property.

Activation function

The implementation of a proper activation function, in the sense of the Universal Approximation Theorem, is one of the major issues for building a complete quantum neural network. Recently, the idea of quantum splines (QSplines) [34] has been proposed to approximate non-linear activation functions by means of a fault-tolerant quantum algorithm. Although a QSpline provides a method to store the value of a non-linear function in the amplitudes of a quantum state, it uses the HHL [35] as subroutine, a quantum algorithm for matrix inversion which imposes several practical limitations [101].

In this work, we do not discuss how to implement in practice a non-linear activation function. Nevertheless, we provide a framework that permits to train a quantum SLP with an exponentially large number of hidden neurons for a given activation function Σ. Furthermore, our architecture is naturally capable of incorporating new implementations of non-linear activation functions as long as they fit in the learning paradigm.

From classical to quantum SLP

Gates as Linear Operators. As discussed in Sec. 5.1.1, an SLP applies multiple linear combinations on the same input based on different sets of parameters. In case of qSLP, we can consider the following parametrized quantum gate to map a generic input quantum state any possible other quantum states:

$$U_3(\boldsymbol{\theta}) = U_3(\alpha, \beta, \gamma) = \begin{pmatrix} e^{i\beta}cos(\alpha/2) & e^{i\gamma}sin(\alpha/2) \\ -e^{-i\gamma}sin(\alpha/2) & e^{-i\beta}cos(\alpha/2) \end{pmatrix}. \tag{5.2}$$

The unitary $U_3(\boldsymbol{\theta})$ represents a bounded linear operation where the set of parameters vector $\boldsymbol{\theta} = \{\alpha, \beta, \gamma\}$ can be learned using classical optimization procedures. Importantly, the adoption of $U_3(\boldsymbol{\theta})$ as linear operator implies transforming the input data using complex coefficients. Therefore, it describes a more general operation with respect to the classical SLP, that only allows for linear combinations with real-valued coefficient. Nonetheless, one can still parametrize the circuit using Pauli-Y rotation to restrict the computation to the real domain.

Ansatz for Linear Operators in Superposition. The original proposal of the qSLP [1] creates entanglement between the *control* and the *data* register using two CSWAP operations and an additional *temp* register to generate a superposition of two different linear transformations. Although this approach allows obtaining the desired result, it implies the adoption of twice the number of qubits to encode the input data (*temp* register). Furthermore, using the CSWAP gates introduces a linear overhead with respect to the size of the two registers *data* and *temp* [102], in terms of gate complexity. To reduce such complexity, we propose a different approach (illustrated in Figure 5.1) which does not require the additional *temp* register and the use of the CSWAP gates.

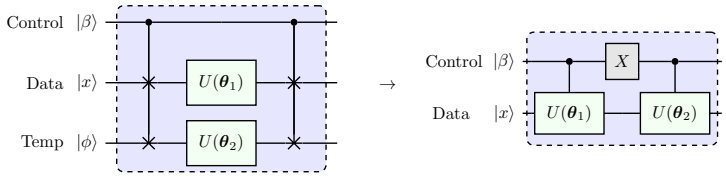


Figure 5.1: Alternative ansatz for linear operators in superposition. On the left the quantum gates adopted in the first version of the two-layer qSLP [1]. On the right the proposed schema to generate two different linear operation in superposition each entangled with one of the basis states of the *control* qubit.

Quantum Single Layer Perceptron

Intuitively, a generalized qSLP can be implemented into five steps: *state preparation*, *entangled linear operators in superposition*, *application of the activation function*, *measurement step*, *post processing optimization*. To this end, two quantum registers are necessary: *control* (d qubits), *data* (n qubits).

Step 1: State Preparation. The *control* register is turned into a non-uniform superposition parameterized by the d-dimensional vector $\hat{\boldsymbol{\beta}}$ by means of d Pauli-Y rotation gates. Also, a single p-dimensional training point is encoded into the n-qubit *data* register through the quantum gate S_x:

$$\begin{aligned} |\Phi\rangle_{\text{SP}} &= \left(\bigotimes_{i=1}^{d} R_y(\hat{\beta}_i) \otimes S_x \right) |0\rangle_{\text{control}}^{\otimes d} \otimes |0\rangle_{\text{data}}^{\otimes n} \\ &= \bigotimes_{i=1}^{d} \left(a_i |0\rangle + b_i |1\rangle \right) \otimes |x\rangle = \bigotimes_{i=1}^{d} |c_i\rangle \otimes |x\rangle , \end{aligned} \tag{5.3}$$

where $\{a_i, b_i\}$ are real numbers such that $|a_i|^2 + |b_i|^2 = 1$. Importantly, the algorithm is completely independent by the encoding strategy chosen for S_x.

Step 2: Entangled Linear Operators in Superposition. The second step generates a superposition of 2^d linear operations with different parameters, each entangled with a basis state of the *control* register. In particular, once the two registers are initialized, each qubit in the *control* register is entangled with two different random transformations of the *data* register. As a result, 2^d different transformations in superposition of the input are generated. Here we describe the procedure assuming 3 *control* qubits where the ansatz described in Sec. 5.1.1 is applied only $d = 3$ times in order to obtain $2^d = 8$ different parametrized unitary transformations of the input in superposition.

Step 2.1 ($d = 1$) The first step applies the unitary $U(\boldsymbol{\theta}_{1,1}, \boldsymbol{\theta}_{1,2})$ to the initialized quantum system, which is defined as follows:

$$U(\boldsymbol{\theta}_{1,1}, \boldsymbol{\theta}_{1,2}) = \left[\mathbb{1}^{\otimes d-1} \otimes C\text{-}U(\boldsymbol{\theta}_{1,1}) \right] \left[\mathbb{1}^{\otimes d-1} \otimes X \otimes \mathbb{1} \right] \left[\mathbb{1}^{\otimes d-1} \otimes C\text{-}U(\boldsymbol{\theta}_{1,2}) \right]$$

where $\mathbb{1}$ is the identity matrix and X is the NOT gate (i.e., Pauli-X gate). Thus, the first step ($d = 1$) leads to the following quantum state:

$$\begin{aligned} |\Phi\rangle_{(d=1)} &= U(\boldsymbol{\theta}_{1,1}, \boldsymbol{\theta}_{1,2}) |\Phi\rangle_{\text{SP}} \\ &= \bigotimes_{i=1}^{2} |c_i\rangle \otimes (a_1 |1\rangle U(\boldsymbol{\theta}_{1,2}) |x\rangle + b_1 |0\rangle U(\boldsymbol{\theta}_{1,1}) |x\rangle). \end{aligned} \tag{5.4}$$

The basis states of the first *control* qubit is then entangled with two parametrized

unitary transformations of the *data* register.

Step 2.2 ($d = 2$) The second step employs another *control* qubit and other two sets of parameters $\boldsymbol{\theta}_{2,1}$ and $\boldsymbol{\theta}_{2,2}$. Then the same procedure is applied through the unitary operations $U(\boldsymbol{\theta}_{2,1}, \boldsymbol{\theta}_{2,2})$:

$$\begin{aligned}
|\Phi\rangle_{(d=2)} &= U(\boldsymbol{\theta}_{2,1}, \boldsymbol{\theta}_{2,2}) \, |\Phi\rangle_{(d=1)} \\
&= \frac{1}{\sqrt{4}} \Big[b_2 a_1 \, |00\rangle \, U(\boldsymbol{\theta}_{2,1}) U(\boldsymbol{\theta}_{1,1}) \, |x\rangle + b_2 b_1 \, |01\rangle \, U(\boldsymbol{\theta}_{2,1}) U(\boldsymbol{\theta}_{1,2}) \, |x\rangle \\
&\quad + a_2 a_1 \, |10\rangle \, U(\boldsymbol{\theta}_{2,2}) U(\boldsymbol{\theta}_{1,1}) \, |x\rangle + a_2 b_1 \, |11\rangle \, U(\boldsymbol{\theta}_{2,2}) U(\boldsymbol{\theta}_{1,2}) \, |x\rangle \Big]
\end{aligned} \tag{5.5}$$

with $U(\boldsymbol{\theta}_{2,1}, \boldsymbol{\theta}_{2,2}) = [\mathbb{1} \otimes C \otimes \mathbb{1} \otimes U(\boldsymbol{\theta}_{2,1})] \, [\mathbb{1} \otimes X \otimes \mathbb{1}^{\otimes n+1}] \, [\mathbb{1} \otimes C \otimes \mathbb{1} \otimes U(\boldsymbol{\theta}_{2,1})]$. The position of the C gate indicates the *control* qubit used to perform the controlled operations C-$U(\boldsymbol{\theta}_{2,1})$ and C-$U(\boldsymbol{\theta}_{2,2})$. As a consequence of this second step, four parametrized transformations of the input $|x\rangle$ are generated, each results from the product of two $U(\boldsymbol{\theta}_{i,k})$ gates, for $i, k \in \{1, 2\}$. Furthermore, these transformations are entangled with the basis states of the two *control* qubits.

Step 2.3 ($d = 3$) We perform the same procedure for the third *control* qubit, using the unitary $U(\boldsymbol{\theta}_{3,1}, \boldsymbol{\theta}_{3,2})$:

$$\begin{aligned}
&|\Phi\rangle_{(d=3)} = U(\boldsymbol{\theta}_{3,1}, \boldsymbol{\theta}_{3,2}) \, |\Phi\rangle_{(d=2)} \\
&= \frac{1}{\sqrt{8}} \Big[\beta_1 \, |000\rangle \, U(\boldsymbol{\theta}_{3,1}) U(\boldsymbol{\theta}_{2,1}) U(\boldsymbol{\theta}_{1,1}) \, |x\rangle + \beta_2 \, |001\rangle \, U(\boldsymbol{\theta}_{3,1}) U(\boldsymbol{\theta}_{2,1}) U(\boldsymbol{\theta}_{1,2}) \, |x\rangle \\
&\qquad + \beta_3 \, |010\rangle \, U(\boldsymbol{\theta}_{3,1}) U(\boldsymbol{\theta}_{2,2}) U(\boldsymbol{\theta}_{1,1}) \, |x\rangle + \beta_4 \, |011\rangle \, U(\boldsymbol{\theta}_{3,1}) U(\boldsymbol{\theta}_{2,2}) U(\boldsymbol{\theta}_{1,2}) \, |x\rangle \\
&\qquad + \beta_5 \, |100\rangle \, U(\boldsymbol{\theta}_{3,2}) U(\boldsymbol{\theta}_{2,1}) U(\boldsymbol{\theta}_{1,1}) \, |x\rangle + \beta_6 \, |101\rangle \, U(\boldsymbol{\theta}_{3,2}) U(\boldsymbol{\theta}_{2,1}) U(\boldsymbol{\theta}_{1,2}) \, |x\rangle \\
&\qquad + \beta_7 \, |110\rangle \, U(\boldsymbol{\theta}_{3,2}) U(\boldsymbol{\theta}_{2,2}) U(\boldsymbol{\theta}_{1,1}) \, |x\rangle + \beta_8 \, |111\rangle \, U(\boldsymbol{\theta}_{3,2}) U(\boldsymbol{\theta}_{2,2}) U(\boldsymbol{\theta}_{1,2}) \, |x\rangle \Big] \\
&= \frac{1}{\sqrt{8}} \sum_{j=1}^{8} \beta_j \, |j\rangle \, U(\boldsymbol{\Theta}_j) \, |x\rangle \, ,
\end{aligned} \tag{5.6}$$

where $U(\boldsymbol{\theta}_{3,1}, \boldsymbol{\theta}_{3,2}) = [C \otimes \mathbb{1}^{\otimes 2} \otimes U(\boldsymbol{\theta}_{2,1})] \, [X \otimes \mathbb{1} \otimes \mathbb{1}] \, [C \otimes \mathbb{1}^{\otimes 2} \otimes U(\boldsymbol{\theta}_{2,1})]$.

The final quantum state is a superposition of 8 different parametrized unitary transformations of the input, each entangled with a basis state of the *control*

register whose amplitudes depend on a set of parameters $\{\beta_i\}_{i=1,\dots,d}$. We can generalize the quantum state $|\Phi\rangle_{(d=3)}$ for a generic d-qubit *control* register:

$$\begin{aligned} |\Phi\rangle_d &= \prod_{i=1}^{d} U(\boldsymbol{\theta}_{i,1}, \boldsymbol{\theta}_{i,2}) \left(\bigotimes_{i=1}^{d} |c_i\rangle \otimes |x\rangle \right) \\ &= \frac{1}{\sqrt{2^d}} \sum_{j=1}^{2^d} \beta_j |j\rangle U(\boldsymbol{\Theta}_j) |x\rangle = \frac{1}{\sqrt{2^d}} \sum_{j=1}^{2^d} \beta_j |j\rangle |L(x; \boldsymbol{\Theta}_j)\rangle , \end{aligned} \tag{5.7}$$

where $U(\boldsymbol{\Theta}_j)$ is the product of d unitaries $U(\boldsymbol{\theta}_{i,k})$ for $i = 1, \dots, d$ and $k = 1, 2$.

To summarize, the underlying idea of this procedure is to initialize the *control* register according to a set of weights and assign each weight β_j to a parametrized unitary function $U(\boldsymbol{\Theta}_j)$. This approach is extremely flexible and allows learning all the parameters β_j and $\boldsymbol{\Theta}_j$ for specific use cases. Furthermore, it allows to generate a superposition of 2^d diverse unitary transformation while increasing linearly the depth correspondent quantum circuit.

Step 3: Activation. A further step consists of applying the Σ gate, representing the quantum version of the classical activation function to the *data* register. Notice that, having all the parametrized unitary operations in superposition allows propagating the application of Σ with a single execution, as follows:

$$\begin{aligned} |\Phi\rangle_\Sigma &= (\mathbb{1}^{\otimes d} \otimes \Sigma) |\Phi_d\rangle \rightarrow \frac{1}{\sqrt{2^d}} \sum_{j=1}^{2^d} \beta_j |j\rangle |\sigma[L(x; \theta_j)]\rangle) \\ &= \frac{1}{\sqrt{H}} \sum_{j=1}^{H} \beta_j |j\rangle |g(x; \boldsymbol{\Theta}_j)\rangle , \end{aligned} \tag{5.8}$$

where $H = 2^d$ and $\mathbb{1}^{\otimes d}$ is the identity matrix. In this way, the result of the algorithm corresponds to the output of the SLP (Eq. (5.1)) with 2^d hidden neurons that can be accessed by measuring the *data* register only.

Step 4: Measurement. Finally, the expectation measurement on the *data* register is performed:

$$\begin{aligned}\langle M\rangle &= \left\langle \Phi_\Sigma | \mathbb{1}^{\otimes d} \otimes M \middle| \Phi_\Sigma \right\rangle \\ &= \sum_{j=1}^{2^d} \beta_j' \left\langle g(x;\mathbf{\Theta}_j)|M|g(x;\mathbf{\Theta}_j)\right\rangle \\ &= \sum_{j=1}^{2^d} \beta_j' \left\langle M_j \right\rangle = \sum_{j=1}^{2^d} \beta_j' g_j = f(\mathbf{x};\boldsymbol{\beta},\ ),\end{aligned} \tag{5.9}$$

where M is a generic measurement operator (e.g., the Pauli σ_z), the function $g(x;\mathbf{\Theta}_j) = \sigma[L(x;\theta_j)]$ and $\beta_j' = |\beta_j|^2$ with $\sum_j |\beta_j|^2 = 1$. Although we do not measure the *control* register, the j-th transformation of the input is associated to a specific amplitude β_j of the *control* register. The parameters of the quantum circuit $\{\hat{\beta}_i\}_{i=1,\dots,d}$ and $\{\boldsymbol{\theta}_{i,1},\boldsymbol{\theta}_{i,2}\}_{i=1,\dots,d}$, that indirectly determine the parameters in Eq. (5.9), can be randomly initialized and hybrid quantum-classical optimization process can be exploited.

Thus, we extended the proposed approach of the qSLP [1] to an exponentially large number of neurons in the hidden layer. In fact, the entanglement of linear combinations to the basis states of the *control* register implies that the number of different linear combinations is equal to the number of basis states of the *control* register. This translates in a number of hidden neurons H which scales exponentially with the number of qubits of the *control* register, d. This is a consequence of each hidden neuron being represented by a single independent trajectory (Eq. (5.8)). The exponential scaling alongside the ability to freely learn the parameters, enables the construction of quantum neural network with an arbitrary large number of hidden neurons as the amount of available qubits increases. In other terms, we can build qSLP with an incredible descriptive power that may be really capable of being an universal approximator. The quantum circuit to implement the qSLP ($d = 3$) is depicted in Figure 5.2.

Step 5: Optimization. The post-processing is task-dependent and performed classically. In particular, given the measurement of the quantum circuit $f(\mathbf{x};\boldsymbol{\beta},\ )$,

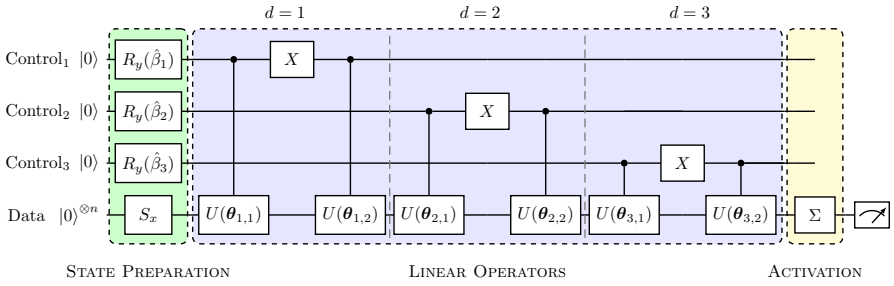


Figure 5.2: Quantum circuit for a qSLP with 8 ($d = 3$) hidden neurons.

a cost function is computed and then classical approaches to update the parameters of the quantum circuit $\{\boldsymbol{\beta},\ \}$ are employed. As loss function, in case of supervised problem, the *Sum of Squared Errors* (SSE) between the predictions and the true values y is usually adopted:

$$SSE = Loss(\beta,\ ; D) = \sum_{i=1}^{N} \left[y_i - f\left(\mathbf{x}_i; \boldsymbol{\beta},\ \right)\right]^2, \tag{5.10}$$

where N is the total number of observations in the training set.

To summarize, the variational algorithm described above allows reproducing a classical neural network with one hidden layer on a quantum computer. In particular, it includes a variational circuit adopted for encoding the data, performing the linear combinations of input neurons and applying the same activation function to their results with just one execution. A single iteration during the learning process is then completed to measure the output of the network, compute the loss function and update the parameters. The whole process is repeated iteratively until convergence, as for classical neural networks.

Discussion

The algorithm provided in the previous section allows building a generalized qSLP with an exponentially large number of neurons, increasing the depth of the quantum circuit linearly. Furthermore, the model supports amplitude encoding strategy, which translates into a exponential advantage in terms of space complexity (i.e., number of qubits) when encoding data into the *data* quantum register. This implies a potential polylogarithmic advantage in terms of the number of parame-

ters with respect to its classical counterpart [103].

There are two main differences between the classical and quantum SLP, deriving from the normalization constraint introduced when dealing with the amplitudes of quantum systems. In the qSLP, the input data $\mathbf{x}$ and the weight vector $\boldsymbol{\beta} = \{\beta_j\}_{j=1,\dots,2^d}$ are normalized to 1. This may seem a limitation since classical neural networks may take raw input and freely learn the weight parameters. However, rescaling the inputs and limiting the magnitudes of the weights are two common strategies adopted in the classical case to avoid overfitting. In particular, these procedures are known as *batch normalization* and *weight decay* [104]. Thus, quantum mechanics' normalization constraint allows automatically implementing of ad-hoc procedures developed in the context of classical neural networks without any additional computational effort.

From a computational point of view, given H hidden neurons and L training epochs, the training of a classical SLP scales (at least) linearly in H and L, since the output of each hidden neuron needs to be calculated explicitly to obtain the final output. Also, if H is too large (a necessary condition for an SLP to be a universal approximator [100]), the problem becomes NP-hard [105]. The proposed qSLP, instead, allows scaling linearly with respect to $log_2(H) = d$ thanks to the entanglement between the two quantum registers that generates an exponentially large number of quantum trajectories in superposition. However, the cost of state preparation (gate S_x) needs to be taken into consideration, as well as the cost implementing the quantum activation function Σ. Nonetheless, once the optimal set of parameters of the qSLP is obtained for a specific task, the whole quantum algorithm can be employed as subroutine in other quantum algorithms to reproduce the function for which it is trained.

5.1.2 Experiments

To test the performances of the qSLP, we implemented the circuit illustrated in Figure 5.2 using IBM Qiskit environment on two different real-world datasets. After training the qSLP on the simulator, we executed the pre-trained algorithm on two real quantum devices. In addition, QNNs [27] and QSVMs [33] are adopted

as benchmark to compare the proposed qSLP with state-of-the-art QML models[1].

Dataset Description

The simulation of a quantum system on a classical device is a challenging task even for systems of moderate size. For this reason, the experiments consider only datasets with a relatively small number of observations (100–200) that will be split in training (80%) and test (20%) set. Furthermore, the PCA is performed to reduce the number of features to 2. We consider five different binary classification problems from two real-world datasets (MNIST and Iris) usually adopted for benchmarking classical machine learning algorithms. The Iris dataset consists of 50 examples for three species of Iris flower (*Setosa, Virginica, Versicolor*) described by four different features. The MNIST dataset contains $60k$ images of ten possible handwritten digits, each represented by 28×28 pixels. Since these two datasets exhibit a multiclass target variable, but the current implementations of qSLP, QNN, and QSVM solve a binary classification problem, we consider five different datasets extracted from the original multiclasses problems (Table 5.1).

State Preparation for the qSLP

The most efficient encoding strategy adopted in QML is amplitude encoding that associates quantum amplitudes with real vectors at the cost of introducing a normalization constraint to the raw input. Formally, a normalized vector $\mathbf{x} \in \mathbb{R}^p$ can be described by the amplitudes of a quantum state $|x\rangle$ as:

$$|x\rangle = \sum_{k=1}^{p} x_k |k\rangle \longleftrightarrow \mathbf{x} = [x_1, \ldots, x_p]^T . \tag{5.11}$$

Two different approaches are used for amplitude encoding in the qSLP: *Padded state preparation* and *Single-qubit state preparation*.

Single-qubit state preparation. This approach encodes a two-dimensional real vector $\mathbf{x}$ into the amplitudes of a qubit by performing two parametric ro-

[1] All code to generate the data, figures and analyses is available at github.com/filorazi/qSLP-quantum-Single-Layer-Perceptron

	MNIST09	**MNIST38**	**SeVe**	**SeVi**	**VeVi**
Training set	160	160	80	80	80
Test set	40	40	20	20	20
Raw features	784	784	4	4	4
PCA features	2	2	2	2	2
% Variance	31%	20.3%	98%	98.4%	92.2%
Target classes	0 vs 9	3 vs 8	Setosa vs Versicolor	Setosa vs Virginica	Versicolor vs Virginica

Table 5.1: **Datasets description**. The columns represent five different binary classification datasets. Each row describes a characteristic of the data: the number of training points (Training set), the number of test points (Test set), the number of features in the original data (Raw features), the number of PCA features, the explained variance of the PCA (% Variance) and the target classes chosen as the binary target variable (Target classes).

tations $R_y(x_1), R_z(x_2)$, where the angles $\{x_1, x_2\}$ are the two components of the (normalized) feature vector $\mathbf{x} = (x_1, x_2)$ [106]. In this case, the parametrized quantum gate $U(\boldsymbol{\theta}_{i,k})$ of the qSLP is directly implemented using the gate U_3 (Eq. (5.2)).

Padded state preparation. A second strategy for amplitude encoding consists of mapping an input state $|x\rangle$ to the all-zero state $|0 \dots 0\rangle$. Once the circuit is obtained, all of the operations are inverted and applied in the reversed order. In this case, a two-dimensional input data $\mathbf{x}$ is first mapped into a four-dimensional vector adding constant values and then normalized. Thus, following the procedure described in [2], a set of angles are defined in such a way to apply a sequence of R_y and CNOT gates to generate a quantum state whose amplitudes are equivalent to the input (four-dimensional) feature vector. The quantum circuit for *padded state preparation* is depicted in Figure 5.3. When using this strategy, since the *data* register is made up of 2 qubits, the unitary operator $U(\boldsymbol{\theta}_{i,k})$ of the qSLP is implemented by two U_3 gates (one per qubit) and a CNOT gate.

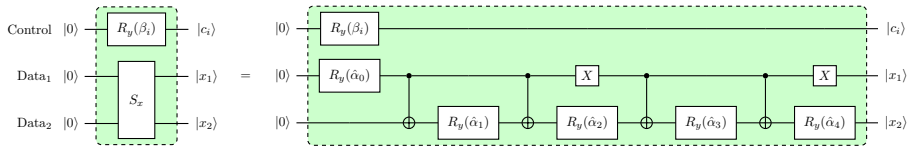


Figure 5.3: Quantum circuit for *padded state preparation* in the qSLP. The $\{\hat{\alpha}_i\}_{i=0,\dots,4}$ parameters are determined following the procedure described in [2].

5.1.3 Results

We consider nine different algorithms based on the three QML models. A QSVM [33] is trained using a two-layer quantum feature map (*ZZFeature-map*). Two types of QNNs are tested: the first one (QNNC v1) uses single-layer quantum feature map (*ZFeature-map*) as state preparation routine and *Real amplitudes* approach as classification ansatz; the second type (QNNC v2) differs from the first for the use of *ZZFeature-map* as state preparation. Regarding the qSLP, six different configurations are tested based on two possible implementations for amplitude encoding (Section 5.1.2) and three values for the parameter d. To the best of our knowledge, there is no quantum routine suitable for near-term quantum computation capable of approximating the behaviour of an activation function. For this reason, as Σ gate (Eq. (5.8)), we adopt the identity gate[2].

The training is performed on a QASM simulator, a backend that simulates the execution of a quantum algorithm in a fault-tolerant quantum device. The number of measurements for each run of the quantum circuits is fixed to 1024. The results for each model are reported in Table 5.2.

Assuming a perfect quantum device, the QSVM and the QNN outperform the qSLP considering the VeVi dataset. This means that for specific cases the use of a quantum feature map provides a practical advantage with respect to standard amplitude encoding. However, in case of MNIST09, MNIST38 and SeVe the qSLP outperforms the QNNs, thanks to its ability to aggregate multiple and diverse unitary functions. Thus, the flexible architecture of the ansatz of the qSLP allows achieving a better performance when comparing full quantum models[3]. Nonethe-

[2]Importantly, the code already allows the embedding of a gate Σ different from the identity gate, as quantum activation function.

[3]The QSVM uses a quantum circuit to translate the classical data into quantum states while the classification is performed classically.

		MNIST09		MNIST38		SeVe		SeVi		VeVi	
Model	**d**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
QSVM		.97	**.91**	.83	**.84**	1.0	**1.0**	1.0	**1.0**	.98	**.95**
QNN (v1)		.87	.86	.55	.61	.95	.90	.98	.95	.92	.90
QNN (v2)		.84	.84	.74	.70	1.0	.95	1.0	**1.0**	.85	.80
(padded) qSLP	1	.92	.89	.89	.81	1.0	**1.0**	1.0	**1.0**	.82	.70
	2	.93	**.91**	.87	.82	1.0	**1.0**	1.0	**1.0**	.82	.70
	3	.92	**.91**	.86	**.84**	1.0	**1.0**	1.0	**1.0**	.82	.70
(Single-qubit) qSLP	1	.83	.85	.86	**.84**	1.0	**1.0**	1.0	**1.0**	.80	.70
	2	.83	.85	.87	**.84**	1.0	**1.0**	1.0	**1.0**	.78	.65
	3	.83	.85	.87	.82	1.0	**1.0**	1.0	**1.0**	.80	.65

Table 5.2: Training and test results of three different models (qSLP, QNN, QSVM) on five different binary classification problems.

less, the qSLP and QSVM equally perform on four cases (MNIST09, MNIST38, SeVe, SeVi), although the implementation of the qSLP is not complete due to the lack of a proper activation function and a limited number of hidden neurons ($H = 2^3 = 8$).

Model	**d**	**MNIST09**	**MNIST38**	**SeVe**	**SeVi**	**ViVe**
QSVM	-	**.91**	**.84**	**1.0**	**1.0**	**.95**
QNN (v1)	-	.89	.64	.95	**1.0**	**.95**
QNN (v2)	-	.89	.68	**1.0**	**1.0**	.75
(padded) qSLP	1	.89	.81	**1.0**	.90	.70
	2	.81	.65	**1.0**	.60	.50
	3	.81	.65	**1.0**	.60	.50
(Single-qubit) qSLP	1	.82	.78	**1.0**	**1.0**	.50
	2	.82	.70	**1.0**	**1.0**	.55
	3	.84	.78	**1.0**	.95	.45

Table 5.3: Test performances using real devices (*ibmq_lima* and *ibmq_quito*).

Furthermore, the trained algorithms have been executed on real IBM quantum devices. Results are reported in Table 5.3. The deterioration in the performance of the *(Single-qubit) qSLP* is negligible. While the results of the *(Padded) qSLP*, which uses a higher number of qubits and deeper quantum circuits, deteriorate

as d increases. Instead, the QSVM seems not to be affected by the use of a real device since the quantum component only generates the new features, while the classification is performed classically. However, being each feature represented by a single qubit (as for QNN), the use of the quantum feature map on real-world datasets seems to be prohibitive even when considering quantum devices with hundreds of (noisy) qubits. Instead, the amplitude encoding strategy adopted in the qSLP represents an optimal strategy to efficiently encode a large dimensional input data in a small number of qubits. Thus, the *(Single-qubit) qSLP* is the best compromise when using a real device, since it seems to preserve its performance while adopting the amplitude encoding strategy.

Importantly, these results are an indication of how the tested QML models perform and do not represent an exhaustive evaluation of their performance.

5.1.4 Conclusion and Outlook

In section, we proposed a quantum algorithm to generate the quantum version of the Single Layer Perceptron (qSLP). The key idea is to use a single state preparation routine and apply different linear combinations in superposition, each entangled with a control register. This allows propagating a generic activation function to all the basis states with only one execution. As a result, a model trained through our algorithm is potentially able to approximate any desired function as long as enough hidden neurons and a non-linear activation function are available. Furthermore, we provided a practical implementation of our variational algorithm that reproduces a quantum SLP for classification with different possible hidden neurons and an identity function as activation.

In addition, we performed a comparative analysis between our algorithm and two quantum baselines on real-world data and demonstrated that the qSLP outperforms other full quantum approaches (QNN) and matches with the QSVM.

The main challenges to tackle in the near future is the design of a routine that reproduces a non-linear activation function, as well as the adoption of the qSLP for regression tasks. Yet, recently it has been shown that quantum feature maps alongside functions aggregation are able to achieve universal approximation [107]. Thus, another promising future work is the study of the qSLP on top of the

quantum feature map, to obtain a universal approximator, without implementing a non-linear quantum activation function.

In conclusion, we are still far from proving that machine learning can benefit from quantum computation in practice. However, thanks to the flexibility of variational algorithms, the hybrid quantum-classical approach may be the ideal setting to make universal approximation possible in quantum computers.

5.2 Variational QSpline

5.2.1 Methods

In this section, the methodological contributions are presented. We describe two different approaches for quantum splines: the *VQSplines* adopts the piece-wise formulation of QSplines but replaces the HHL with the Variational Quantum Linear Solver (VQLS) [108]. As a consequence, we obtain a quantum algorithm suitable for NISQ devices capable of estimating any non-linear function. Second, we propose the *GQSplines* model, a novel formulation of QSplines that allows us to obtain an end-to-end quantum algorithm for non-linear approximation suitable for existing quantum neural networks.

Preliminaries

Quantum Splines

Spline functions are smoothing methods for modeling the relationships between variables, typically adopted either as a visual aid in data exploration or for estimation purposes [99]. The underlying idea is to use linear models in which the input features are augmented with the *basis expansions*. Technically, splines are constructed by dividing the sample data into sub-intervals delimited by breakpoints, also referred to as knots. A fixed degree polynomial is then fitted in each of the segments, thus resulting in piecewise polynomial regression.

While the formulation in terms of truncated basis functions is conceptually simple, its numerical and computational properties are not very attractive. For this reason, in practice, the *B-splines* parametrization [109] is adopted. This generates a block design matrix where the *sparsity* is constant and depends on the

degree of the polynomial fitted in each local interval. Given a sequence of knots $\xi_1, \xi_2, \cdots, \xi_T$, we fit a line in each interval $[\xi_k, \xi_{k+1}]_{k=1,\cdots,T-1}$ without derivability constraints.

$$\tilde{\boldsymbol{y}} = \boldsymbol{S}\boldsymbol{\beta} \rightarrow \begin{pmatrix} \tilde{y}_1 \\ \tilde{y}_2 \\ \cdots \\ \tilde{y}_K \end{pmatrix} = \begin{pmatrix} S_1 & 0 & \cdots & 0 \\ 0 & S_2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & S_K \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \cdots \\ \beta_K \end{pmatrix}, \tag{5.12}$$

where $\tilde{y_k}$ contains the function evaluations in ξ_k and ξ_{k+1}, β_ks are the spline coefficients and $\boldsymbol{S}_{(2K)\times(2K)}$ is a block diagonal matrix with each block S_k that represents the basis expansions in the k-th interval. Therefore, solving the linear system in Eq. (5.12) allows computing the splines coefficients, which serve to approximate non-linear functions encoded in the vector $\tilde{\boldsymbol{y}}$.

The idea of the Quantum Splines (*QSplines*)[34] is to adopt the B-spline formulation in the context of quantum computation to approximate non-linear functions. In particular, the computation of the QSplines is performed in three steps. First, the HHL computes the spline coefficients for the k-th interval encoded into the quantum state $|\beta_k\rangle$. Second, $|\beta_k\rangle$ interacts with the quantum state $|x_k\rangle$ encoding the input in the k-th interval via quantum interference through the swap-test [58]. This allows generating the state $|f_k\rangle$ which encodes the estimate of the non-linear function evaluated in x_k; Third, $|f_k\rangle$ is measured and post-processed to obtain y_k.

Although QSplines allow overcoming the limitation of unitary operation on quantum states, their applicability is very limited since the use of the HHL as a subroutine requires a massive number of error-corrected qubits to be executed. Furthermore, in the current formulation, the final quantum state is obtained using the swap test, whose results are not directly encoded in the amplitude and need a post-processing step to be calculated. All these factors forbid the adoption of QSplines in current models of quantum neural networks, which run on a limited set of noisy qubits.

Variational Quantum Linear Solver

An alternative quantum algorithm to solve a linear system of equations is the

Variational Quantum Linear Solver (*VQLS*) [108]. Specifically, the VQLS solves a linear system of equations using a variational hybrid quantum-classical approach which is suitable for near-term quantum devices. Given a matrix A and a state vector $|b\rangle$, the VQLS prepares the state $|x\rangle$ such that:

$$A|0\rangle = A|x\rangle \propto |b\rangle . \tag{5.13}$$

The matrix A is defined as a linear combination of unitary matrices A_l:

$$A = \sum_{l=0}^{L} A_l c_l \tag{5.14}$$

where c_l are complex numbers. With this input, the VQLS generates the state $|x\rangle$ employing an ansatz for the gate sequence $V(\theta)$ such that $|x(\theta)\rangle = V(\theta)|0\rangle$. The parameters θ are input to a quantum computer, which prepares $|\theta\rangle$ and runs an efficient quantum circuit that estimates a cost function $C(\theta)$. The value of $C(\theta)$ from the quantum computer is returned to the classical computer which then adjusts θ (via a classical optimization algorithm) in an attempt to reduce the cost. This process is iterated many times until one reaches a termination condition of form $C(\theta) < \gamma$, at which point we say that $\theta = \theta_{\text{opt}}$.

Once the cost function is minimized, the Ansatz $V(\theta_{opt})$ with the optimal parameters θ_{opt} prepares the normalized state $|x(\theta_{\text{opt}})\rangle$ which approximates $|x\rangle$:

$$V(\theta_{\text{opt}})|0\rangle = |x(\theta_{\text{opt}})\rangle . \tag{5.15}$$

Notice that this description of the problem is coherent with the original VQLS. However, in the case of QSplines, the linear system of equation in Eq.(5.12) represents the target solution (the B-spline coefficients) as $|\beta\rangle$, the A_l circuit encodes the matrix S, and $|y\rangle$ replaces $|b\rangle$.

VQSplines: Variational Algorithm for QSplines

The idea behind the *VQSplines* is to implement the QSplines in the context of hybrid quantum-classical computation and provide an efficient method to estimate non-linear functions using near-term quantum computers. In particular, the VQS-

plines replace the HHL with the VQLS and adopt the quantum inner product [110] to generate a quantum state encoding the value of the non-linear target function, avoiding the use of the swap-test.

In practice, the computation of the VQSplines is performed in two steps. First, the VQLS estimates the spline coefficients for the k-th interval:

$$S_k \left|\beta'_k\right\rangle = \left|y'_k\right\rangle \xrightarrow{VQLS} \left|\beta'_k\right\rangle = \beta'_{k,0} \left|0\right\rangle + \beta'_{k,1} \left|1\right\rangle \approx S_k^{-1} \left|y'_k\right\rangle \tag{5.16}$$

In order to obtain the quantum state $\left|\beta'_k\right\rangle$, each 2×2 linear system requires the training of a parametrized quantum circuit which needs three different quantum gates: a unitary $V(\theta_k)$ implemented with a Pauli rotation gate R_y that allows generating a quantum state whose amplitudes encode the B-spline coefficients, once the optimal set of parameters $\theta_{k,\text{opt}}$ is obtained. Notice that, as for the QSplines, the entire problem is decomposed into K 2×2 linear systems, and each set of coefficients can be encoded using a single qubit; the second set of quantum gates are the unitaries A_l, which encode the information related to the S_k matrix. In particular, the matrix S_k is decomposed by means of a linear combination of known unitary matrices as follows:

$$S_k = \begin{bmatrix} 1 & a \\ 1 & b \end{bmatrix} = \sum_{l=0}^{3} A_l c_l = I c_0 + X c_1 + Z c_2 + R_y(3\pi) c_3 \tag{5.17}$$

where the coefficients of the linear combination are initialized as follows:

$$c_0 = (b+1)/2; \qquad c_1 = (a+1)/2; \qquad c_2 = (1-b)/2; \qquad c_3 = (a-1)/2;$$

Third, the unitary U encodes the classical vector y_k into the amplitude of $\left|y'_k\right\rangle$ [2]. The QSplines model for the k-th interval is depicted in Figure 5.4.

As a result of the optimization process, we obtain a variational circuit that creates a quantum state encoding the solution of the linear system:

$$V(\theta_{k,opt}) \left|0\right\rangle = \left|\beta'_k\right\rangle = \beta'_{k,0} \left|0\right\rangle + \beta'_{k,1} \left|1\right\rangle \tag{5.18}$$

The second step of the VQSplines computes the inner product between the

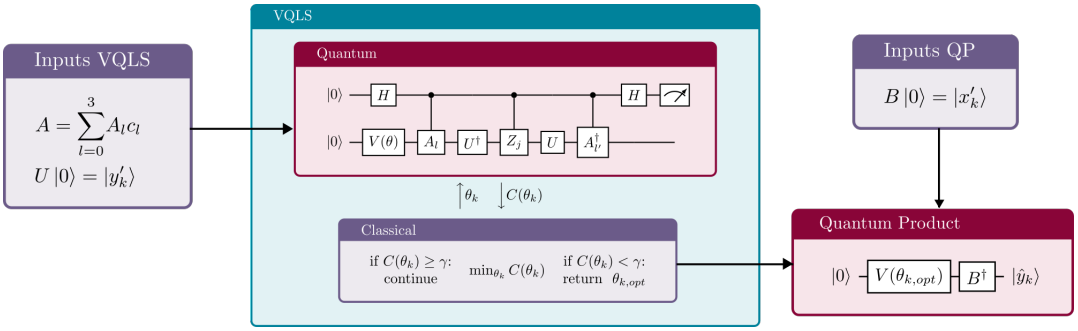


Figure 5.4: The VQSplines architecture. The quantum part of the VQLS is optimized classically and then $\theta_{k,opt}$ is used in the quantum inner product to compute $|\hat{y}_k\rangle$

basis expansion of the input $|x'_k\rangle$ and the B-splines coefficients $|\beta'_k\rangle$. To this end, the quantum inner product [110] generates a quantum state whose amplitudes encode the value of the non-linear function:

$$B^\dagger V(\theta_{k,opt})\,|0\rangle_n = a_o\,|0\rangle_n + \sum_{i=1}^{N-1} a_i\,|i\rangle_n \tag{5.19}$$

$$a_0 = \langle 0|\,B^\dagger V(\theta_{k,opt})\,|0\rangle = \langle x'_k|\,BB^\dagger\,|\beta'_k\rangle = \langle x'_k|\beta'_k\rangle = \hat{y}_k \tag{5.20}$$

where B is the amplitude encoding quantum routine [110] of the basis expansion of the input x_k. The quantum circuit of the non-linear target function for the k-th interval is built as shown in Figure 5.4 (*Quantum Inner product*).
These steps are repeated for each sub-interval and the solutions of the quantum system in Eq. (5.12) are obtained. Therefore, as for QSplines, once $|\beta'\rangle$ is known, the VQSplines require $\mathcal{O}(K)$ repetitions of the quantum inner product circuit to calculate a spline function with K knots.

Importantly, the use of the quantum inner product allows one to directly obtain the value of the target function as amplitudes of a quantum state and does not require any post-processing as for QSplines.

GQSplines: Generalized Quantum Splines

The original formulation of the QSplines and VQSplines relies on a block diagonal B-Spline matrix which allows decomposing the entire problem into K sub-problems

that can be solved independently. However, this approach does not allow the estimation of a non-linear function in a full quantum manner, which is a requirement for embedding a quantum activation function into a quantum neural network. For this reason, we propose the *Generalized Quantum Splines* (GQSplines), a novel approach that relies on a more general formulation of the B-splines and allows estimating the value of a non-linear function with a single quantum circuit without problem decomposition.

Given the recursive definition of B-Spline [109] and the related basis expansion with knots list $\xi = [\xi_1, .., \xi_i, \xi_{i+1}, .., \xi_T]$, a non linear function f can be estimated using the observed values $Y = \{y_1, .., y_K\}$ given the inputs $X = \{x_1, .., x_K\}$. In this case, the linear system of equations describing the relation between the estimates of the activation function Y, the matrix S and the spline coefficients β is the following:

$$Y_{K\times 1} = S_{K\times K}\beta_{K\times 1} \implies \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} B_{1,d}(x_1) & \dots & B_{l,d}(x_1) \\ B_{1,d}(x_2) & \dots & B_{l,d}(x_2) \\ \vdots & \ddots & \vdots \\ B_{1,d}(x_K) & \dots & B_{l,d}(x_K) \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_K \end{bmatrix}, \tag{5.21}$$

where d is the degree of the B-spline, T is number of knots and $l = T - d - 1$.

Nonetheless, to adopt the VQLS (or the HHL) for solving the linear system of equations, the basis expansion matrix S has to be hermitian and, therefore, square and non-singular. The GQSplines formulation imposes the matrix S to be Hermitian, i.e., $K = l = T - 2$, and adopts a quantum linear solver to find the set of optimal parameters $|\beta\rangle$[4]. Assuming to fit a linear function in each interval (i.e., $d = 1$), the linear system of the GQSplines is:

[4]It is also possible to define the Hermitian matrix H from S as $H = \begin{pmatrix} 0 & S \\ S^\dagger & 0 \end{pmatrix}$.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{k-1} \\ y_K \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & \dots & 0 & 0 \\ 0 & 1-x_2 & x_2 & \dots & \dots & 0 \\ \dots & 0 & 1-x_3 & x_3 & \dots & \dots \\ \dots & \dots & 0 & 1-x_4 & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & 1-x_{K-1} & x_{K-1} \\ 0 & 0 & \dots & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \vdots \\ \beta_{K-1} \\ \beta_K \end{bmatrix} . \tag{5.22}$$

which leads to the following quantum linear system of equations:

$$S\,|\beta\rangle = |Y\rangle \tag{5.23}$$

where $|\beta\rangle$ and $|Y\rangle$ are two quantum states that encode in their amplitudes the vectors Y and β. Importantly, the normalization constraint of quantum states $|\beta\rangle$ and $|Y\rangle$ imposes rescaling the values of the target variable such that $\langle Y|Y\rangle = \langle\beta|\beta\rangle = 1$.

With a particular focus on the VQLS as a quantum linear solver, the size of the linear system in Eq. (5.23) is K which gives us the number of qubits required to find the optimal coefficients. Precisely, since the VQLS requires the vectors to be encoded as quantum state, the number of qubits scale logarithmically in the number of knots K, which is directly related to the quality of the fitting of the curve (the higher is K, the better the fitting is). This exponential scaling with respect to the number of inputs is a significant improvement when compared to other quantum approaches for quantum activation functions [111] whose number of qubits scales linearly with the size of the inputs.

Given the linear system in Eq. (5.21), the GQSplines proceed in two steps. Firstly, the coefficients $|\beta'\rangle$ are generated using the VQLS (or the HHL):

$$|\beta'\rangle = VQLS(S, Y) = \sum_{i=1}^{K} \beta_i\,|i\rangle\,. \tag{5.24}$$

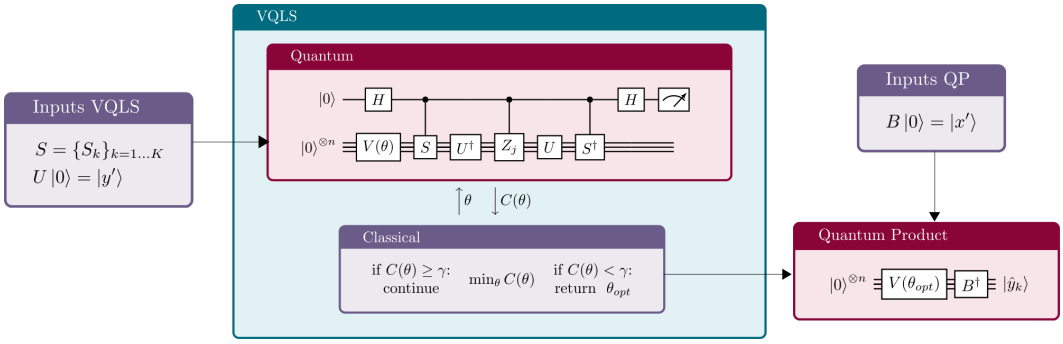


Figure 5.5: GQSplines architecture. The quantum part of the VQLS is optimized classically to obtain θ_{opt} that is then used in the quantum inner product to compute $|\hat{y}_k\rangle$. All $y'_k \in y$ can be approximated with the results of a single optimization of the VQLS.

Thus, quantum state $|\beta'\rangle$ interacts via interference with $|x'\rangle$ representing the basis expansion of x by means of the quantum inner product, as for VQSplines.

In the case of GQSplines, the VQLS circuit requires $n+1$ qubits, where $K = 2^n$ is the number of knots. Therefore, the Hadamard test [112] employs n qubits to implement the operators ($V(\theta)$,A and U) and one for the ancilla qubit. Furthermore, if $d = 1$, we end up with a diagonal block matrix which allows decomposing the matrix S (Eq. (5.14)) in terms of quantum gates in such a way to define the quantum circuit of the VQLS efficiently. Specifically, we can act independently on each interval to define the k-th matrix decomposition as follows:

$$S_k = \begin{bmatrix} 1-a & a \\ 0 & 1-b \end{bmatrix} = \sum_{l=0}^{3} A_l c_{k,l} = I c_{k,0} + X c_{k,1} + Z c_{k,2} + R_y(3\pi) c_{k,3} \qquad (5.25)$$

where the coefficients of the linear combination are computed as:

$$c_0 = 1 - a/2 - b/2; \qquad c_1 = a/2; \qquad c_2 = (b-a)/2; \qquad c_3 = a/2$$

This method allows the S matrix to be efficiently decomposed and obtain the linear combination of quantum gates required for the VQLS ([108]). The operator U is realized by amplitude encoding state preparation [110]. Still, the normalization of Y is required. With this new formulation, we are able to solve a singular

linear system and encode all the spline coefficients β in a unique quantum state by implementing the Ansatz only once. Subsequently, this state is used to compute the inner product with the k-th row of the matrix S (encoded through the routines B_i) and return the $\hat{y}_k$ estimates describing $\hat{Y}$. The workflow of GQSplines is depicted in Figure 5.5.

5.2.2 Evaluation

In this section, we implement and evaluate the proposed VQSplines and GQSplines for approximating three typical non-linear activation functions usually adopted in classical neural networks (*sigmoid*, *elu*, and *relu*) and the *sine* function[5]. The algorithms are implemented through the use of Pennylane [6](0.27.0 version).

Experimental Settings

Data Generation We consider *sigmoid*, *elu*, and *relu* activation functions and the *sine* function to showcase the non-linearity of our models. Eq. (5.26) shows the formulation of the functions tested for VQSplines, which is coherent with the experimental setting of original QSplines [34]:

$$\begin{aligned} \text{ReLu}(x) &= max(0, x) \qquad \text{Elu}(x) = \begin{cases} x \text{ if } x > 0 \\ 0.3(e^x - 1) \text{ otherwise} \end{cases} \\ \text{Sigmoid}(x) &= \frac{1}{1 + e^{-4x}} \qquad f_{\text{sin}}(x) = sin(\pi x) \end{aligned} \tag{5.26}$$

In the case of GQSplines, the input features and the value of the non-linear activation function are encoded into the amplitude of a quantum state. In this respect, the same functions are normalized in the interval $[0, 1]$. Thus, for the VQSplines the approximation of *Elu*, *ReLu*, and *Sigmoid* is carried out according to the experiments of the original QSplines paper, while *sin* allows for comparison of other approaches [111].

Metrics As discussed in the previous sections, different models are tested using different renormalizations. In order to perform a fair comparison between all methods, we

[5]Though the primary objective of QSplines is to embed non-linearity into quantum neural networks, they can serve to approximate other types of non-linearity.

[6]https://pennylane.ai/

consider the Normalized Root Mean Squared Errors (NRMSE) are defined as:

$$\textbf{NRMSE} = \frac{\sqrt{N^{-1}\sum_{i=0}^{N}(\hat{y}_i - y_i)^2}}{y_{max} - y_{min}}. \tag{5.27}$$

The NMRSE allows for a robust comparison that investigates the quality of the fitting independently of the scale of the target variable and the number of points used to approximate it.

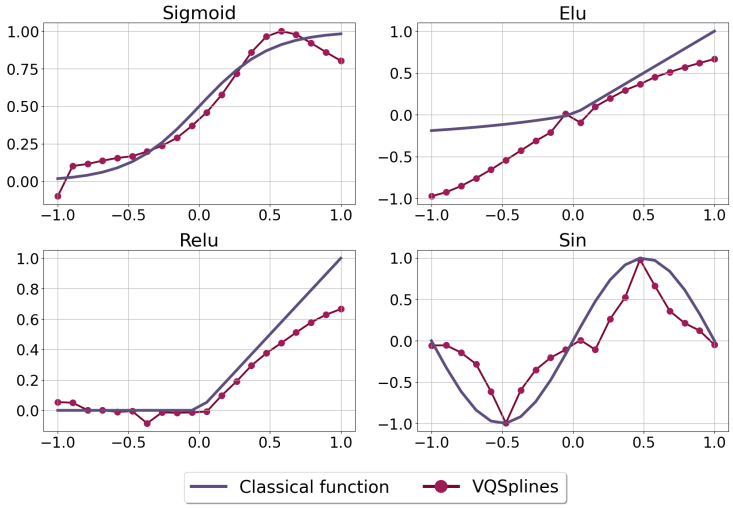


Figure 5.6: VQSpline experimental results. The model is able to approximate the function and obtain non-linearity. Each point j is computed as a product between $|x_j\rangle$ and the spline coefficient $|\beta'_j\rangle$

Results

We test the VQSplines by training the VQLS and computing the quantum inner product to produce the final estimates $\hat{Y}$ for curve described in Section 5.2.2. The results are shown in Figure 5.6.

We can see that in all the cases, the VQSplines can capture the non-linearity of the curves as expected. For *relu* and *sigmoid*, we have a good approximation, while the same cannot be observed for the *elu* and *sine*. In particular, the approximation of the curves deteriorates at the boundaries of the activation functions with input close to -1 or 1.

Considering the results of the GQSplines, the experiments are performed on the four functions normalized into the interval $[0, 1]$. For each curve, we show the fitting results in Figure 5.7, using 16 knots and 4 qubits. In this case, the approximation quality seems

to be sensitively better with respect to the VQSplines. However, while increasing the number of knots allows a better fitting, it implies using a larger quantum state which flatters the curve since the vectors $|Y\rangle$ and $|\beta\rangle$ are normalized to 1.

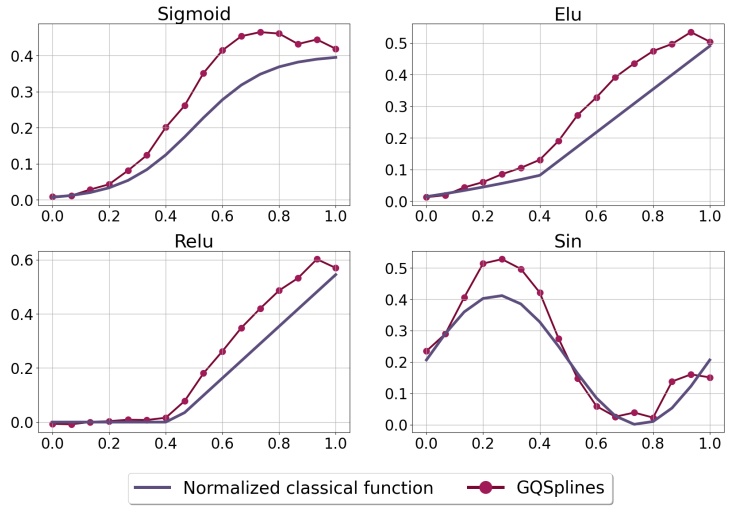


Figure 5.7: GQSplines experimental results. The model is able to approximate and emulate the trend of all 4 normalized functions. Each point is computed as the product of the j-th row of S and the B-spline coefficients $|\beta'\rangle$.

We can observe that there is a slight tendency of the GQSplines to overestimate the target function. Nevertheless, the method can capture the non-linearity of the curves while approximating their, especially in the case of the activation function. These results are achieved using only 4 qubits which is a significant improvement with respect to the experiments of other proposed approaches in the literature.

In order to make a fair comparison between QSplines, VQSplines and GQSplines, we calculate the NRMSE for the four curves and report the results in Table 5.4. We can observe that both proposed methods outperform the original QSplines. Although VQSplines perform better in terms of NRMSE than GQSplines for the *relu* and *sigmoid*, the differences between the two methods are minimal. The same happens when looking at the approximation of *elu* and *sigmoid*, where the GQSplines perform better.

5.2.3 Discussion

GQSplines and VQSplines allow estimating the value of non-linear functions by means of parameterized quantum circuits. Both methods outperform the original proposal of

Model	Knots	Elu	Relu	Sigmoid	Sin
Qsplines	20	0.4874	0.5240	0.1589	—
GQSpline	16	**0.0126**	0.0111	0.0156	**0.0099**
VQSpline	20	0.1278	**0.0069**	**0.0067**	0.0677

Table 5.4: NRMSE on each function for the proposed models and the baseline. The best approximation for each function is highlighted, and we can see that our model provides a considerable increase in performance with respect to QSplines.

QSplines in terms of fitting and require a significantly less number of qubits with respect to other existing approaches for quantum activation functions [111, 36].

In the case of VQSplines, the quality of the estimates in each interval depends to the specific condition number of linear systems, that for activation functions increases as one moves towards the extremes of the function. In fact, if the matrix S in Eq. (5.23) is ill-conditioned (i.e., it has a high condition number) the quality of the solution provided by a quantum solver is negatively affected. In particular, the larger the condition number is, the higher the probability of obtaining numerical errors in solving the linear system [108] since ill-conditioned systems converge slowly [113]. This implies that subsystems where the condition number is high lead to worse estimated coefficients.

The GQSplines algorithm uses a problem formulation that allows obtaining an end-to-end quantum algorithm representing the spline coefficients with a single linear system of equations. In this case, results are more stable, the shape of the curve is well-approximated, and the non-linearity is well-captured. However, the limitation is that the curves must be normalized, and the normalization depends on the number of Knots which defines the size of the linear system. To tackle this problem, one can consider adopting amplitude amplification [114] to stretch the shape of the curve and increase the amplitudes to correct the normalization effects. Nonetheless, increasing the number of Knots (which scales logarithmically with the number of qubits of the VQLS) allows to build smaller intervals and potentially improves the fitting quality. This is a significant improvement with respect to existing quantum approaches [111] where the input scale linearly with the number of qubits.

Furthermore, to use of GQSplines as a method for quantum activation functions requires the ability to calculate the target variable $\hat{y}_j$ from a generic input x_j. In this regard, we have to devise a mapping circuit M that creates a vector representing the basis expansion of x_j. Then, in order to obtain $\hat{y}_j$ we have to apply the dot-product

Method	End-to-end	Linear Problem Solver	Quantum Inner Product	Post
QSplines	×	HHL	Swap Test	×
VQSplines	×	VQLS	Inner Product	✓
GQSplines	✓	VQLS	Inner Product	✓

Table 5.5: Comparison between QSplines [34], VQSplines, and GQSplines. The table describes whether the approach allows obtaining an end-to-end quantum routine for non-linear approximation (*end-to-end*), the quantum linear solver in use (VQLS [108] or HHL [35]), and whether the results are directly encoded into a quantum state or not depending on the use either the swap-test [60] (not accessible) or the quantum inner product [110] (accessible). This dichotomy is described by the column *Post*.

between the basis expansion of x_j and the β coefficients, both encoded as a quantum state. As before, the output $\hat{y}_j$ will be encoded by the quantum state calculated by the following circuit:

$$\left|x'_j\right\rangle \otimes |0\rangle^{\otimes(n-1)} \quad \boxed{M(T,[\xi_i,\xi_{i+1}])} \quad \boxed{V^\dagger(\theta_{opt})} \quad \hat{y}_j\,|0\rangle + \Sigma_{i=1}^{k}\alpha_i\,|i\rangle \tag{5.28}$$

Note that the only difference between this approach and the one described for the GQSplines is that here the Ansatz ($V^\dagger$) is transposed and applied in reverse order with respect to the encoding of the basis expansion of x_j. This procedure allows the proposed GQSplines method to produce a non-linear function f for a given input x_j which approximates the target variable y. This approach can be adopted in existing quantum neural networks such as the quantum Single Layer Perceptron [1, 115] or the more general MAQA Framework [116], which require an end-to-end quantum routine storing the input-output of the non-linear activation function into the amplitudes of a quantum state.

Table 5.5 summarizes the properties of QSplines [34] and the two proposed methods. The VQSplines overcome the issue of performing the post-processing step and replace the HHL with the VQLS. However, they still require an ad-hoc problem decomposition with a diagonal block matrix. The GQSplines overcome this issue by defining a single linear system of equations representing the splines function.

5.2.4 Conclusion

Quantum Machine Learning (QML) has recently attracted ever-increasing attention and promises to impact various applications by leveraging quantum computational power and novel algorithmic models, such as Variational Algorithms. However, the field is still in its infancy, and its practical benefits need further investigation. One of the major issues in building a complete quantum neural network is the limitation of unitary, and therefore linear, operations on quantum states. In this work, we move toward a non-linear approximation of quantum activation functions using parametrized quantum circuits. In particular, we showed that it is possible to circumvent the constraint of unitarity in quantum computation by presenting an efficient version of the QSplines, whose implementation falls within the context of fault-tolerant quantum computation. The proposed methods do not require a fault-tolerant quantum subroutine (such as HHL) and allow the approximation of non-linear functions using near-term quantum technology.

The contribution of this paper is twofold. The first part proposes the Variation Quantum Splines (VQSplines), an implementation of the fault-tolerant QSplines [34] in the context of hybrid quantum-classical computation. Additionally, Generalized QSplines (GQSplines) are formulated and discussed. The benefit of this new formulation lies in the ability to be generalizable with respect to the structure of the spline matrix. The GQSplines adopt a new basis expansion matrix formulation, avoid the problem decomposition, and allow for tackling the problem of the matrix inversion in an end-to-end manner, with one single linear system and the number of qubits that scales logarithmically with the number of knots. Furthermore, the GQSplines are more efficient with respect to the number of qubits required with respect to existing quantum approaches for quantum activation functions. Experiments showed that both methods outperform the QSplines and can efficiently capture the non-linearity of typical activation functions adopted in classical neural networks.

Future work will be dedicated to embedding the GQSplines as a subroutine in existing quantum neural networks to leverage the properties of quantum computing in typical machine learning tasks where the non-linearity is crucial.

5.3 Dimensionality reduction for SWAP Test

5.3.1 Preface

In quantum computation, the SWAP test is an ubiquitous routine that can be found in various circumstances and that is part of multiple more complex algorithms, including applications like cross-platform verification [61] and error mitigation [59].
In our case we focus our attention on two problems: computing higher-order functions of density matrices $\mathrm{tr}(\psi^n O)$ where ψ is a density matrix and O is an observable through the use of the multipartite generalization of the SWAP test [58], and the more practical comparison of two quantum state using the SWAP test after they pass through a communication channel.
The SWAP test can be seen as a function that, given two (or more) states in input, computes the inner product between the two

$$\mathrm{SWAP}(\phi, \psi) = |\langle\phi|\psi\rangle|^2 \tag{5.29}$$

This is done through the circuit presented in figure 5.8.
This routine is used mainly to compute the overlap between two states and can be expanded to an arbitrary number of quantum states. The main drawback of the SWAP

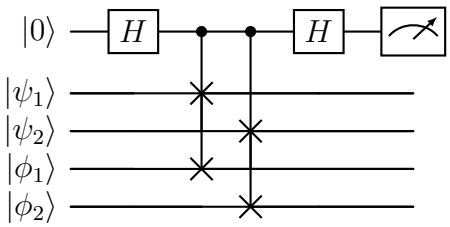


Figure 5.8: A Simple SWAP test circuit between two states ψ, ϕ stored in two qubit each.

test can be found in the high number of swap gates that it requires. This is particularly expensive because current quantum hardware has very limited connectivity between qubits, meaning that to perform a controlled swap operation, we have to move around each qubit multiple times. The cost in terms of swap gates highly increases when dealing with an arbitrary number of states since each qubit must be compared with its counterpart in every other state.

For these reasons, our research focuses on reducing the dimensionality of quantum states to simplify the swap test and to obtain a more reliable result after the state passes through a communication channel. The main idea is to utilize a quantum autoencoder to reduce the dimensions needed to reproduce the state, working under the hypothesis that when training a model to reduce the dimensions of a family of states, the reduced states have the same overlap as the original ones.
We explore the practicality of training a quantum autoencoder and how much its efficiency depends on the architecture or the loss function.

5.3.2 Methodology

We planned the experiments in steps, where at each step we tried to ascertain what was better for the model while moving from a toy example to a more realistic dataset.
First of all, we looked at the quantum autoencoders:
Quantum autoencoders are variational quantum machine learning architectures composed of two stages. The first stage (encoding) takes as input a register of n qubits and divides it into two groups: a compressed register of m qubits, which contains the compressed input state, and a trash space of t qubits, which ideally ends up in the $|0\rangle^{\otimes t}$ state if the model's training converges. The second stage (decoding) consists of the adjoint of the encoding operation, applied to the compressed state and t ancilla qubits initialized in the $|0\rangle^{\otimes t}$ state. Due to the unitary and reversible nature of quantum computation, if no information is lost during encoding, the output of the decoding process will be equal to the original input state.
Quantum autoencoders are classically inspired architectures that differ from their classical counterparts thanks to quantum properties such as reversibility, as discussed above. For this reason, quantum autoencoders are trained differently compared to classical autoencoders. Classically, autoencoders are self-supervised architectures trained using a reconstruction loss that compares the input and output, optimizing both the encoder and decoder simultaneously. In contrast, quantum autoencoders typically contain trainable parameters only in the encoder and are trained using a trash space loss, which focuses on driving the trash space toward a known basis state (typically $|0\rangle^{\otimes t}$).

Step 1: architecture

For the first part of the research We focused our attention on the architecture of the autoencoder. We didn't search for a mathematical proof of what architecture is best

since we observed that the answer seems to change based on the family of states we want to compress. Instead, we tried to come up with some general guidelines for the choice of architecture.
We decided to experiment on a trivial family of states called isotropic states [117]:

$$\rho_a = \frac{1-\alpha}{d^2}\mathrm{Id} + \alpha \left|\Phi^+\right\rangle \left\langle\Phi^+\right| \tag{5.30}$$

These states are governed by the parameter α and depend by the dimensions of the Hilbert space d.
Using these states as input for our experiments, we observed some interesting behaviours of the model: a modular architecture (similar to QNNs) seems to work better than a monolithic one (like the structure of QCNN). At the same time, utilizing results from relevant literature [118], we tried to understand which properties of the modular circuit best suit the autoencoder. Much like we anticipated, we observed that expressivity is an important parameter when deciding the architecture, but needs to be controlled to avoid barren plateaus [119]. This observation is nothing if not expected and pretty much a fundamental of variational quantum machine learning, but we were able to add something more connected with the quantum autoencoders, such as that the entangling capability of the circuit is not as important as it would be in other architectures. We hypothesize that this stems from the fact that the main objective of a quantum autoencoder (QAE) is to disentangle the states, meaning that the entangling capability of the model should be just enough to reverse the degree of entanglement of the family of states we utilize as input. Anything more might instead lead to loss of information. Unfortunately, we are still working on a mathematical proof for these observations that, to this day, are only conjectures derived from experimental observations.

Step 2: Loss function

Once we settled for a specific architecture, we increased the complexity of the experiments by changing to a slightly more complex input state family, the Transverse Field Ising Hamiltonian:

$$H = -J \sum_{\langle i,j \rangle} \sigma_i^z \sigma_j^z - h \sum_i \sigma_i^x \tag{5.31}$$

With $J = -1$ and h parameter. The dataset version of this Hamiltonian can be found in the Pennylane Library [120]. From this dataset, we took two instances with a chain lattice and closed periodicity of size 1×4 and 1×8.

This input is a well-known spin model that is used as a basic example of quantum phase transitions, spin systems, and ferromagnetism.
Having decided on the input, we moved on with the next experiment and focused on the loss function. Until this point, the quantum autoencoder utilized the fidelity measure as a cost function, but this function has some limitations that could limit the performance of the training phase. In the context of quantum information theory, the fidelity quantifies the similarities between two density matrices by expressing the probability that one state will pass a test to identify as the other. Notably, this by itself is not a metric on the space of density matrices, but it gives us the basis to develop a metric of our own.

The fidelity measure for two density matrices is defined as:

$$F(\rho, \sigma) = \left(\operatorname{tr} \sqrt{\sqrt{\rho}\sigma\sqrt{\rho}} \right)^2 \tag{5.32}$$

This can be simplified if one of the two density matrices is a pure state $\rho = |\psi_\rho\rangle \langle\psi_\rho|$:

$$\begin{aligned} F(\rho, \sigma) &= \left(\operatorname{tr} \sqrt{|\psi_\rho\rangle \langle\psi_\rho| \sigma |\psi_\rho\rangle \langle\psi_\rho|} \right)^2 \\ &= \langle\psi_\rho| \sigma |\psi_\rho\rangle \left(\operatorname{tr} \sqrt{|\psi_\rho\rangle \langle\psi_\rho|} \right)^2 \\ &= \langle\psi_\rho| \sigma |\psi_\rho\rangle \end{aligned} \tag{5.33}$$

and it becomes even more streamlined if both of them are pure $\sigma = |\psi_\sigma\rangle \langle\psi_\sigma|$:

$$F(\rho, \sigma) = \langle\psi_\rho| \sigma |\psi_\rho\rangle = |\langle\psi_\rho|\psi_\sigma\rangle|^2 \tag{5.34}$$

The motivation for exploring alternative loss functions comes from a known limitation of fidelity: it suffers from an orthogonality problem, where two orthogonal states are always evaluated to 0 overlap. For example, consider the target state $|t\rangle = |0000\rangle$ and two other states $|a\rangle = |1000\rangle$ and $|b\rangle = |1111\rangle$. The fidelity function F will return zero for both, i.e., $F(t, a) = F(t, b) = 0$, even though $|a\rangle$ is intuitively closer to $|t\rangle$ than $|b\rangle$ under other metrics, such as the Hamming distance.
Based on this observation, we decided to test the performance of the Wasserstein distance [24], which does not suffer from this limitation and may also help mitigate the barren plateau problem encountered in quantum machine learning [121].
The Wasserstein distance of order 1 also known as the quantum Earth Mover distance (D_{EM}) between two states ρ, σ is equal to the maximum difference between the quantum

states ρ, σ is equal to the maximum difference between the expectation values on ρ and σ of a quantum observable with Lipschitz constant [122] at most one:

$$D_{EM}(\rho, \sigma) = \max\{\text{tr}[(\rho - \sigma)H] : H \in O_n, ||H||_L \leq 1\} \tag{5.35}$$

this metric allows us to assign different distances to states that are orthogonal to the target since it becomes a quantum Hamming distance in those cases (e.g. $D_{EM}(|000\rangle, |001\rangle) < D_{EM}(|000\rangle, |111\rangle)$).
The EM distance has one main drawback: we need to run an optimization to identify the operator H, and this could take a lot of time given that it could be any operator that satisfies $||H||_L \leq 1$.
In the literature, this obstacle has been tackled with either some approximation, like considering only Pauli gates [123] instead of all possibilities, or with clever strategies.

In our case, after an analysis of how the EM Distance is applied to the autoencoder, we could find a good strategy not to waste computational resources. By simply limiting H to $\{\sigma_z, I\}$ and their combination, we were able to drastically reduce the computation time while always keeping the best operator in the group since the target state would always be $|0\rangle^{2^n}$ for some n trash qubits.

5.3.3 Experiments

Since the research is still ongoing, we can only disclose parts of the results we obtained from this project. In particular, we will describe the experimental setting and some of the results we obtained from the loss function study.

Experimental setting

The experiment was carried out using a quantum autoencoder model with a layered architecture composed of modular two-qubit composite gates. Each module is composed of two parametric $R_y(\theta)$ gates and two parametric $R_z(\theta)$ gates, one of each type on each qubit. Following these rotations, a CNOT is applied targeting the first qubit with the control on the second. A visual representation of the module is depicted in Figure 5.9.

This module is repeated multiple times in parallel, linking every second qubit with its following neighbor. This process is then repeated, starting from the second qubit instead of the first, to entangle each qubit with the qubits close to it. In general, we

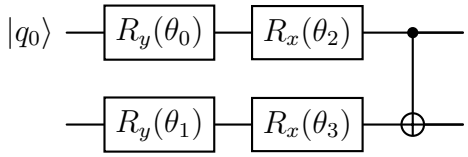


Figure 5.9: Visual representation of the modular two-qubit gate.

could say that this application of the modules follows what is known as a linear even-odd entanglement strategy, where we mix a single qubit rotation before the entangling two-qubit gate.
The layered application of this module means that the procedure is repeated multiple times. In the first part of our experiments, we concluded that a 3-layer depth could achieve a sufficient expressibility for the kind of inputs we were considering. One of the advantages of this modular circuit is that the number of two-qubit gates and parameters increases only linearly in the number of qubits and in the number of layer repetitions. This means that a more complex input could benefit from a deeper (and more expressive) architecture with only a linear increase in complexity. The full encoding architecture can be found in Figure 5.10.

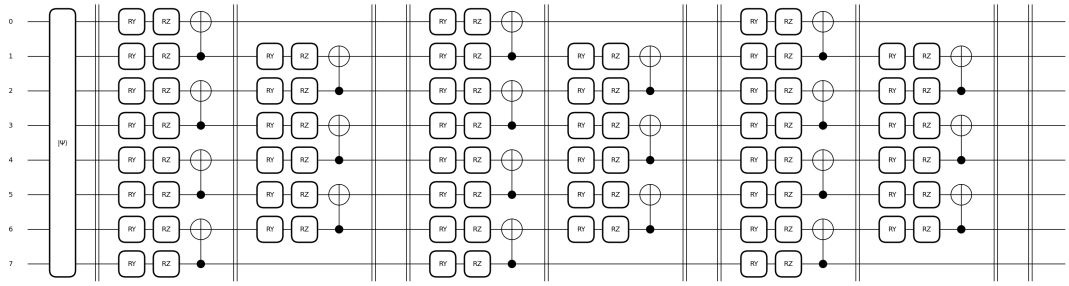


Figure 5.10: Full architecture of the encoder.

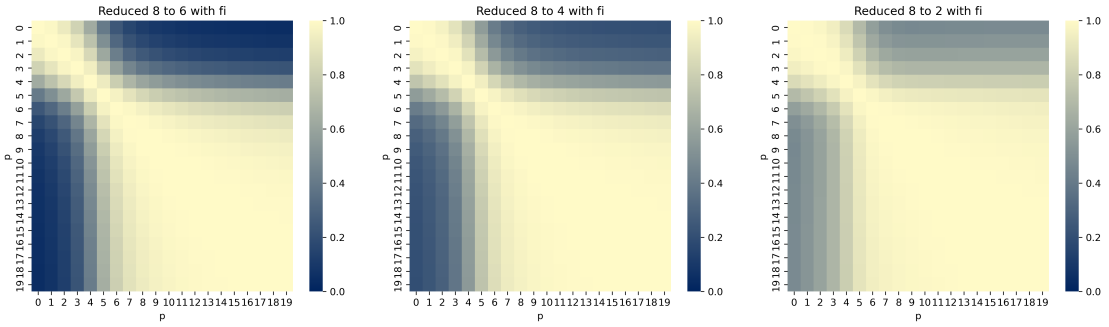
Since the input is assumed to be already prepared, it being a quantum state, we utilized the library Pennylane [120] and its dataset section to prepare the correct starting state. The output we receive from this architecture is dual: on one side, m of the original qubits will contain the compressed information, and we will call these qubits middle qubits since they live in the middle between the encoder and the decoder. All other t qubits are discarded after the compression and for this reason are called trash qubits. Against typical machine learning fashion, the training of the architecture is done by

forcing the trash space into the state $|0\rangle^t$ so that all information contained in the input state is moved into the middle qubits.

Results

We will discuss the results of the experiment under two aspects: first, we will discuss how well the trained model can compress the data and how well this compressed state can approximate the original one when passed through a SWAP test. Secondly, we will present the initial results on the comparison between loss functions.

We decided to train the models using four different loss functions: fidelity for the baseline and three different approaches to the EM distance. These approaches differ in the distance parameter, which regulates the maximum range of the qubit gate used: a distance parameter of one tells us that only single-qubit gates are going to be utilized, while a distance metric of eight (in our specific case) means that all qubits can interact directly with all others during the optimization of the EM distance. One other consequence of the distance parameter is the change in the loss function optimization complexity. The higher the parameter, the more operators are explored. In the following experiment, we worked with EM distances of two (EM2), three (EM3), and four (EM4).

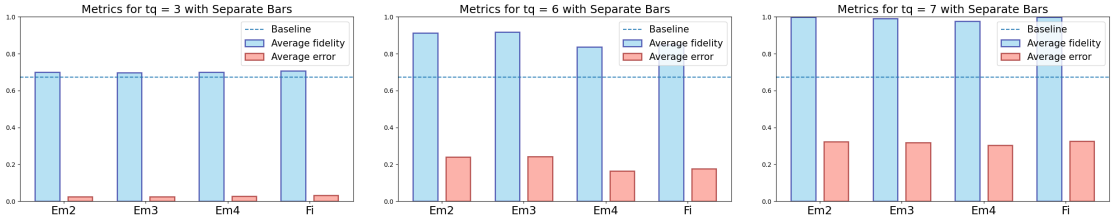


(a) Fm for tq=2 (b) Fm for tq=4 (c) Fm for tq=6

Figure 5.11: Three fidelity matrix (Fm) obtained by computing the fidelity between pairs of reduced states. You can see that by reducing the output size (increasing the trash qubits) the phase shift degrades in intensity but remains visible.

For the first phase, we present the results in Figure 5.11. There, we can see how the phase transition works in the compressed states. Clearly and predictably, when considering a bigger compression rate, the fidelity between different states increases on

average, signaling that there are fewer differences between the states. One simplistic metric we can use to evaluate these graphs is the average fidelity as well as the average difference between the phase transition graph of the original states and the one we produced (average error in figure 5.12). Using these metrics, we didn't find any substantial difference between the different loss functions, besides the fact that the more complex EM distances score an average fidelity closer to the original one with respect to the other EM distances.



(a) trash qubit = 3 (b) trash qubit = 6 (c) trash qubit = 7

Figure 5.12: Average fidelity and average error for each model (em2, em3, em4, fi) over trash qubit $tq \in \{3, 6, 7\}$

Our experiments showed that while reducing the sizes of states through an imperfect quantum autoencoder removes some of the distinction between the states (orthogonal states originally are not anymore), their pairwise comparison produces a result that follows the same structure as the original. We also noticed that while our architecture performs very well with a few trash qubits, it might not be expressive enough for more demanding compressions. We also took into consideration findings from [70] where a theoretical limit for errorless quantum state compression has been found. We were not able to reach the same results as the theoretical work, likely because of the highly heuristic approach of the classical optimization phase. Furthermore, the aforementioned work didn't give us an overall limit to compression but the maximum achievable by means of fidelity loss, meaning that there's nothing to the best of our knowledge that poses a theoretical limit when using different loss functions.

Subsequently, for the second phase, we looked at the performances of the two loss functions we tested. The metric we used to compare the efficacy of the models was by passing one state through the full encoder-decoder architecture and comparing the result to the input via fidelity. This approach allows us to gain an understanding of the amount of information lost during the encoding.
We trained multiple models with the same training loss and obtained the average re-

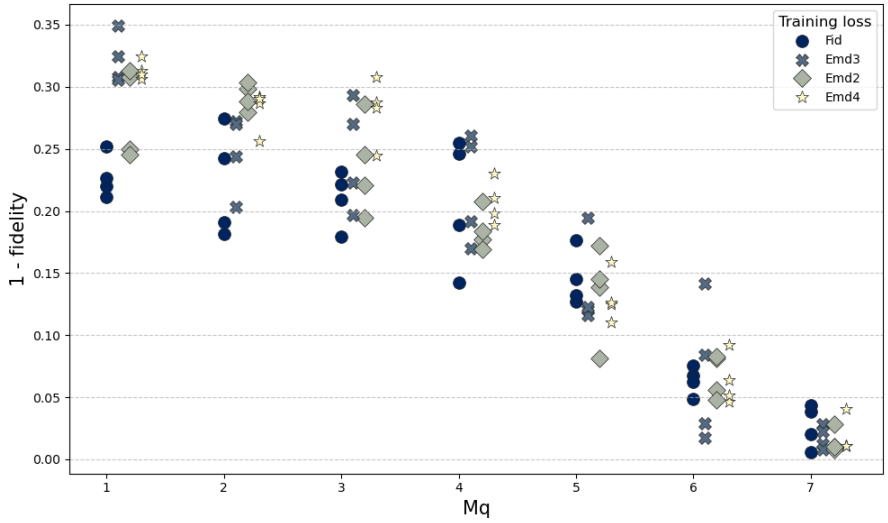


Figure 5.13: Average reconstruction fidelity over different numbers of middle qubits. Each point is an average of the fidelity obtained with a test set with a different training instance.

construction fidelity for each instance. Figure 5.13 shows how the performance changes as we increase the number of middle qubits. Optimally, we would have a perfect reconstruction for each model (fidelity of 1), but in reality, we observe that, while for less demanding compression this result might be achievable by all models, the higher the compression rate, the more information is lost.

From the average result of the models reported in table 5.6, we see that the performance of the models is very similar, and except for the $Mq = 1$ model, the performance of the other models is really close together, suggesting that there is no clear superior training loss function. Furthermore, it is worth adding that using only four training instances for each loss function is not sufficient to obtain a precise understanding of their performances.

Considering everything we discussed and the fact that $Emd4$ is not the most complete Earth mover possible, we believe more resources and experiments are needed to reach a more definite conclusion. As of now, we can limit our findings to the suggestion that EM distance doesn't seem to produce better results compared to fidelity and vice versa. As for the computation of the fidelity matrix and the original objective of the SWAP test, our results suggest that the best EM based model ($Emd4$) works slightly better than the best fidelity model 5.12c but even in this case, given the stochasticity of the training, more experiment are needed to reach a definitive conclusion.

Mq	1	2	3	4	5	6	7
Emd2	0.279	0.292	0.237	0.184	0.134	0.067	**0.014**
Emd3	0.322	0.247	0.246	0.218	0.138	0.068	0.018
Emd4	0.314	0.281	0.281	**0.207**	**0.13**	**0.063**	0.018
Fid	**0.227**	**0.222**	**0.21**	0.208	0.145	0.064	0.027

Table 5.6: Results table with the average reconstruction fidelity of each model. Highlighted are all the best performing results.

5.4 Benchmarking different technologies

5.4.1 Methods

The graph coloring problem consists of assigning a color to each graph node so that the colors of connected nodes are different. Finding a solution for k colors is called k-coloring, while minimizing the number of colors used is called the Chromatic Number Problem. The maximum independent set refers to a set of vertices with the largest number of vertices, where no two vertices are connected by edges. Here, the MIS is regarded as a sub-problem of the GCP because finding the maximum independent set of the original graph is equivalent to assigning a color to the nodes found, and this procedure can be repeated for the remaining nodes until all nodes have an assigned color.

For D-Wave's quantum annealers, the problem needs to be transformed into a QUBO formulation. Depending on the algorithm we use, we could end up with two main formulations. The first involves directly solving the problem as GCP itself, the second uses an iteration of the MIS formalization, which needs post-processing to get the final coloring solution. The MIS formulation leverages structural insights from graph theory, potentially offering performance advantages depending on the problem instance.

For gate-based quantum computers such as IBM's general-purpose quantum devices, the methodology involves more flexible circuit design techniques. Quantum circuits can be explicitly constructed to address the graph coloring problem by encoding the solution space into qubit states and using quantum gates to represent constraints and objective functions. Additionally, the QAOA [22] is employed as a key optimization method. QAOA is used to optimize the quantum circuits, refining the solution quality by variationally adjusting the parameters of the circuit to minimize the cost function associated with the problem.

QUBO

We first introduce the methods for converting the graph coloring problem and the maximum independent set problem into QUBO forms.

The QUBO model is expressed by the optimization problem [124]:

$$\min y = x^t Q x$$

where x is a vector of binary decision variables and Q is a square matrix of constants.

QUBO for GCP GCP aims to assign colors to the vertices of a graph such that no two adjacent vertices share the same color. The K-coloring problem specifically seeks to find a valid coloring using exactly K colors. This model has numerous applications, including frequency assignment in telecommunications and the design of printed circuit boards. The problem can be formulated as a satisfiability problem in the following manner:

Let $x_{ij} = 1$ if node i is assigned color j, and 0 otherwise.

- **Constraint 1:** Since each node must be colored, we have the constraints:

$$\sum_{j=1}^{K} x_{ij} = 1 \quad i = 1, ..., n$$

  where K is the number of colors, n is the number of nodes in the graph.

- **Constraint 2:** A feasible coloring, in which adjacent nodes are assigned different colors, is assured by imposing the constraints:

$$x_{ip} + x_{jp} \leq 1 \quad p = 1, ..., K$$

  for all adjacent nodes (i, j) in the graph.

QUBO for MIS The optimization form of the MIS, expressed in QUBO form as:

$$f_{MIS}(x) = -\sum_{i=1}^{n} x_i + \alpha \sum_{i,j \in E} x_i x_j (\alpha \geq 2, x \in 0, 1)$$

The size of the MIS matrix is proportional to the square of the number of vertices, whereas, for the GCP, it scales with the square of the number of vertices multiplied by

the number of colors. Thus, for the same graph instance, the QUBO matrix for the MIS problem is much smaller in dimension compared to that of the GCP. Even when factoring in that MIS requires $K-1$ iterations, the dimension of the problem diminishes each time, and given the modern constraint of quantum computing, the matrix size is a far more critical factor.

It is important to highlight that the GCP approach requires specifying the value of K (the number of colors) in advance in order to generate the matrix for computation. In contrast, the MIS method takes the problem graph as input and determines the value of K without any prior assumptions or presets.

D-Wave Quantum Annealer

In this study, we utilized the D-Wave quantum annealer to address the graph coloring problem by either directly solving it (GCP Approach) or by converting the problem into an MIS problem, followed by iterative subproblems to obtain the solution (Iterative MIS Approach). The specifics of both methods are outlined below.

GCP Approach This method begins by constructing a QUBO matrix from the problem graph and the predefined number of colors K. The matrix is then fed into the quantum annealing machine's sampler. Multiple samples are taken within the specified annealing time, and the result with the lowest energy value is selected as the final solution.

Iterative MIS Approach The GCP can be solved indirectly through the MIS problem. After each MIS is found and colored, remove the vertices in the MIS from the graph. This reduces the problem size, and the remaining subgraph contains only the uncolored vertices. Repeat the process of finding MISs, in the current MIS of each iteration, a new color is assigned to the vertices until no vertices are left. The result of each iteration corresponds to the original graph, and finally, the color solution of all vertices is found.

IBM Quantum Computer

In gate-based quantum computing, mainly two methods are used to design quantum circuits for the graph coloring problem. The first method involves directly constructing a quantum circuit based on the specific instance called the NK method. The second

method converts the problem into a QUBO form and then builds a quantum circuit to solve the QUBO problem using QAOA. Below is a brief introduction to both approaches.

NK Method We utilized the NK method proposed in Clerc's work [57]. The NK method aims to find a valid K-coloring for a given graph with N nodes without assuming any specific graph structure, thanks to the unique characteristics of quantum gate circuits.

QAOA Method Based on the common basis of using QAOA principles and quantum circuits to solve optimization tasks, there are two methods to implement QAOA, namely QAOA Ansatz and QAOA Optimizer. QAOA Ansatz is a quantum algorithm where a parameterized quantum circuit is designed to approximate solutions to optimization problems. This method focuses on the quantum aspect of the problem, utilizing quantum gates and circuits to encode and solve the optimization task. QAOA Optimizer combines the QAOA Ansatz with classical optimization techniques. In this approach, QAOA is used to prepare a quantum state that approximates the solution, while the MinimumEigenOptimizer is a classical algorithm that fine-tunes the parameters of the QAOA circuit to minimize the eigenvalue of the problem Hamiltonian.

Experimental setup

We performed experiments on multiple architectures and with multiple techniques. The problem focuses on first solving the Chromatic Number Problem through MIS and then comparing the performance of the GCP approach with the chromatic number found. While different graph layouts were considered during the experimentation, here we discuss only the fully connected versions (unless it explicitly says otherwise), meaning that all graphs contain an edge between each pair of vertices and their chromatic number is equal to the number of vertices. The purpose of this seemingly naive setup is to reach the hardware limits by producing the most complex QUBO matrix with the smallest number of vertices, as well as having a standardized layout.

D-Wave Quantum Annealer setup We use D-Wave's *Advantage_system4.1* and *Advantage2_prototype2.4*. The systems have respectively 5627 and 1217 qubits with connectivity degrees of 15 and 20, arranged in two different topologies named Pegasus and Zephyr. The Advantage system has a typical programming time of 14,072.88 microseconds, while Advantage2 has 18,231.11 microseconds. Both systems operate with a default

annealing time of 20 microseconds, 500 reads, and experience a 20.54 microseconds QPU delay time.

IBM Quantum Computer setup The IBM Sherbrooke quantum computer (*ibm_sherbrooke*), equipped with a 127-qubit Eagle processor, is optimized for error mitigation and delivers a Quantum Volume of 128.

5.4.2 D-Wave annealer results

Iterative MIS Approach

We consider graphs with the number of edges ranging from 3 to 170. As the number of nodes increases, the QPU time also increases, which is expected since the MIS is an iterative procedure. For smaller graphs, the QPU time grows steadily but remains under 1500 ms up to 19 nodes. As the graph size increases, QPU time starts rising more rapidly, peaking at over 3500 ms for graphs with 60 nodes. For the largest graph we chose, with 100 nodes, the QPU time is slightly over 3500 ms, which suggests the annealer has reached its computational limit.

The total time, which includes both the QPU time and overhead (such as classical processing and communication), shows a steep increase. For instance, for smaller graphs (3–17 nodes), the total time is already around 30,000–80,000 ms and escalates rapidly beyond this point. For the largest graph with 100 nodes, the total time reaches over 225,000 ms (around 3.75 minutes), showing that classical overhead becomes a major bottleneck as problem sizes grow.

The number of qubits used scales with the number of nodes from the table. We can see that for small graphs (up to 20 nodes), the qubit usage is manageable (around 55 qubits), but for larger graphs (such as 60 nodes), the system uses 554 qubits, and for 100 nodes, it uses 1125 qubits. The maximum successful graph is for 170 nodes and 14365 edges, it uses 3903 qubits (*Advantage*). When testing a fully connected graph with 180 vertices, the problem could not be embedded because the number of qubits required exceeded the approximately 5,600 qubits available on the existing quantum annealing machine. This growth in qubit usage highlights the physical limitations of the current D-Wave system, which may restrict the size of the graphs that can be effectively solved as the number of vertices increases.

When the result of the computation became different from the expectation, we extended the default annealing time from 10ms to 30ms, obtaining results consistent with

the prediction.

Notably, the QPU time did not increase dramatically with the number of nodes and edges. This suggests that quantum computing could have the potential to handle large-scale problems efficiently in terms of processing time. However, the accuracy of the results, particularly as the complexity of the problem grows, might still need further improvement.

It is important to note that in our selected examples, when the number of nodes exceeds 30, the results obtained from the algorithm deviate from expectations. This inconsistency suggests that an error occurred during the solution process, leading to incorrect outcomes. To address this issue, we increased the number of reads, extended the annealing time, and experimented with various configurations. However, these adjustments did not yield the correct result so far.

GCP Approach

Based on the K value obtained through the previous MIS method, we reformulated the QUBO matrix using the same graph and subsequently applied the GCP algorithm.

As the number of nodes increased, both QPU time and total time generally increased, reflecting the growing complexity and computational demands associated with larger graphs. The error count remained at zero for smaller graphs, indicating successful color assignments. However, as the size increased, some instances showed errors, particularly notable in graphs with 10 nodes and above. This suggests that larger graphs may present challenges in achieving optimal solutions. The matrix size increases significantly with the number of nodes, contributing to the observed higher processing times. A precise assessment of the computational resources needed to run is crucial for assessing the feasibility of implementing such algorithms on quantum hardware.

Comparison of the Two Methods

When comparing the results of the two approaches for solving fully connected graphs, focusing on time and qubit usage as shown in Fig. 5.14, we observed that in most cases, the QPU and total time of the GCP algorithm were significantly lower than those of the MIS method. This difference stems from the iterative nature of the MIS method, which requires more preprocessing computations. At the same time, the number of qubits required by the GCP method grows much faster than that of the MIS method, reaching 4601 qubits at 16 nodes, close to the maximum number of qubits that could be used.

As the graph size increased, especially at 16 nodes, the total time of the GCP algorithm exceeded that of the MIS method. Referring to the comparison of the number of qubits, this shows that the demand for qubit resources due to GCP increases significantly with the increase in graph size. Therefore, the increase in the time spent by GCP can be attributed to the complexity involved in finding a suitable embedding for the larger qubit resources required.

Overall, while GCP shows excellent efficiency on smaller graphs, its performance in terms of total time may degrade as the problem size increases, indicating the need to optimize resource allocation and embedding strategies for larger graphs. The MIS method has an advantage in terms of problem size and can solve the same problem using fewer qubits, but over multiple runs.

Pegasus and Zephyr Topologies

We explored the performance of quantum annealers utilizing different topologies, Pegasus and Zephyr, on the same graphs using the GCP method. Overall, the Zephyr topology demonstrates a clear advantage over Pegasus by requiring less time and fewer qubits to solve the same problem, despite having fewer qubit resources available compared to Pegasus. It is anticipated that as Zephyr scales in qubit capacity, its performance will further surpass that of the Pegasus topology, making it an even more effective tool for quantum computation tasks.

5.4.3 IBM Gate based results

Based on the two methods proposed above, we conducted experiments on the simulator and the actual gate-based quantum computer. The experiments on the simulation were performed with and without noise using multiple simulation techniques, while the experiments on the real hardware were performed on the freely available *ibm_sherbrooke* quantum hardware. During the experiments, we also considered various optimization levels for our circuit.

NK Method

Due to limited simulation capabilities, we mostly run experiments on a 3-node graph. The quantum circuit generated for the triangle graph has a circuit depth of 58 and a circuit width of 18 qubits. The high complexity of the circuit shows the rapid increase in resource requirements even for relatively small graphs. When adding more nodes or

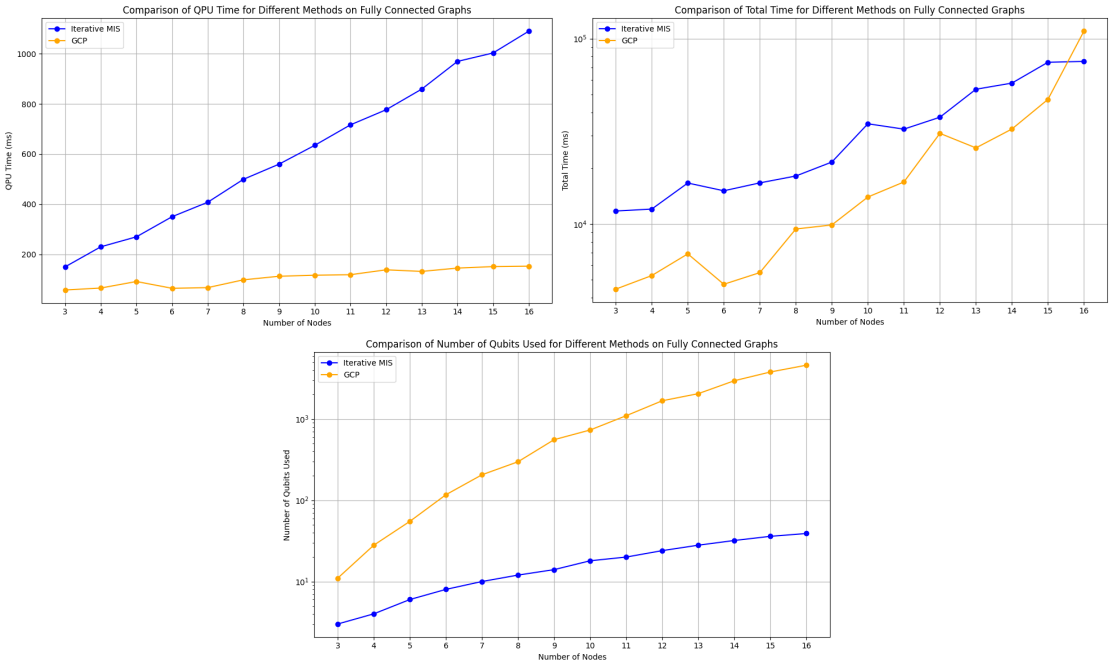


Figure 5.14: Comparison of QPU time, Total time, and Number of Qubits Used for Different Methods on Fully Connected Graphs using D'Wave quantum annealer. The graph clearly shows that CGP is faster in terms of QPU time but its total time increases fast for bigger problems, probably because of the high number of qubits used.

colors, the circuit complexity grows exponentially, limiting the practical scalability of the method.

Simulators without noise For 1000 measurement shots, the resulting output, as depicted in Fig. 5.15, confirms the method's validity. If we exclude all 0 bitstrings, all other results represent valid colorings of the triangle graph. For instance, the second bitstring, 001010100, can be interpreted from right to left. This corresponds to the first node being assigned color 3, the second node color 2, and the third node color 1, all in compliance with the graph coloring constraints.

While the method successfully produces correct results for the triangle graph, it is important to note that the number of qubits required and the circuit depth increase rapidly as more nodes or colors are added to the graph. In the triangular graph example, 18 qubits are necessary to execute the quantum circuit. However, when a fourth node is introduced, the number of required qubits increases to 28, nearing the maximum qubit

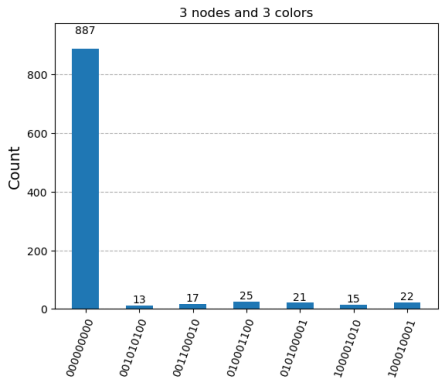


Figure 5.15: Results of 3 nodes and 3 colors graph for 1000 shots on simulator without noise.

capacity of the simulator.

In cases where we simulate a graph with four nodes and four colors, the required 32 qubits surpass the simulator's limit of 30 qubits. Despite the relatively small size of the graph, the qubit requirements exceed the capabilities of current quantum simulators. Consequently, our ability to simulate larger graphs is constrained, and we are currently limited to testing on smaller examples, such as the triangle graph.

This highlights that while this method works for small graphs, scalability remains a significant challenge due to the exponential growth in qubit requirements, which makes it impractical to simulate larger graphs.

We utilized several *Qiskit* simulators and successfully obtained correct simulation results using the *automatic*, *statevector*, and *matrix_product_state* simulators. These simulators proved capable of handling the required qubit numbers and produced valid outputs. However, other simulators either exceeded their qubit capacity or failed to produce satisfactory results.

Simulators with noise models Quantum hardware is inherently noisy, and various errors, such as decoherence, gate infidelities, and measurement errors, can affect computation results. To simulate realistic conditions, we first defined a basic bit-flip error noise model with specific error probabilities for resets, measurements, and quantum gates.

When running the quantum circuit under the noise model, we observed a significant degradation in the accuracy of the results compared to the ideal noiseless simulations. The added noise caused a notable increase in incorrect results, with a higher proportion

of invalid colorings. Further examination reveals that none of the top 20 result groups with the highest probability contained a correct solution. Additionally, correct solutions appeared only 2 to 3 times across the result probabilities, with some correct solutions never appearing at all. This is to be expected, as noise can lead to bit flips, decoherence, and other issues that disrupt the logical operations in the circuit.

Furthermore, we generated a realistic simulation of the thermal relaxation errors noise model in quantum computing systems and it has been used in the simulation. The results show higher variance and all correct results appear with similar probability. The simulation times for each simulator with noise models are presented in Tab. 5.7. The matrix product state simulator is the best performer in terms of speed for both bit-flip noise and thermal relaxation noise. The state vector simulator provides decent performance but lags behind in speed, especially with thermal relaxation noise.

Despite the noise, the simulator was still able to produce a subset of correct results. This reiterates the fact that error mitigation or error correction techniques are needed to improve the results but stable enough hardware could solve simple problems without too much overhead.

Table 5.7: Simulation results for different simulators with noise.The accuracy here refers to the percentage of correct results in the total number of shots.

	bit-flip noise		**thermal relaxation noise**	
	Time (s)	**Accuracy (%)**	**Time (s)**	**Accuracy (%)**
Automatic	29.46	1.2	321.25	2.5
Statevector	29.30	1.3	186.86	3.0
Matrix Product State	9.84	1.6	101.96	3.6

Real Quantum Device We submit the program onto the IBM quantum platform and run it on real quantum computers. The performance of the hardware is here represented by the accuracy measure that refers to the percentage of correct results in the total number of shots and by the time a QPU is committed to completing a workload.

When applying the NK method to solve a graph coloring problem with three vertices, three edges, and three colors on a real quantum computer, we evaluated various parameters. The average accuracy was close to the accuracy after adding the bit-flip noise model above, and determined that the optimal configuration for running the quantum circuit on the *ibm_sherbrooke* quantum computer is to use optimization level 2, *Sabre*

placement, and *StochasticSwap* routing. This configuration offers the best balance of accuracy and execution time. Using only optimization level 2 we obtained an accuracy of 1.2% with a short execution time of 11 seconds, while *Sabre* placement further enhances accuracy to 1.6% without increasing the runtime in Fig. 5.16. Among the routing methods, *StochasticSwap* delivers the highest accuracy at 1.5%, matching the execution time of other methods. Although this setup optimizes both accuracy and speed, it's important to note that, given the simplicity of the graph problem, we were able to assess accuracy by comparing configurations against known correct results. However, for more complex problems, the probability of finding the correct solution remains low, and the computation time is significantly longer than typical microsecond-level quantum tasks, thus showing that the current overall performance of the NK method is very limited.

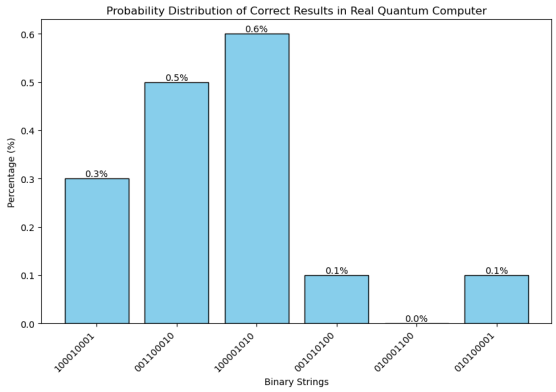


Figure 5.16: Correct results of 3-nodes and 3-colors graph for 1000 shots on a real quantum device with optimization level 2, Sabre placement. The accuracy of each possible solution is very low due to hardware limitations.

QAOA Method

Using the QAOA Method we were able to study problems with graphs up to 5 nodes but in order to facilitate comparison with the NK method above, we first refer to the same triangular graph mentioned before.

QAOA Ansatz We begin by generating a QUBO matrix from the problem graph and converting it into a cost function Hamiltonian using Pauli gates transformation. Subsequently, we construct the corresponding quantum circuit and define the loss function. An optimizer is utilized to iteratively minimize the loss and obtain the optimal quantum circuit parameter and finally, the optimized circuit solves the problem.

We run the QAOA Ansatz on the simulator with different circuit depths, using optimization level 0 and 10,000 shots for both iteration and execution. The iteration process employed the Constrained Optimization BY Linear Approximation (COBYLA) optimizer with a termination tolerance set to 0.0001. Results for the triangular graph are shown in Tab. 5.8 while the loss function can be found in Fig. 5.17.

Table 5.8: Results of QAOA Ansatz on the simulator with different circuit depth with optimal level 0 and 10000 shots.

	Depth		
	1	**2**	**3**
Accuracy(%)	1.11	10.33	6.18
Iteration time(s)	2.64	4.66	7.04
Execute time(s)	0.05	0.06	0.06

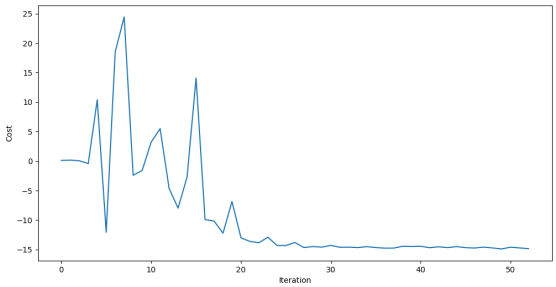


Figure 5.17: Loss function curve during optimization. We utilized a COBYLA optimizer termination tolerance set to 0.0001, optimization lavel 0 and 10,000 shots.

Accuracy improves significantly as the circuit depth increases from 1 to 2, jumping from 1.11% to 10.33%. However, accuracy drops to 6.18% at depth 3. This suggests that increasing the circuit depth does not always lead to better performance. The increased noise affecting a deeper circuit might explain the reduced accuracy at higher depths.

We then save the optimized circuit obtained from the simulator and execute it for 1000, 2000, and 10000 shots on an actual quantum computer with different configurations. When using a *trivial* layout with different shot counts, the accuracy improves slightly as the number of shots increases; meanwhile, the number of shots does not significantly impact execution time. The accuracy increases slightly from 1.13% for the *trivial* layout to 1.29% for the *sabre* layout with the same 10000 shots. Using Optimal

Level 2 routing yields an accuracy of 1.25%, while *StochasticSwap* routing gives a slightly lower accuracy of 1.19%. As expected, the accuracy of running on an actual quantum computer is still far lower than that achieved on a simulator due to the presence of errors and noise, and gate-based quantum hardware is not yet a viable solution to solve this kind of problem.

QAOA Optimizer Since the optimization has been improved internally, under the premise of using the solution of the actual solver as a reference for verification, the highest probability solution given by this quantum algorithm is the correct solution. All correct solutions are ranked at the top and can be distinguished from other incorrect solutions. Compared with the above method, the sorting verification step is omitted, so the accuracy is no longer calculated here.

The Tab. 5.9 compares the performance of two different optimizers, COBYLA and Sequential Least Squares Programming (SLSQP), when applied to the QAOA for problem instances of a fully connected graph with 3, 4, and 5 nodes. The metrics compared include the Exact solver time and QAOA solver time, each given in seconds. The exact solver's time scales rapidly with problem size, becoming computationally prohibitive for larger instances (5 nodes). SLSQP outperforms COBYLA in terms of time efficiency for both 3-node and 4-node instances. However, for 5 nodes, QAOA did not provide results, possibly due to the difficulty of solving larger instances with this approach.

Table 5.9: Results of QAOA Optimizer on the simulator. CO refers to the COBYLA optimizer and SL to the SLSQP

	3 nodes, k=3		**4 nodes, k=4**		**5 nodes, k=5**	
	CO.	**SL.**	**CO.**	**SL.**	**CO.**	**SL.**
Exact solver time(s)	0.01	0.01	0.29	0.29	191.08	194.16
QAOA solver time(s)	0.70	0.32	50.34	33.62	NA	NA

Both methods presented operate within the QAOA framework, which is designed to solve combinatorial optimization problems by constructing a parameterized quantum circuit. This circuit alternates between layers of problem-specific and mixing Hamiltonians, with the primary goal of optimizing circuit parameters to maximize the probability

of measuring the optimal or near-optimal solution. In both approaches, the optimization problem is encoded into the quantum circuit, with the problem Hamiltonian representing the cost function integrated into the circuit design. These methods leverage quantum gates and circuits to explore the solution space and address complex combinatorial problems. However, in practical applications, due to the limited resources of quantum computers, they can only solve problems with small graphs.

5.4.4 Conclusion

We explored several methods to solve the graph coloring problem using the D-Wave quantum annealer and the IBM universal quantum computer on simulators and real devices. On D-Wave, we successfully solved a fully connected graph with 170 nodes and a graph with 300 nodes but fewer edges using the MIS method. Along with a fully connected graph with 120 nodes, they were able to directly solve the graph coloring problem, though they produced some inaccurate results. On IBM's platform, the largest graph simulated was a fully connected graph with 5 nodes, while a 3-node graph was executed and analyzed on the actual quantum processor. However, the IBM quantum computer exhibited significant error rates and relatively low accuracy in these tests. In terms of execution speed, the D-Wave system required 150 ms of quantum processing time for the 3-node graph, whereas IBM's quantum computer took at least 2 seconds. Based on these experiments, we conclude that the D-Wave quantum annealer outperforms the IBM universal quantum machine for solving graph coloring problems in terms of execution time, convenience, and accuracy.

Both D-Wave's quantum annealers and IBM's gate-based quantum computers encounter several challenges, primarily stemming from issues like errors and noise, that hinder their performance, particularly when applied to larger and more complex problems. These limitations impact the scalability, reliability, and efficiency of the platforms, making them difficult to apply to large-scale, real-world scenarios. Due to the limitations of the freely available resources used in this experiment, the quantum devices exhibited significantly limited computational capabilities. For example, D-Wave's device could process comparatively very large graphs with more than a hundred nodes, albeit one at a time, while IBM's system was restricted to solving only very small graphs with 3 or 5 nodes. This discrepancy significantly limited the number of comparable examples and problem sizes that could be evaluated. The more detailed configurations, such as quantum embedding and mapping methods, were not studied in this research. There-

fore, future research will continue to study the configurations and advantages of each quantum device in depth. Experiments and comparisons with other types of quantum devices can also be performed on specific problems. At the same time, we are working on methods to optimize universal quantum circuits to improve IBM's accuracy and applicability. Quantum error correction and error mitigation techniques are also critical to overcoming noise and gate errors. Developing hybrid algorithms that combine the strengths of quantum computing with classical methods could extend the problem sizes that can be handled and may also leverage quantum systems to provide speedups over classical algorithms.

Overall, both D-Wave's quantum annealers and IBM's gate-based quantum processors demonstrate potential for solving NP-complete problems, but their current capabilities are limited by issues of scalability, noise, and hardware constraints. Advancements in qubit technology, error correction, and algorithm development are critical to overcoming existing barriers. By pursuing hybrid approaches and improved quantum algorithms, we could see the possibility of quantum computing becoming a practical tool for tackling complex combinatorial optimization problems.

Chapter 6

Quantum annealing approaches

6.1 Quantum Annealing for K-means

6.1.1 Preliminaries

In classical machine learning, clustering is one of the most studied problems, given its pervasive nature in everyday life. The problem asks us to divide a dataset into groups, such that each group contains data points that are considered similar. It is used in the context of exploratory analysis, statistical analysis, and many other fields, including pattern recognition, information retrieval, and more [125].
One fundamental property of the clustering problem is that while a cluster is defined as a group of data points, there is no strict definition of what connects the data points in the group. For this reason, there are a lot of different ways to solve the clustering problem based on which properties the algorithm uses to recognize the clusters. Furthermore, clustering is, for the most part, an unsupervised learning problem without an objective solution that tells us which approach is the best one.
In this context, we focus on one specific clustering technique called K-Means [126]. This algorithm solves an NP-hard version of the clustering problem that can efficiently converge thanks to heuristic algorithms. The algorithm follows this scheme:
Given a set of datapoint $D = \{P^1, \dots, P^N\}$ where each P^i is a n-dimensional vector, K-means divides D into K different clusters $S = \{s_1, \dots, s_K\}$ so that the intra-cluster

sum of squares is minimized. The objective function is:

$$\arg\min_{s} \sum_{i=1}^{K} \sum_{P \in s_i} ||P - \mu_i||^2 = \arg\min_{s} \sum_{i=1}^{K} |s_i| \text{Var } s_i \tag{6.1}$$

Where μ_i is the centroid of points in s_i:

$$\mu_i = \frac{1}{|s_i|} \sum_{x \in s_i} x \tag{6.2}$$

with $|s_i|$ the size of s_i and $|| \cdot ||$ the L^2 norm. Given that the total variance is constant, this approach is equivalent to reducing the pairwise squared deviations of points in these same clusters or maximizing the sum of squared deviations between points in different clusters (maximization of inter-cluster distance).
The Lloyd implementation of the K-means algorithm (also known as the naive implementation) assigns the centroid to points in space with certain criteria [127] and assigns the data points to them based on the Euclidean distance between the points and the centroid. Once the points are assigned to the clusters, each centroid is computed again and updated. This process is repeated until no change is found between iterations.
While this approach is theoretically NP-hard, the use of heuristics like Lloyd's algorithm improves the complexity to a super-polynomial worst case. Furthermore, for this algorithm, the worst-case scenario is so difficult to encounter that for practical application, Lloyd's algorithm is considered to have 'linear' complexity.

6.1.2 Methods

Our objective is to propose another approach that differs from Lloyd's algorithm and that allows us to shift the worst-case complexity from the time domain to the memory. We propose the use of quantum annealers to solve the centroid-based formulation of the clustering problem. Quantum annealers have the ability to solve NP-hard optimization problems in an efficient way, with the downside of the necessary transformation from the original problem to a Quadratic Binary Optimization Problem (QUBO).
This subsection is dedicated to transforming the K-means algorithm into a QUBO problem, analyzing the advantages and disadvantages of this approach.

Let's define the building block we will use to describe our model: first, we have the dataset composed of the N points $D : \{P^1, \ldots, P^N\}$, each point is defined in an M-dimensional space as $P^n = \{p_1^n, \ldots, p_M^n\}$.

For the K-Means algorithm, we will also have a K parameter that defines the number of clusters and consequently the number of centroids $C : \{C^1, \ldots, C^K\}$ where $C^k = \{c_1^k, \ldots, c_M^k\}$.

The K-Means algorithm works by randomly positioning the centroids, assigning each point to the nearest centroid, and then moving the centroid to the center of the shape and repeating until convergence. for each point we consider the K variables $\eta_k^n \in \{0, 1\}$ that indicates if the point P^n belongs to the centroid C^k.
In our version, the centroids are not assigned a random position and iteratively improved upon, but we instead consider a loss function that tries to minimize the distance between the data points of a cluster and its centroid without specifying the position of the latter until the end of the computation.

Objective function

We define the distance according to the limitation of the QUBO formulation as:

$$d(P^i, P^j) = \sum_m^M (p_m^i - p_m^j)^2 \tag{6.3}$$

The inter-cluster distance (Inter-CD) between the points and the centroids can be computed as

$$\text{Inter-CD} = \sum_k^K \sum_n^N d(C^k, P^n)(1 - \eta_k^n) \tag{6.4}$$

while the intra-cluster distance (Intra-CD) between the points and the centroids is:

$$\text{Intra-CD} = \sum_k^K \sum_n^N d(C^k, P^n)\eta_k^n \tag{6.5}$$

We need to minimize the intra-cluster distance and maximize the inter-cluster distance. We can do either:

$$\min \sum_k^K \sum_n^N d(C^k, P^n)\eta_k^n \tag{6.6}$$

$$\max \sum_k^K \sum_n^N d(C^k, P^n)(1 - \eta_k^n) \tag{6.7}$$

Given that the two formulations are equivalent for now, we will focus on the minimization. To complete it, we need to add some constraints and prepare them as penalty terms suited for a QUBO formulation.

Single Cluster constraint

As we mentioned before, we have the variables η_k^n that indicate the cluster a point P^i belongs to. Obviously, this should be only one cluster, and we need to reflect this limitation in the form of a constraint. If we consider the vector $H^n = \{\eta_1^n, \ldots, \eta_K^n\}^T$ with $n = 1, \ldots, N$ we can write the constraints as:

$$H^{nT} H^n = 1 \tag{6.8}$$

Or equivalently:

$$\forall n \in \{1, \ldots, N\} : \quad \sum_{k=1}^{K} \eta_k^n \eta_k^n = \sum_{k=1}^{K} \eta_k^n = 1 \tag{6.9}$$

This becomes a constraint:

$$\begin{aligned} &\forall n : \sum_{k=1}^{K} \eta_k^n - 1 = 0 \\ &\sum_{n}^{N} (\sum_{k=1}^{K} \eta_k^n - 1)^2 \\ &\sum_{n}^{N} (\sum_{k_1=1}^{K} \sum_{k_2=1}^{K} \eta_{k_1}^n \eta_{k_2}^n - 2 \sum_{k=1}^{K} \eta_k^n + 1) \end{aligned} \tag{6.10}$$

Centroid coordinates

The centroids should be limited in the space they can be positioned, the same as the range of the points. Making the assumption of normalized coordinates for the points $\forall n \in \{1, \ldots, N\}, \forall m \in \{1, \ldots, M\} : p_m^n \in [0, 1]$ (a custom assumption for datapoints used in a machine learning model) we can limit the coordinates of the centroids:

$$\forall k \in \{1, \ldots, K\}, \forall m \in \{1, \ldots, M\} : \quad c_m^k \in [0, 1] \tag{6.11}$$

or equivalently:

$$\forall k \in \{1, \dots, K\}, \forall m \in \{1, \dots, M\} : \quad 0 \leq c_m^k \leq 1 \tag{6.12}$$

Binary variables

The target variables for this formulation are: c_m^k and n_k^n. The total number of variables can be expressed as $KMR + KN$, where K is the number of clusters, M is the dimensionality of the space, N is the number of datapoints, and R is the number of binary variables used to represent the position of the centroids. R is used to binarize c_m^k in the following way:

$$c_m^k = \sum_{r=0}^{R} 2^{-r} \gamma_r^{k,m} \tag{6.13}$$

Where $\gamma_r^{k,m} \in \{0, 1\}$.
This binarization complies with the equation 6.12 for $r \to \infty$, for our purpose we consider R such as $1 - \sum_{r=1}^{R} 2^{-r} = 2^{-R} = \epsilon$ where ϵ is small enough that it can be discarded. This approximation should be done case by case, taking also into consideration that ϵ is also the precision with which the position of the centroid is described. Equation 6.12 is changed in the QUBO formulation to:

$$\forall k \in \{1, \dots, K\}, \forall m \in \{1, \dots, M\} : \quad 0 + 2^{-R} \leq c_m^k \leq 1 \tag{6.14}$$

6.1.3 QUBO objective function

Here we rewrite the objective function in equation 6.6, substitute everything we can, and resolve it to the correct form to compute the QUBO matrix.

$$\begin{aligned}
&\min \sum_{k}^{K} \sum_{n}^{N} d(C^k, P^n) \eta_k^n \\
&= \min \sum_{k}^{K} \sum_{n}^{N} \eta_k^n \sum_{m}^{M} (c_m^k - p_m^n)^2 \\
&= \min \sum_{k}^{K} \sum_{n}^{N} \eta_k^n \sum_{m}^{M} ({c_m^k}^2 - 2 c_m^k p_m^n + {p_m^n}^2)
\end{aligned}$$

$$= \min \sum_{k}^{K} \sum_{n}^{N} \eta_k^n \sum_{m}^{M} ((\sum_{r=0}^{R} 2^{-r} \gamma_r^{k,m})^2 - 2p_m^n (\sum_{r=0}^{R} 2^{-r} \gamma_r^{k,m}) + {p_m^n}^2) \tag{6.15}$$

Following this we separate every term in is own sum:

$$\begin{aligned} = \min & \sum_{k}^{K} \sum_{n}^{N} \sum_{m}^{M} \sum_{r_1=0}^{R} \sum_{r_2=0}^{R} 2^{-r_1-r_2} \gamma_{r_1}^{k,m} \gamma_{r_2}^{k,m} \eta_k^n \\ & - \sum_{k}^{K} \sum_{n}^{N} \sum_{m}^{M} \sum_{r=0}^{R} 2^{-r+1} p_m^n \gamma_r^{k,m} \eta_k^n \\ & + \sum_{k}^{K} \sum_{n}^{N} \sum_{m}^{M} {p_m^n}^2 \eta_k^n \end{aligned} \tag{6.16}$$

We can see here that the fourth term of the objective function has a cubic term $\gamma_{r_1}^{k,m} \gamma_{r_2}^{k,m} \eta_k^n$. Since this is not allowed in a QUBO formulation, we utilize new support variables $\xi_{r_1,r_2}^{k,m}$ that are gonna substitute the product $\gamma_{r_1}^{k,m} \gamma_{r_2}^{k,m}$ in the first and fourth term. We then need to add a new constraint that ties $\xi_{r_1,r_2}^{k,m}$ to the other variable:

$$\forall k, m, r_1, r_2 : \xi_{r_1,r_2}^{k,m} = \gamma_{r_1}^{k,m} \gamma_{r_2}^{k,m} \tag{6.17}$$

And let's rewrite it in a format compatible with the QUBO as described in [128]:

$$\sum_{k}^{K} \sum_{m}^{M} \sum_{r_1=0}^{R} \sum_{r_2=0}^{R} \gamma_{r_1}^{k,m} \gamma_{r_2}^{k,m} - 2\gamma_{r_1}^{k,m} \xi_{r_1,r_2}^{k,m} - 2\gamma_{r_2}^{k,m} \xi_{r_1,r_2}^{k,m} + 3\xi_{r_1,r_2}^{k,m} \tag{6.18}$$

Finally, we add this constraint and the one in equation 6.9 to the full formula, introducing the variables ρ as regularization parameters:

$$\begin{aligned} = \min & \sum_{k}^{K} \sum_{n}^{N} \sum_{m}^{M} \sum_{r_1=0}^{R} \sum_{r_2=0}^{R} 2^{-r_1-r_2} \xi_{r_1,r_2}^{k,m} \eta_k^n \\ & - \sum_{k}^{K} \sum_{n}^{N} \sum_{m}^{M} \sum_{r=0}^{R} 2^{-r+1} p_m^n \gamma_r^{k,m} \eta_k^n \\ & + \sum_{k}^{K} \sum_{n}^{N} \sum_{m}^{M} {p_m^n}^2 \eta_k^n \\ & + \rho_{clust} N \end{aligned}$$

$$
\begin{aligned}
&-2\rho_{clust}\sum_{n}^{N}\sum_{k=1}^{K}\eta_k^n \\
&+\rho_{clust}\sum_{n}^{N}\sum_{k_1=1}^{K}\sum_{k_2=1}^{K}\eta_{k_1}^n\eta_{k_2}^n \\
&+\rho_{triple}\sum_{k}^{K}\sum_{m}^{M}\sum_{r_1=0}^{R}\sum_{r_2=0}^{R}\gamma_{r_1}^{k,m}\gamma_{r_2}^{k,m} \\
&-2\rho_{triple}\sum_{k}^{K}\sum_{m}^{M}\sum_{r_1=0}^{R}\sum_{r_2=0}^{R}\gamma_{r_1}^{k,m}\xi_{r_1,r_2}^{k,m} \\
&-2\rho_{triple}\sum_{k}^{K}\sum_{m}^{M}\sum_{r_1=0}^{R}\sum_{r_2=0}^{R}\gamma_{r_2}^{k,m}\xi_{r_1,r_2}^{k,m} \\
&+3\rho_{triple}\sum_{k}^{K}\sum_{m}^{M}\sum_{r_1=0}^{R}\sum_{r_2=0}^{R}\xi_{r_1,r_2}^{k,m}
\end{aligned}
\tag{6.19}
$$

6.1.4 Complexity analysis

The computational complexity of the K-means algorithm is a well-studied topic in both theoretical and applied machine learning. The standard implementation we discussed above (Lloyd's algorithm) operates iteratively in two steps. Each iteration involves computing distances between n data points and k centroids in d-dimensional space, resulting in a per-iteration cost of $O(nkd)$. The centroid update step is less expensive, typically $O(nd)$, so the total cost per iteration is the same as the assignment phase. If the algorithm converges in t iterations, the overall time complexity is $O(tnkd)$. However, convergence is not guaranteed in all cases.

In worst-case scenarios, Lloyd's algorithm can require an exponential number of iterations. Research in literature [129] showed that K-means may take $2^{\Omega(n)}$ steps to converge, depending on the initialization and data configuration. This result highlights the sensitivity of the algorithm to initial centroid placement. To mitigate this, the K-means++ initialization method was proposed, which selects initial centroids probabilistically to improve spread and reduce the likelihood of poor local minima. K-means++ offers an $O(\log k)$-approximation to the optimal clustering and often leads to faster convergence in practice, though it does not eliminate the possibility of slow convergence entirely.

From a complexity-theoretic perspective, the K-means problem is NP-hard,

even for small values of k and d. This hardness persists under various constraints, such as binary data or fixed cluster count. Approximation algorithms and heuristics are therefore necessary in practice. Polynomial-time approximation schemes (PTAS) exist for specific cases, such as when both k and d are fixed, but these are not applicable to general high-dimensional or large-scale datasets.

To address computational challenges, several acceleration techniques have been developed. Elkan's algorithm [130] uses the triangle inequality to avoid redundant distance computations, reducing the effective cost per iteration. Data structures like KD-trees and ball trees can also speed up nearest centroid searches, though their efficiency diminishes in high dimensions.

Smoothed analysis provides a more realistic view of K-means complexity by analyzing its behavior under slight random perturbations of the input. This framework shows that the expected number of iterations is polynomial in n, explaining why K-means often performs efficiently in practice despite its worst-case exponential behavior. Space complexity is also relevant. The standard algorithm requires $O(nd + kd)$ space to store data points, centroids, and assignments. Memory-efficient variants use summary statistics or sampling to reduce this footprint.

As for quantum approaches to K-means, when comparing our approach to the other approaches in literature, we improve in terms of generalization (our approach allows unbalanced clusters) while at the same time reducing the need for connectivity between qubits at the cost of increased qubit count.

$$|Q| = K(M * R + N + M * R * R) \tag{6.20}$$

The connectivity required changes based on what the qubit represents:

$$c_\gamma = 2R - 1(\gamma\xi) + R(\gamma\gamma) + N(\gamma\eta) \tag{6.21}$$

$$c_\eta = M * R * R\xi + M * R\gamma + K\eta \tag{6.22}$$

$$c_\xi = \xi + N\eta + 7/4 \tag{6.23}$$

if we compute the number of active connection we obtain $O(RKM + NKM + MR^2KN + K^2N + NKMR^2)$ The only quadratic dependencies are found in R and K and not in N, as it happens in the other approaches. For this reason, we

can say that while the quantum approach doesn't have a rigorous time complexity study and thus cannot be effectively compared to a classical solution, when compared to other quantum approaches, our QUBO formulation improves in terms of connectivity while allowing for imbalance clusters at the cost of a higher number of qubits.

6.2 Quantum unit commitment

6.2.1 Preliminaries

The unit commitment problem is a well-known problem in the context of electrical engineering and optimization.
It is described as a family of optimization problems where the production of electric generators is coordinated to reach an objective, typically the minimization of economic cost or the maximization of revenue.
Since it's classified as a family of problems, these have many different forms [92, 95] that depend on the problem's specifics. In general terms, we can say that a UC problem has these common characteristics:

- Generator set: a set of elements that outputs energy in the system.
- Consumption forecast: A prediction of the energy network needs, typically based on models or experience.
- Market rules/laws: Sets of rules that describe which sequence of steps can be taken in practice.
- Error margins: Given the imprecise nature of the forecast, the network needs to be able to stay operative even in unpredictable scenarios.
- Time horizon: each problem works on a set time period.

Since the UC is a well-known problem, different solutions have been proposed [131, 95], and in the literature, we can also find some attempts at solving this problem with quantum computation [96, 132, 133].
Here, we will first define the specifics of the problem we want to tackle, and then

we will describe the proposed transformation in a QUBO formulation that can be solved by quantum computers.

6.2.2 The problem's abstract formulation

The overall objective of this formulation is to

$$\max \text{Profit} = \max \text{Revenue} + \text{Cost}$$

. In this expression *Revenue* includes all components that in an ordinary situation create earnings for the network, while *Cost* includes all components that in an ordinary situation imply a monetary spending for the network. This generalization in the definition of Revenue and Cost is not always satisfied (e.g., in some time-frames selling energy will lose money instead of earning it) and highly depends on the energy market. The final formulation needs to be able to manage this property. For a relevant time period t with $t \in (1, \ldots, T)$ of length Δt we define the revenue $R(t)$ as

$$R(t) = E_{sold}(t)P_{sold}(t) + E_{inc}(t)P_{inc}(t) \tag{6.24}$$

Where $E_{sold}(t)$ is the energy sold at time t, $P_{sold}(t)$ is the predicted selling price for energy unit at timestep t, $E_{inc}(t)$ is the energy considered for the computation of the national incentives, and $P_{inc}(t)$ is the unitary value of the incentives at timestep t.
The final revenue should also consider the energy that could be in storage in the final moment of the time period $t = T$. This energy is called $E_a(T)$ (a more in-depth definition will be given in subsection 6.2.2) for each battery $a \in \{1, \ldots, A\}$. To this quantity is assigned a (unitary) monetary value P_{aT}.:

$$\text{Revenue } = \sum_{t}^{T} R(t) + \sum_{a}^{A} E_a(T)P_{aT} \tag{6.25}$$

We can now define similarly the cost $C(t)$:

$$C(t) = E_{buy}(t)P_{buy}(t) - \sum_{k}^{K} cost_k E_k(t) \tag{6.26}$$

Where $E_{buy}(t)$ is the energy bought and $P_{buy}(t)$ is the buying price, both for timestep t. Furthermore, in the cost formulation, we consider the monetary spending needed for a generator $k \in \{1, \dots, K\}$ to produce energy, and we approximate it as a value that grows linearly with the amount of energy produced at time t $E_k(t)$. A more formal definition of the generators and their production can be found in subsection 6.2.2.
While computing the *Cost*, besides the sum for $C(t)$ over every t, we need to consider the value of the energy stored in the batteries at starting time $t = 0$, to which is assigned a unitary price of P_{a0}.

$$Cost = \sum_{t}^{T} C(t) - \sum_{a}^{A} E_a(0) P_{a0} \tag{6.27}$$

For the remainder of this section, it is important to highlight the sign convention we will use. Supposing that every price is expressed with a positive number, then we consider that energies that are associated with a monetary loss (E_{buy}, E_{cons}) are represented with negative numbers, while energies that are associated with an increase in profit will be positive.
This convention will be used for this section, but in section 6.2.4, given the binary representation of the variables, we will discuss another convention that suits the formulation.

Sharing energy incentives

Given the Italian national incentives named *Shared energy incentive*, we need to formalize the energy that is considered for this part of the profit.
The energy $E_{inc}(t)$ is computed as the minimum between the modules of $E_{prod}(t)$ and $E_{cons}(t)$:

$$E_{inc}(t) = \min(|E_{prod}(t)|, |E_{cons}(t)|) \tag{6.28}$$

Conservation of energy constraint

Given the introduction of the energy sold and bought, we need to balance them and specify where this energy comes from and where it goes. For this reason, we

introduce the *Conservation of energy constraint*:

$$E_{prod}(t) - E_{buy}(t) - E_{sold}(t) + E_{cons}(t) = 0 \tag{6.29}$$

Where E_{prod} is the energy produced and E_{cons} is the energy consumed.
The dissipation of energy is not considered in this equation since the formulation of batteries already takes into account the efficiency, and any other loss in the network can be assimilated into the E_{cons} quantity.

Energy produced

The formula to compute the quantity E_{prod} is as follows:

$$E_{prod}(t) = E_{pf}(t) + \sum_{k}^{K} E_k(t) + \sum_{a}^{A} E_a^s(t) \tag{6.30}$$

where $E_{pf}(t)$ is the electricity produced at time t by sources that cannot be controlled and are deemed fixed (fixed production energy).
In the formula, the first sum contains all the electricity produced by the K generators, while the second represents the energy released from the A batteries.

Energy consumed

The formula to compute the energy consumed E_{cons} is the following:

$$E_{cons}(t) = E_{cf}(t) + \sum_{a}^{A} E_a^c(t) \tag{6.31}$$

Where E_{cf} is the electricity consumed at time t by sources that cannot be controlled and are deemed as fixed (fixed consumption), and $E_a^c(t)$ is the electricity that is charged into the batteries at time t. From the sign conventions, it follows that both variables in the formula represent a negative quantity since $E_{cons}(t)$ is negative.

Battery constraints

We already introduced the concept of energy stored in a battery $E_a(t)$ for each battery $a \in \{1, \ldots, A\}$ and the variables describing the energy charged ($E_a^c(t)$) and discharged ($E_a^s(t)$) from them at time t. Let's now add a constraint that tells us that a battery cannot be charging and discharging at the same time:

$$E_a^s(t)E_a^c(t) = 0 \tag{6.32}$$

Thanks to this equation, at least one of the two variables needs to be equal to 0 at any time t for each battery a. Let's now introduce some constraints on the maximum and minimum:

$$\forall t, \forall a \Rightarrow E_a(t-1) - \frac{1}{\eta_a^s} E_a^s(t) > 0 \tag{6.33}$$

$$E_a(t-1) - \eta_a^c E_a^c(t) < E_a^{\max} \tag{6.34}$$

Where $E_a^{\max}$ is the maximum amount of energy that can be stored in a battery. We can describe the evolution of the energy inside the batteries through the following formula:

$$E_a(t) = E_a(t-1) - \eta_a^c E_a^c(t) - \frac{1}{\eta_a^s} E_a^s(t) \tag{6.35}$$

This recursive definition has a basis case in $E_a(0)$ that needs to be specified as input. In this formula, the values η_a^c and η_a^s represent respectively the charging and discharging efficiency of the battery a.

Energy Bought and sold

Two other quantities that we should define are the amount of electricity sold $E_{sold}(t)$ and bought $E_{buy}(t)$. The simplest solution is to utilize these two as variables that need to be balanced during the optimization. We do this simply by adding a constraint telling us that for a given timestep t, we cannot buy and sell at the same time:

$$\forall t, \Rightarrow E_{sold} \cdot E_{buy} = 0 \tag{6.36}$$

This creates a mutually exclusive relationship between the two, where at least one is forced to be 0 at any given time.
The computation of the quantities represented by these variables is directly connected to the conservation of energy constraint.

Maximum and minimum constraints

Now that we have formalized different variables and constraints, we need to restrain the first ones in a predetermined range. Let's consider the energy produced by a generator k at time t $E_k(t)$, the amount of energy charged or discharged from a battery a at time t and the electricity contained in them $E_a^c(t)$, $E_a^s(t)$, $E_a(t)$:

$$0 \leq E_k(t) \leq E_k^{\max}(t) \tag{6.37}$$

$$0 \leq E_k(t) \leq E_{nom_k} \tag{6.38}$$

$$E_a^{\max c} \leq E_a^c(t) \leq 0 \tag{6.39}$$

$$0 \leq E_a^s(t) \leq E_a^{\max s} \tag{6.40}$$

$$0 \leq E_a(t) \leq E_a^{\max} \tag{6.41}$$

Here $E_k^{\max}(t)$ is the maximum energy a generator can produce at time t while E_{nom_k} is the maximum energy that the generator can produce in the best-case scenario. These two different interpretations of 'maximum generated energy' will be very useful later on when we have to distinguish between green energy production and traditional power generators.
$E_a^{\max c}$, $E_a^{\max s}$ are the maximum values for the charging and discharging of batteries, while $E_a^{\max}$ is the maximum quantity of electric energy a battery can hold.

6.2.3 Summary table

Symbol	Meaning	D.	O.	S.	M.
$E_{sold}(t)$	Energy sold	t	D	+	MWh
$P_{sold}(t)$	Unitary selling price	t	I		€/MWh
$E_{buy}(t)$	Energy bought	t	D	-	MWh
$P_{buy}(t)$	Unitary buying price	t	I		€/MWh
$E_{inc}(t)$	Amount of energy the incentive are computed on	t	D	+	MWh
$P_{inc}(t)$	Incentives unitary value	t	I		€/MWh
$E_{prod}(t)$	Energy produced (total)	t	D	+	MWh
$E_{cons}(t)$	Energy consumed	t	D	-	MWh
P_{a0}	Unitary value of stored energy at time t=0	-	I		€/MWh
P_{aT}	Unitary value of stored energy at time t=T	-	I		€/MWh
$E_k(t)$	Energy produced by a generator	t, g	O	+	MWh
$cost_k$	Generator production cost	g	I		€/MWh
$E_{cf}(t)$	Fixed consumptions	t	I	-	MWh
$E_a(t)$	Energy contained in a battery	t, a	D	+	MWh
$E_a(0)$	Energy contained in a battery at t=0	a	I	+	MWh
$E_a^s(t)$	Energy discharged from a battery	t, a	O	+	MWh
$E_a^c(t)$	Energy charged into a battery	t, a	O	-	MWh
$E_a^{\max}$	Maximum capacity of a battery	a	I	+	MWh
η_a^c	Efficiency of a charging battery	a	I		
η_a^s	Efficiency of a discharging battery	a	I		
$E_a^{\max c}$	Maximum energy chargeable from a battery	a	I	-	MWh
$E_a^{\max s}$	Maximum energy dischargeable from a battery (•_•)	a	I	+	MWh
E_{nom_k}	Best case maximum energy produced by a generator	g	I	+	MWh
$E_k^{\max}(t)$	Maximum predicted energy produced by a generator at time t	g,t	I	+	MWh

Table 6.1: Summary table for all symbols and their meanings. Column *D.* (Dependencies) identifies the dependency of each variable between (t time, g generators, a batteries), column *O.* (origin) tells us where these quantities come from (D derived from other quantities, I inputs, O outputs), column *S.* (Sign) tells us the sign of the quantities based on the conventions we described above. Column *M.* (unit of measurement) tells us the physical quantity to which the variables refer (MWh MegaWatt hour, €/MWh Dollars per Megawatt hour)

Complete formula

Since we now have all the definitions for the abstract formulation, let's put everything together

$$\begin{aligned}\max \text{Profit} =& \text{Revenue} + \text{Cost} \\ =& \sum_{t}^{T} \left(E_{sold}(t) P_{sold}(t) + E_{inc}(t) P_{inc}(t) + E_{buy}(t) P_{buy}(t) - \sum_{k}^{K} cost_k E_k(t) \right) \\ &+ \sum_{a}^{A} \left(E_a(T) P_{aT} - E_a(0) P_{a0} \right)\end{aligned}$$

If we also include the constraint, we can see the full formulation of the problem:

$$\begin{aligned}\max \text{Profit} =& \text{Revenue} + \text{Cost} \\ =& \sum_{t}^{T} \left(E_{sold}(t) P_{sold}(t) + E_{inc}(t) P_{inc}(t) + E_{buy}(t) P_{buy}(t) - \sum_{k}^{K} cost_k E_k(t) \right) \\ &+ \sum_{a}^{A} \left(E_a(T) P_{aT} - E_a(0) P_{a0} \right) \\ \forall t, a, k : \; & E_a^c(t) E_a^s(t) = 0 \\ & E_a(t-1) - \frac{1}{\eta_a^s} E_a^s(t) > 0 \\ & E_a(t-1) - \eta_a^c E_a^c(t) < E_a^{\max} < E_a^{\max} \\ & E_{prod}(t) - E_{buy}(t) - E_{sold}(t) + E_{cons}(t) = 0 \\ & 0 \leq E_k(t) \leq E_k^{\max}(t) \\ & 0 \leq E_k(t) \leq E_{nom_k} \\ & 0 \geq E_a^c(t) \geq E_a^{\max c} \\ & 0 \leq E_a^s(t) \leq E_a^{\max s} \\ & 0 \leq E_a(t) \leq E_a^{\max}\end{aligned}$$

Where the following variables are derived as:

$$E_{inc}(t) = \min(|E_{prod}(t)|, |E_{cons}(t)|)$$

The energy used to compute the incentives is the minimum between the energy produced and consumed.

$$E_{prod}(t) = E_{pf}(t) + \sum_{k}^{K} E_k(t) + \sum_{a}^{A} E_a^s(t)$$

The energy produced is the sum of the energy discharged from the batteries and generated from the generators (both fixed and flexible production).

$$E_{cons}(t) = E_{cf}(t) + \sum_{a}^{A} E_a^c(t)$$

The energy consumed is the sum of the energy charged into the batteries and the fided consumptions.

$$E_a(t) = E_a(t-1) - \eta_a^c E_a^c(t) - \frac{1}{\eta_a^s} E_a^s(t)$$

The energy stored in the batteries at time t is computed as a function of charge and discharge as well as the starting state.

6.2.4 QUBO formulation of the problem

Target variables

We have three main target variables; these are the ones that we are going to obtain in the output. Specifically we are referring to $E_k(t)$, $E_a^s(t)$ and $E_a^c(t)$.
Let's now reformulate these continuous variables as the sum of binary ones by means of binarization. Here we introduce the term g_i to regulate the approximation of the binarization:

$$E_k(t) = E_k^{min} + \sum_{p=0}^{P_k(t)} 2^{p+g_1} x_{k,p}(t) \tag{6.42}$$

$$E_a^s(t) = \sum_{b=0}^{B_a} 2^{b+g_2} x_{a,b}^s(t)$$

$$E_a^c(t) = -\sum_{b=0}^{B_a} 2^{b+g_2} x_{a,b}^c(t) \tag{6.43}$$

Where x represents a binary variable ($x \in \{0, 1\}$) while $P_k(t)$ and B_a tell us the number of binary variables needed to transform the continuous one:

$$P_k(t) = \lfloor \log_2(E_k^{max}(t) - E_k^{min}) \rfloor - g_1 \tag{6.44}$$

$$B_a = \lfloor \log_2(E_a^{\max c}) \rfloor - g_2 \tag{6.45}$$

By changing the value of g_i, we are able to lower or increase the approximation of the transformation.
Now we can notice that while we considered a generic continuous variable in the previous section, we can't use negative numbers. For this reason, from now on, we will explicitly represent a negative value by changing the sign in front.
It is worth restate the meaning of the variables t, k, and a:

- $t \in (1, 2, \ldots, T)$ tells us the current temporal period between the total T.
- $k \in \{1, 2, \ldots, K\}$ represents one of the K generators.
- $a \in \{1, 2, \ldots, A\}$ represents one of the A batteries.

This formulation requires us to add an explicit constraint for the maximum values that can be represented. We will refer to this as the maximum constraint.

$$\forall k, T \rightarrow \sum_{p=0}^{P_k(t)} 2^{p+g_1} x_{k,p}(t) + E_k^{\min} < E_k^{\max}(t)$$

$$\forall a, T \rightarrow \sum_{b=0}^{B_a} 2^{b+g_2} x_{a,b}^c(t) < E_a^{\max c}$$

$$\forall a, T \rightarrow \sum_{b=0}^{B_a} 2^{b+g_2} x_{a,b}^s(t) < E_s^{\max c}$$

Since QUBO formulation doesn't allow for external constraint, we need to transform this into a penalty term by adding a support variable $\Psi_k(t)$ to balance the inequality. Considering the maximum constraint for generators, we can use a simple squaring to obtain a penalty term:

$$\sum_{t}^{T}\sum_{k}^{K}\left(\sum_{p=0}^{P_k(t)} 2^{p+g_1} x_{k,p}(t) + E_k^{\delta}(t) + \Psi_k(t)\right)^2 \tag{6.46}$$

Where $E_k^{\delta}(t) = E_k^{\min} - E_k^{\max}(t)$ and $\Psi_k(t)$ has the form:

$$\Psi_k(t) = \sum_{p=0}^{P_k(t)} 2^{p+g_1}\psi_{k,p}(t)$$

A similar operation can be applied to the other maximum constraints:

$$\sum_{t}^{T}\sum_{k}^{K}\left(\sum_{b=0}^{B_a} 2^{b+g_2} x_{a,b}^{c}(t) - E_a^{\max\, c} + \Psi_a^{c}(t)\right)^2 \tag{6.47}$$

$$\sum_{t}^{T}\sum_{k}^{K}\left(\sum_{b=0}^{B_a} 2^{b+g_2} x_{a,b}^{s}(t) - E_a^{\max\, s} + \Psi_a^{s}(t)\right)^2 \tag{6.48}$$

In this two penalty terms we utilized the support variables $\Psi_a^c(t)$ and $\Psi_a^s(t)$, defined as:

$$\Psi_{a,b_2}^{c}(t) = \sum_{b=0}^{B_a} 2^{b+g_2}\psi_{a,b}^{c}(t) \tag{6.49}$$

$$\Psi_{a,b_2}^{s}(t) = \sum_{b=0}^{B_a} 2^{b+g_2}\psi_{a,b}^{s}(t) \tag{6.50}$$

Batteries

Let's now consider the variables and the constraints related to batteries.
The first constraint:

$$\forall a, t \in A, T \rightarrow E_a^c(t) \cdot E_a^s(t) = 0$$

easily becomes a penalty term by multiplying the two quantities:

$$\rho_{cs} \sum_{t}^{T} \sum_{a}^{A} \sum_{b_1}^{B_a} \sum_{b_2}^{B_a} x^c_{a,b_1}(t) x^s_{a,b_2}(t) \tag{6.51}$$

Here we also added a hyperparameter ρ_{cs} representing a penalty coefficient for this term. Now let's look at the quantity of energy in a battery $E_a(t)$ as presented before:

$$E_a(t) = E_a(0) + \sum_{\tau}^{t} \sum_{b}^{B_a} \left(\eta^c_a 2^{b+g_2} x^c_{a,b}(\tau) - \frac{1}{\eta^s_a} 2^{b+g_2} x^s_{a,b}(\tau) \right) \tag{6.52}$$

This allows us to compute E_a for each timestep in a recursive manner.
From this definition, we need to add a penalty term that doesn't allow the maximum $E_a(t)$ to be either above the maximum energy storable or below 0. We will use a support variable Ξ^s_a to remove the inequality:

$$E_a(0) + \sum_{\tau}^{t-1} \sum_{b}^{B_a} \left(\eta^c_a 2^{b+g_2} x^c_{a,b}(\tau) - \frac{1}{\eta^s_a} 2^{b+g_2} x^s_{a,b}(\tau) \right) - \sum_{b}^{B_a} \frac{1}{\eta^s_a} 2^{b+g_2} x^s_{a,b}(t) - \Xi^s_a(t) = 0$$

To improve readability let's use the recursive definition of $E_a(t)$:

$$E_a(t-1) - \sum_{b}^{B_a} \frac{1}{\eta^s_a} 2^{b+g_2} x^s_{a,b}(t) - \Xi^s_a(t) = 0 \tag{6.53}$$

Let's now do the same for the charging using the support variable Ξ^c_a:

$$E_a(t-1) + \sum_{b}^{B} \eta^c_a 2^{b+g_2} x^c_{a,b}(t) - E^{\max}_a + \Xi^c_a(t) = 0 \tag{6.54}$$

We need to binarize the supporting variables with values between 0 and $E^{\max}_a$:

$$\Xi^c_a(t) = \sum_{q=0}^{Q_a} 2^{q+g_3} \xi^c_{a,q}(t) \tag{6.55}$$

$$\Xi_a^s(t) = \sum_{q=0}^{Q_a} 2^{q+g_3} \xi_{a,q}^s(t) \tag{6.56}$$

Where Q_a is obtained as:

$$Q_a = \lfloor \log_2(E_a^{\max}) \rfloor - g_3 \tag{6.57}$$

Notice how the approximation of the variables has to be the same as the variables $E_a^c(t)$ and $E_a^s(t)$. For this reason, we specify that

$$g_3 = g_2$$

Now to finish the transformation, we add the hyperparameters ρ_{as} for the discharge formula and ρ_{ac} for the charge one, and we square everything so that the penalty always has the same sign when violated:

$$\rho_{as} \sum_t^T \sum_a^A \left(E_a(t-1) - \sum_b^{B_a} \frac{1}{\eta_a^s} 2^{b+g_2} x_{a,b}^s(t) - \sum_q^{Q_a} 2^{q+g_3} \xi_{a,q}^s(t) \right)^2 \tag{6.58}$$

$$\rho_{ac} \sum_t^T \sum_a^A \left(E_a(t-1) + \sum_b^{B_a} \eta_a^c 2^{b+g_2} x_{a,b}^c(t) - E_a^{\max} + \sum_q^{Q_a} 2^{q+g_3} \xi_{a,q}^c(t) \right)^2 \tag{6.59}$$

Bought and sold energy

The quantities of energy sold and bought are added to the target variables, and we start by binarizing them:

$$E_{sold}(t) = \sum_{r=0}^{R} 2^{r+g_4} x_r^{sold}(t) \tag{6.60}$$

$$E_{buy}(t) = \sum_{r=0}^{R} 2^{r+g_4} x_r^{buy}(t) \tag{6.61}$$

both of them will have the same range $[0; 2^R]$ with step 2^{g_4} where R is computed as:

$$R = \left\lfloor \log \left(\sum_{k}^{K} E_k^{\max} + \sum_{a}^{A} E_a^{\max c} \right) \right\rfloor - g_4 \tag{6.62}$$

This formula takes into consideration the limit case where all the producible energy is produced and has to be sold. We consider the opposite case (where all the energy has to be bought) to be smaller than this quantity, and a more precise definition is left for future work.
Moving on to the constraint, we first define the two variables as mutually exclusive:

$$\rho_{va} \sum_{t}^{T} \sum_{r_1}^{R} \sum_{r_2}^{R} x_{r_1}^{sold}(t) x_{r_2}^{buy}(t) \tag{6.63}$$

Where ρ_{va} is a penalty coefficient that we need to fine-tune to the specific problem. The power of two has been removed from this formula without loss of generality. We can now redefine the conservation of energy constraint into a penalty:

$$\forall t \in T : \; E_{prod} + E_{buy} - E_{sold} - E_{cons} = 0 \tag{6.64}$$

$$\begin{aligned} &\sum_{t}^{T} \left(E_{pf}(t) + \sum_{k}^{K} \left(E_k^{min} + \sum_{p}^{P_k} 2^{p+g_1} x_{k,p}(t) \right) + \right. \\ &\quad \sum_{a}^{A} \sum_{b}^{B_a} 2^{b+g_2} x_{a,b}^{s}(t) + \\ &\quad \sum_{r}^{R} 2^{r+g_4} x_r^{buy}(t) - \sum_{r}^{R} 2^{r+g_4} x_r^{sold}(t) - \\ &\quad \left. E_{cf}(t) - \sum_{a}^{A} \sum_{b}^{B_a} 2^{b+g_2} x_{a,b}^{c}(t) \right) \end{aligned} \tag{6.65}$$

To conclude, let's add the penalty coefficient and square everything (so, to improve readability, we reported the original name of the variable as a placeholder for the

binarized term):

$$\rho_{e0} \sum_{t}^{T} \left(E_{pf}(t) + \sum_{k}^{K} E_k(t) + \sum_{a}^{A} (E_a^s(t) - E_a^c(t)) + E_{buy}(t) - E_{sold}(t) - E_{cf}(t) \right)^2 \tag{6.66}$$

Incentive

We can transform the formula for $E_{inc}(t)$ simply through a binary variable $\xi^{inc}(t)$. This variable forces the model to choose the minimum between $E_{prod}(t)$ and $E_{cons}(t)$ using the penalty term:

$$\rho_{inc} \sum_{t}^{T} \left(\xi^{inc}(t) E_{prod}(t) + (1 - \xi^{inc}(t)) E_{cons}(t) \right) = \tag{6.67}$$

and substituting

$$E_{inc}(t) = \xi^{inc}(t) E_{prod}(t) + (1 - \xi^{inc}(t)) E_{cons}(t)$$

Objective function

The objective function

$$\begin{aligned} \max Profit = \sum_{t}^{T} & (E_{sold}(t) P_{sold}(t) + E_{inc}(t) P_{inc} \\ & - E_{buy}(t) P_{buy}(t) - \sum_{k}^{K} cost_k E_k(t) \Bigg) + \\ & \sum_{a}^{A} (E_a(T) P_{aT} - E_a(0) P_{a0}) \end{aligned} \tag{6.68}$$

It can be easily transformed into a QUBO formulation by substituting the variable we saw before, without the need to square the formula. Finally, to achieve the complete QUBO formulation, we just need to add the penalty terms discussed

above (6.51,6.66,6.63,6.58).

$$
\begin{aligned}
\max Profit = & \sum_{t}^{T}\left(E_{sold}(t)P_{sold}(t)+E_{inc}(t)P_{inc}-E_{buy}(t)P_{buy}(t)-\sum_{k}^{K}cost_{k}E_{k}(t)\right)\\
& +\sum_{a}^{A}\left(E_{a}(T)P_{aT}-E_{a}(0)P_{a0}\right)\\
& -\rho_{cs}\sum_{t}^{T}\sum_{a}^{A}\sum_{b_1}^{B_a}\sum_{b_2}^{B_a}x^{c}_{a,b_1}(t)x^{s}_{a,b_2}(t)\\
& -\rho_{as}\sum_{t}^{T}\sum_{a}^{A}\left(E_{a}(t-1)-\frac{1}{\eta^{s}_{a}}E^{s}_{a}(t)-\Xi^{s}_{a}(t)\right)^{2}\\
& -\rho_{ac}\sum_{t}^{T}\sum_{a}^{A}\left(E_{a}(t-1)+\eta^{c}_{a}E^{c}_{a}(t)-E^{\max}_{a}+\Xi^{s}_{a}(t)\right)^{2}\\
& -\rho_{va}\sum_{t}^{T}\sum_{r_1}^{R}\sum_{r_2}^{R}x^{sold}_{r_1}(t)x^{buy}_{r_2}(t)\\
& -\rho_{e0}\sum_{t}^{T}\left(E_{pf}(t)+\sum_{k}^{K}E_{k}(t)+\sum_{a}^{A}\left(E^{s}_{a}(t)-E^{c}_{a}(t)\right)+E_{buy}(t)-E_{sold}(t)-E_{cf}(t)\right)^{2}\\
& -\rho_{inc}\sum_{t}^{T}\left(\xi^{inc}(t)E_{prod}(t)+(1-\xi^{inc}(t))E_{cons}(t)\right)\\
& +\rho_{mx}\sum_{t}^{T}\sum_{k}^{K}\left(\sum_{p=0}^{P_k(t)}2^{p+g_1}x_{k,p}(t)+E^{\delta}_{k}(t)+\psi_{k}(t)\right)^{2}\\
& +\rho_{mx}\sum_{t}^{T}\sum_{a}^{A}\left(\sum_{p=0}^{B_a}2^{p+g_2}x^{s}_{a,b}(t)+E^{\max s}_{a}+\psi^{s}_{a,b}(t)\right)^{2}\\
& +\rho_{mx}\sum_{t}^{T}\sum_{a}^{A}\left(\sum_{p=0}^{B_a}2^{p+g_2}x^{c}_{a,b}(t)+E^{\max s}_{a}+\psi^{c}_{a,b}(t)\right)^{2}
\end{aligned}
\tag{6.69}
$$

The formula above didn't include the binarization of the variables and the expansion of the squared terms for readability reasons, but a simple substitution of the binarized terms and some basic mathematical steps allows for a complete and expanded formula.

6.2.5 Complexity

One very important aspect in solving a QUBO problem is the number of variables that the problem has. In our case, the total can be obtained by summing up all the relative quantities:

- $x_k^p(t)$: variable that encodes in P elements the energy generated by k at time t (total of: $T \cdot \sum_k^K P_k$)

- $\psi_{k,p}(t), \psi_{a,b}^c(t), \psi_{a,b}^s(t)$: supporting variables used to transform the maximum constraints at time t (total of: $T \cdot (2\sum_a^A B_a + \sum_k^K P_k)$)

- $x_{a,b}^c(t), x_{a,b}^s(t)$: variable that encode in B_a elements the charging and discharging of a at time t (total of: $2 \cdot T \cdot \sum_a^A B_a$)

- $\xi_{a,q}^s(t), \xi_{a,q}^c(t)$: supporting variable that encode in Q_a elements some of the coefficient for the charging and discharging constraints of each battery a at time t (total of: $2 \cdot T \cdot A \cdot \sum_a^A Q_a$)

- $x_r^{sold}(t), x_r^{buy}(t)$: supporting variables that encode in R elements the energy sold and bought at time t (total of: $2 \cdot T \cdot R$)

- $\xi^{inc}(t)$: supporting variable used in computing the incentives (total of: T)

The total number of variables is:

$$T \cdot (2 \cdot \sum_k^K P_k + 4 \cdot \sum_a^A B_a + 2 \cdot A \cdot \sum_a^A Q_a + 2 \cdot R + 1) \tag{6.70}$$

6.3 Hybrid technologies QSVM

6.3.1 Preliminaries

An SVM is a supervised machine learning algorithm designed for both classification and regression tasks. It operates on a dataset $D\{(x_n, t_n) : n = 0, \ldots, N-1\}$, where $x_n \in \mathbb{R}^d$

represents a point in d-dimensional space, serving as a feature vector, and t_n denotes the target label assigned to x_n. We will focus on the classification task and learning a binary classifier that assigns a class label $\hat{t}_n = \pm 1$ to a given data point x_n. For clarity, we designate the class $t_n = 1$ as 'positive' and the class $t_n = -1$ as 'negative'.

The training of an SVM entails solving the quadratic programming (QP) problem:

$$minimize \;\; E = \frac{1}{2} \sum_{nm} \alpha_n \alpha_m t_n t_m k(x_n, x_m) - \sum_n \alpha_n \tag{6.71}$$

with

$$0 \leq \alpha_n \leq C \quad and \quad \sum_n \alpha_n t_n = 0 \tag{6.72}$$

For a set of N coefficients $\alpha_n \in \mathbb{R}$, where C denotes a regularization parameter and $k(\cdot, \cdot)$ represents the kernel function of the Support Vector Machine [134, 135], the resulting coefficients α_n establish a $(d-1)$-dimensional decision boundary that partitions $\mathbb{R}^d$ into two regions corresponding to the predicted class label. The decision boundary is defined by points associated with $\alpha_n \neq 0$, commonly referred to as the support vectors of the SVM. Prediction for an arbitrary point $x \in \mathbb{R}^d$ can be accomplished by

$$f(x) = \sum_n \alpha_n t_n k(x_n, x) + b \tag{6.73}$$

where b can be estimated by formula [135]:

$$b = \frac{\sum_n \alpha_n (C - \alpha_n)[t_n - \sum_m \alpha_m t_m k(x_m, x_n)]}{\sum_n \alpha_n (C - \alpha_n)} \tag{6.74}$$

Geometrically, the decision function $f(x)$ corresponds to the signed distance between the point x and the decision boundary. Consequently, the predicted class label $\hat{t}$ for x as determined by the trained Support Vector Machine is given by $\hat{t} = \text{sign}(f(x))$.

The problem formulation can be equivalently expressed as a convex quadratic optimization problem [48], indicating its classification among the rare minimization problems in machine learning that possess a globally optimal solution. It is crucial to note that while the optimal solution exists, it is dataset-specific and may not necessarily generalize optimally across the entire data distribution.

Kernel-based SVMs exhibit exceptional versatility as they can obtain nonlinear decision boundaries denoted by $f(x) = 0$. This is achieved through the implicit mapping

of feature vectors into higher-dimensional spaces [136]. Importantly, the computational complexity does not escalate with this higher dimensionality, as only the values of the kernel function $k(x_n, x_m)$ are involved in the problem specification. This widely recognized technique is commonly referred to as the "kernel trick" and has been extensively explored in the literature [134, 135].

The selection of the kernel function significantly influences the outcomes, with radial basis function kernels (RBF) [48] generally serving as the search starting point for the right kernel of a Support Vector Machine problem. An RBF kernel is distinguished by the property that $k(x_n, x_m)$ can be expressed as a function of the distance $\|x_n - x_m\|$ [134]. The Gaussian kernel, often referred as "The RBF kernel", is the most prevalent RBF kernel and is represented as

$$rbf(x_n, x_m) = e^{-\eta||x_n - x_m||^2} \tag{6.75}$$

Here, the value of the hyperparameter $\eta > 0$ is typically determined through a calibration procedure before the training phase [137].

The SVM are very susceptible to the choice of the hyper-parameters, like η or C, and different assignments can radically change the result of the optimization.

Annealer-Based Quantum Support Vector Machines

Quantum computers leverage diverse hardware approaches and technologies, with quantum annealing being one notable paradigm.

Quantum annealing commences by establishing a quantum-mechanical superposition of all possible states, followed by the system's evolution governed by the time-dependent Schrödinger equation. The amplitudes of all states undergo continuous changes and, if the rate of change is sufficiently slow, the system remains in the ground state of the instantaneous Hamiltonian, characterizing adiabatic quantum computation [138]. In the case of insufficiently slow changes, the system may leave the ground state temporarily, while at the same time producing a higher likelihood of concluding in the ground state of the final problem Hamiltonian in adiabatic quantum computation [139, 140]. Quantum annealers efficiently solve problems formulated in either Quadratic Unconstrained Binary Optimization (QUBO) or Ising formulations.

Since we know how to formulate Support Vector Machines as convex quadratic optimization problems, a simple transformation is enough to achieve a QUBO formulation suitable for quantum annealing.

In a study by Willsch et al. [81], the authors introduced and explored the application of kernel-based Support Vector Machines on a DW2000Q quantum annealer [82]. From here on out, we refer to this methodology as Quantum Annealer Support Vector Machines (QaSVM). This approach offers distinct mathematical advantages, generating a spectrum of different classifiers with each iteration of the training process.

The study's findings demonstrate that the ensemble of classifiers produced by the quantum annealer can surpass the single classifier derived from classical SVMs when addressing the same computational problem. Performance is assessed through metrics such as Area Under the Receiver Operating Characteristic curve (AUROC), Area Under the Precision–Recall curve (AUPRC) [141, 83], and accuracy. This advantage is attributed to the quantum annealer's ability to yield not only the global optimum for the training dataset but also a distribution of solutions close to optimality for the given optimization problem. The potential to combine these solutions enhances generalization to the test dataset.

Additional studies [84, 85] corroborate the efficacy of QaSVMs and extend their promise to diverse problems, including multiclass classification.

Gate-Based Quantum Support Vector Machines

Gate-based quantum computers operate within the quantum circuit model, where quantum computation involves initialization of qubits to known values, the application of a sequence of quantum gates, and measurements. While this approach is broader in scope than the one outlined in Section 6.3.1, it presents formidable challenges in both hardware and algorithm development.

Quantum machine learning is predominantly studied on gate-based quantum computers, meaning there is more research on the QSVM model. THe quantum machine learning (QML) models exhibit diverse architectures and ansatz configurations for processing data. Even so, most models share a common initial step wherein classical data undergoes encoding in the Hilbert space, known as feature mapping. Schuld and Killoran [142] showed the equivalence of this phase to a quantum kernel and proposed one of the initial gate-based Support Vector Machine models (QgSVM). Their work outlines two approaches: firstly, they introduce the quantum kernel as a means to process classical data, transforming it into new data with different dimensionality analogous to a classical kernel, which is then utilized in classical algorithms such as SVM. Secondly, they propose a parametric circuit for classifying input in a quantum environment. While

this feature map–ansatz approach has become standard for Quantum Neural Networks (QNNs), a detailed discussion exceeds the scope of this document.

Research by Maria Schuld [80] aggregates results from various sources and concludes that all supervised quantum machine learning models (excluding generative ones) operate as kernel methods.

Another noteworthy contribution to the state of the art in gate-based QSVMs is the work by Havlicek et al. [33]. They introduce two SVM classifiers optimizing classically over data obtained using a quantum kernel trick to achieve quantum advantage. Similar to previous approaches, this involves non-linearly mapping data to a quantum state:

$$\phi : \tilde{x} \in \Omega \rightarrow |\phi(\tilde{x})\rangle \langle\phi(\tilde{x})| \tag{6.76}$$

The authors also highlight a crucial requirement for applying QgSVM: if the feature vector kernel $K(\tilde{x}, \tilde{z}) = |\langle\Phi(\vec{x})|\Phi(\vec{z})\rangle|^2$ is overly simplistic, such as generating only product states, it can be easily computed classically and loses its benefit. The advantage lies in leveraging the high dimensionality of the Hilbert space, necessitating a kernel that is impossible to simulate to surpass classical approaches.

6.3.2 Materials and Methods

In the outlined quantum computing landscape, we introduced two distinct architectures for employing Support Vector Machines in classification tasks. Both models, still in their early stages, warrant further thorough investigation and testing.

Quantum Annealer Support Vector Machines (QaSVM) demonstrate partial efficacy in addressing the problem, yet encounter challenges in precisely determining the optimal hyperplane. Additionally, QaSVM relies on a classical kernel trick to compute the $k(x_n, x_m)$ component of the formulation.

Conversely, Quantum gate-based Support Vector Machines (QgSVM) exhibit impressive data manipulation capabilities. However, they currently lack the readiness to handle the optimization aspect of the SVM algorithm. Present-day quantum technology does not possess the computational power necessary for intricate optimization problems. Consequently, the optimization step necessitates an approximate approach, achieved either through an ansatz (transitioning from QgSVM to a Quantum Neural Network, QNN model) or by resorting to classical optimization methods.

Our proposed approach involves a fusion of the quantum annealing and gate-based models, establishing a connection through a classical channel. The core idea is to cap-

italize on the strengths of each approach: annealing excels in optimization but lacks a dedicated kernel method, while the gate-based model performs well with the kernel trick but faces challenges in optimization.

Building upon our previous discussions, we recognize the ability to derive the Kernel Matrix from a quantum feature map within the gate-based architecture. Once the Kernel matrix is defined, we employ a standard procedure to reformulate the problem into a Quadratic Unconstrained Binary Optimization (QUBO) format, achieved by discretizing the continuous variables of a quantum computing problem. Subsequently, the QUBO formulation is encoded into the annealer, enabling the extraction of minima, as previously described. The illustrated process, as depicted in Figure 6.1, allows the attainment of results for even challenging problems, leveraging the combined advantages of both quantum technologies.

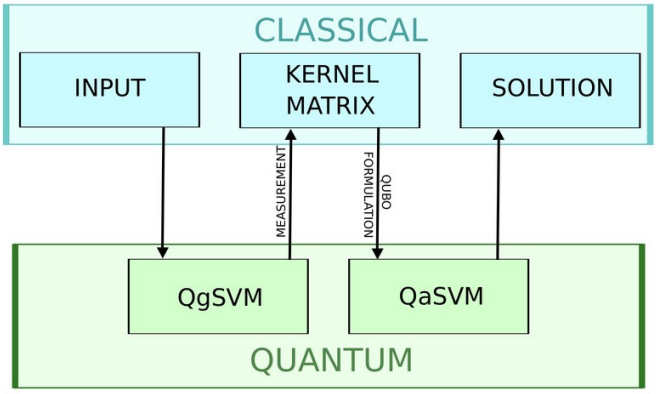


Figure 6.1: The image represents a detailed schematic outlining our proposed methodology for computing the Quantum Support Vector Machine (QSVM). The process initiates with the encoding of input data into the Quantum gate-based Support Vector Machine (QgSVM), which is responsible for the computation of the Kernel matrix. Following this initial stage, the Quantum annealer Support Vector Machine (QaSVM) encodes the problem's Quadratic Unconstrained Binary Optimization (QUBO) formulation. The subsequent solution extraction occurs through the quantum annealing process within the QaSVM framework. In the image the arrow show the passages from classical to quantum and vice-versa. QgSVM and QaSVM are connected through the classical Kernel matrix by the measurement operation and the QUBO formulation.

From Classical SVM to QaSVM

We proceed to illustrate the translation of the Support Vector Machine into a form suitable for solving with Quantum Annealers. The complete formulation is detailed in the work by Willsch et al. [81].

The initial challenge in this transformation arises from the fact that, by definition, $\alpha_n \in \mathbb{R}$ while quantum annealers are only capable of producing discrete binary values. A straightforward resolution to this challenge involves binarizing the values of α as follows:

$$\alpha = \sum_{k}^{K-1} B^k a_{Kn+k} \tag{6.77}$$

where $a_{Kn+k} \in \{0, 1\}$, B represents the chosen basis and K denotes the number of variables a_k used for representing α. We proceed by substituting α with a and utilizing Equation (6.77) as a constraint, multiplied by a factor ξ:

$$\begin{aligned} E =& \frac{1}{2} \sum_{nmkj} a_{Kn+k} a_{Km+j} B^{k+j} t_n t_m k(x_n, x_m) \\ & - \sum_{nk} B^k a_{Kn+k} + \xi \left(\sum_{nk} B^k a_{Kn+k} t_n \right)^2 \qquad (6.78) \\ =& \sum_{n,m=0}^{N-1} \sum_{k,j=0}^{K-1} a_{Kn+k} \tilde{Q}_{Kn+k,Km+j} a_{Km+j} \qquad (6.79) \end{aligned}$$

where $\tilde{Q}$ is a matrix of $KN \times KN$ given by

$$\tilde{Q}_{Kn+k,Km+j} = \frac{1}{2} B^{k+j} t_n t_m (k(x_n, x_m) + \xi) - \delta_{nm} \delta_{kj} B^k \tag{6.80}$$

Naturally, given that $\tilde{Q}$ is symmetric, the upper triangular Quadratic Unconstrained Binary Optimization (QUBO) matrix Q we seek is defined as $Q_{ij} = \tilde{Q}_{ij} + \tilde{Q}_{ji}$ for $i < j$, and $Q_{ii} = \tilde{Q}_{ii}$.

The last operation concludes the problem formulation so that it becomes suitable for a quantum annealer.

We do not delve into the embedding of the problem into the current available quantum annealing system for two primary reasons: Firstly, embedding is inherently tied to the architecture and could evolve with technological advancements, while the QUBO formulation remains constant. Secondly, in our experiments, we entrusted the embedding

to the library's methods since they are optimized for this operation [143].

A pivotal aspect in formulating our new model involves recognizing that in Equation (6.80) all components are either hyperparameters or labels, with the exception of the function $k(\cdot,\cdot)$, traditionally a classical step. In the literature, various classical functions have been chosen for k, but our proposal is to use a quantum kernel from gate-based quantum computation.

Throughout our experiments, we utilized the Advance 4.1 machine from D-Wave for all annealing experiments.

From Classical to Quantum Kernel

As previously discussed, the Gaussian kernel is the most used radial basis function (RBF) kernel. It facilitates the computation of the similarity between each pair of points in a higher-dimensional space while circumventing explicit calculations. Once an RBF kernel is selected, the similarities between each pair of points are computed and stored in a kernel matrix. Due to the symmetry of the distance function, it naturally follows that both the kernel matrix and, subsequently, $\tilde{Q}$ are symmetric as well.

In our proposal, we assume that the kernel is a Quantum Kernel. This approach allows us to leverage the exponentially higher dimension of the Hilbert space to separate the data effectively.

For the sake of simplicity, in the experiment we propose as a proof of concept, we employ the simple algorithm of the quantum inner product to compute the similarity between two vectors in the Hilbert space (Figure 6.2). Consider two points x_i and x_j along with their respective feature maps A and B, such that $|x_i\rangle = A\,|0\rangle$ and $|x_j\rangle = B\,|0\rangle$ on a M qubit register. Our objective is to derive the similarity s between the two through the quantum inner product:

$$B^\dagger A\,|0\rangle = a_0\,|0\rangle + \sum_{k=1}^{2^M-1} a_k\,|k\rangle \tag{6.81}$$

$$a_0 = \langle 0|B^\dagger A\,|0\rangle = \langle x_j|x_i\rangle = s \tag{6.82}$$

where $|k\rangle$ and a_k are, respectively, the kth standard basis vector and its coefficient. It is evident that if $x_i = x_j$, then a_0 is equal to 1, while if they are orthogonal to each other, the result will be 0. This behavior describes a similarity metric in a high-dimensional space, precisely what we sought to integrate into our Quantum Support Vector Machine.

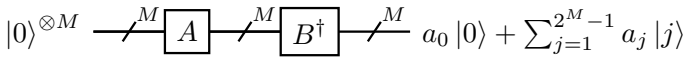


Figure 6.2: Generic representation of the quantum inner product of two vectors in a M qubit circuit. A and B represent feature map circuits that encode the two vectors, while a_0 is the similarity metric that we are looking for.

The final step to complete the QgSVM involves determining two feature maps, A and B, capable of encoding the data in the Hilbert space. To obtain the quantum inner product between vectors, we employ the same feature map technique for both A and B. Additionally, given that each vector x_i for all $i \in \{0, \ldots, N-1\}$ possesses the same number of features γ, it follows that the number of qubits in every circuit used for gate-based computation depends solely on γ and the chosen feature map technique.

In our study, we opt for one of the widely employed feature map techniques known as the Pauli expansion circuit [33]. The Pauli expansion circuit, denoted as U, serves as a data encoding circuit that transforms input data $\vec{x} \in \mathbb{R}^N$, where N represents the feature dimension, as

$$U_{\Phi(\vec{x})} = \exp\left(i \sum_{S \in \mathcal{I}} \phi_S(\vec{x}) \prod_{i \in S} P_i\right). \tag{6.83}$$

Here, S represents a set of qubit indices describing the connections in the feature map, $\mathcal{I}$ is a set containing all these index sets, and $P_i \in \{I, X, Y, Z\}$. The default data mapping is

$$\phi_S(\vec{x}) = \begin{cases} x_i \text{ if } S = \{i\} \\ \prod_{j \in S}(\pi - x_j) \text{ if } |S| > 1 \end{cases}. \tag{6.84}$$

This technique offers various degrees of freedom, including the choice of the Pauli gate, the number of circuit repetitions, and the type of entanglement. In our implementation, we select a second-order Pauli-Z evolution circuit, a well-known instance of the Pauli expansion circuit readily available in libraries such as Qiskit [144]. A more detailed formulation will be provided in the next subsection.

Our decision to use this specific feature map aligns with prior research by Havlicek et al. [33], where they leverage a Pauli Z expansion circuit to train a Quantum gate-based Support Vector Machine (QgSVM). It is essential to note that this choice is arbitrary and the feature map can be substituted with a variety of circuits. One important constraint applied to the feature map circuits is that, in a real-world implementation, they must encode data in a manner not easily simulated by classical computers. This requirement is necessary, though not sufficient, for achieving a quantum advantage.

Given that the focus of this paper is on proposing a new methodology rather than experimentally proving quantum advantage, we opted to simulate the quantum gate-based environment on classical hardware during the experimental phase.

Feature Map

As previously discussed, the feature map we chose for our experiments is a variation of the Pauli expansion gate known as the ZZ feature map. We employ a linearly entangled, single repetition, N-qubit ZZ feature map, illustrated in Figure 6.3. The rotation gates in this context involve rotations around the Z-axis:

$$R_Z(\theta) = \begin{pmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{pmatrix} \tag{6.85}$$

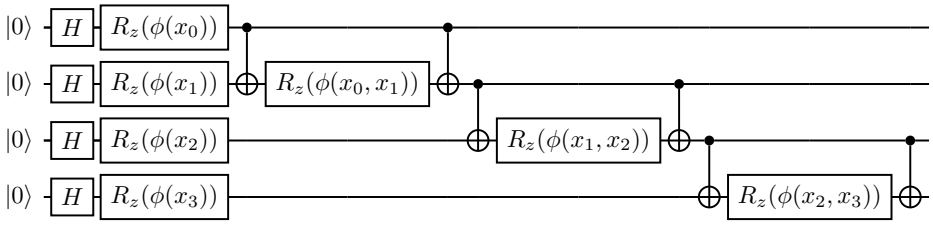


Figure 6.3: As an example of the feature map used, consider the circuit that takes input $\vec{x} = \{x_0, x_1, x_2, x_3\}$ and performs the encoding of classical data in the quantum space. The circuit depicted in the figure represents the first part of the inner product circuit. The second part involves the adjoint of this circuit with a different vector $\vec{z}$ as input.

The function $\phi(\cdot)$ is the one described in Equation (6.84) that assumes the default forms depending on how many parameters are passed:

$$\begin{aligned} \phi(x) &= x \\ \phi(x_0, x_1) &= (\pi - x_0)(\pi - x_1) \end{aligned} \tag{6.86}$$

Once the kernel is constructed, it can generates the similarity score between any pair of data points. We utilize this kernel to compile a kernel matrix K, where $K_{n,m} = k(x_n, x_m)$, similar to the classical SVM approach. As mentioned earlier, the matrix is symmetric over the main diagonal since $k(x_n, x_m) = k(x_m, x_n)$.

It is crucial to highlight the computational bottleneck in this methodology: obtaining the similarity score involves repeated executions of the circuit to derive the probability. Once we have the probability p_0 of the state $|0\rangle^{\otimes n}$, we can approximate s to a value $\hat{s}$, which, for our purposes, serves as a valid proxy for s. Our proposal addresses this problem by generating an ensemble of models that compute smaller K. This approach is a standard practice in classical SVM, as documented in [145]. It has been demonstrated to enhance generalization and decrease the computational cost in the kernel part of the algorithm.

Experiments

Our experiments serve as a demonstration of the viability of the proposed methodology. All experiments involve binary classification problems conducted on pairs of classes from the MNIST dataset [146], following preprocessing. Due to the limited capabilities of current quantum computers and simulations, and to maintain simplicity for demonstration purposes, we applied Principal Component Analysis (PCA) [147] to each image, considering only the first two principal components as input to the model (Table 6.2 reports the explained variance). The resulting values are then mapped into the range $x_i \in [0, \pi/2]$ to fully exploit the properties of the Quantum gate-based Support Vector Machine (QgSVM). Simultaneously, the labels are adapted to the Quantum Annealer-based Support Vector Machine (QaSVM) standard, where $y_i \in \{-1, 1\}$.

Dataset	Explained Variance
MN09	30.8 %
MN38	21.0 %
MN47	23.1 %
MN56	25.4 %

Table 6.2: The sum of the explained variance based on the first two components from the PCA of each dataset. Principal Component Analysis generates new features to represent the data and orders them based on the variance they explain. Our implementation considers only the first two components.

We present four different experiments utilizing class pairs 0–9, 3–8, 4–7 and 5–6; referred to as MN09, MN38, MN47, and MN56, respectively. In each experiment, we

selected 500 data points with balanced classes and divided them into a training set (300 data points) and a test set (200 data points), each with two features (referred to as γ).

The experiments are divided into three phases: hyperparameter tuning, training, and testing.

Hyperparameter Tuning: Initially, a 4-fold Monte Carlo cross-validation is performed on the training set for hyperparameter tuning. The optimized hyperparameters include K, B, and ξ from Equation (6.78). While an exhaustive search on the hyperparameter space is beyond the scope of this work, additional information and results on the effect of hyperparameter tuning can be found in [81]. To assess the performance of each model, we compute the Area Under the Receiver Operating Characteristic curve (AUROC) and the Area Under the Precision–Recall curve (AUPCR), as shown in Figure 6.4.

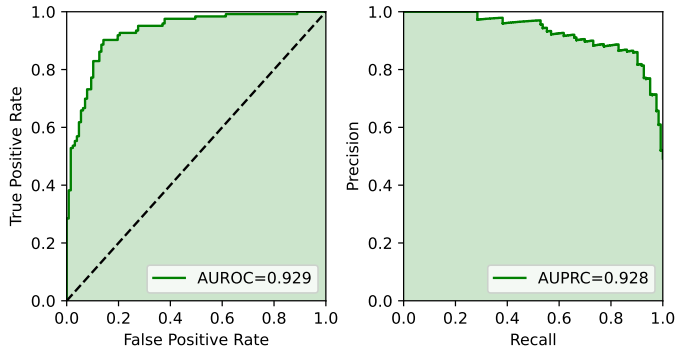


Figure 6.4: Example of AUROC and AUPRC curves. This graph illustrates the True Positive vs. False Positive curve (AUROC on the **left**) and the area under the Precision vs. Recall curve (AUPRC on the **right**) of a classifier when predicting classes for the training set on the dataset MN09 while performing hyperparameter tuning.

Training: Once the optimal hyperparameters are determined, we instantiate the best model for each dataset and train it on the entire training set. Following the original proposal by Willsch et al. [81], and to address the challenge of limited connectivity in the annealing hardware, the entire training set is divided into six small, disjoint subsets (referred to as folds), each containing approximately 50 samples. The approach involves constructing an ensemble of quantum weak Support Vector Machines, with each classifier trained on one of the subsets.

This process unfolds in two steps. First, for each fold, the top twenty solutions (qSVM$(B, K, \xi)\#i$ where $i \in \{1, \cdots, 20\}$ is the index of the best solutions) obtained

from the annealer are combined by averaging their respective decision functions. Since the decision function is linear in the coefficients, and the bias for each fold l, denoted as $b^{(l,i)}$, is computed from $\alpha_n^{(l,i)}$ using Equation (6.74), this step effectively produces a single classifier with an effective set of coefficients given by

$$\alpha_n^{(l)} = \sum_i \alpha_n^{(l,i)}/20$$

and bias given by

$$b^l = \sum_i b^{(l,i)}/20$$

Second, an average is taken over the six subsets. It is important to note that the data points $(\mathbf{x}_n, y_n)^l$ are unique for each l. The complete decision function is expressed as

$$F(\mathbf{x}) = \frac{1}{L} \sum_{nl} \alpha_n^{(l)} y_n^{(l)} k\left(\mathbf{x}_n^{(l)}, \mathbf{x}\right) + b \tag{6.87}$$

where L is the number of folds and $b = \sum_l b^{(l)}/L$. Similar to the formulation showed before, the decision for the class label of a point $\mathbf{x}$ is determined by $\widetilde{t} = \text{sign}(F(\mathbf{x}))$.

Testing: After the training concludes, we test the models on the 200 data points of the test set. Similar to the training phase, a total of 6 weak SVMs, each resulting from the combination of the 20 annealer's solutions, are utilized to determine the class of the data point.

A visual representation of the process can be found in Figure 6.5.

6.3.3 Results

In this section, we present the results obtained from the new methodology. As discussed earlier, these experiments should be viewed as a proof of concept and as evidence of the correctness of the methodology, rather than an exhaustive search for the best possible model. We searched over a narrow hyperparameter space to obtain a model slightly superior to the most naive approach. For each dataset, we performed hyperparameters tuning in the space $B = \{2, 10\}$, $K = \{2, 3\}$, and $\xi = \{1, 2\}$, where B is the basis used to represent α (Equation (6.77)), K is the number of a_k used to represent α (Equation (6.77)), and ξ is the coefficient used when adding the constraint in Equation (6.78). The best hyperparameter for each model is shown in Table 6.3. It is noteworthy to emphasize that the choice of hyperparameters can significantly impact the results, even

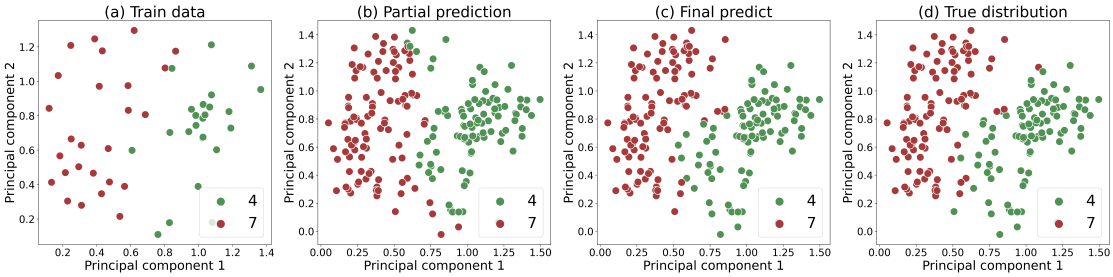


Figure 6.5: This image illustrates various stages of prediction. In (**a**), one of the subsets of the training set is displayed; a weak QSVM trains on it. (**b**) shows the prediction of the weak model on the test data, obtained by combining the 20 results computed by the annealer. (**c**) contains the result of aggregating the predictions from the six weak QSVMs. Lastly, in (**d**), the real distribution of labels on the test set is depicted. It can be observed that the partial prediction is significantly off-target with respect to the true distribution, while the final prediction is very close to it. This is partly the result of the folding of the training set, which allows the weak classifiers to see only a small subset of the training data, and partly stems from the intrinsic randomness inherent in quantum processes like quantum annealing.

with a relatively shallow search as the one we performed.

Once we have selected the best hyperparameters for each model, we proceed training six weak classifiers, each composed of 20 solutions returned by the quantum annealer. The combination of these six can be regarded as one model trained on the entire dataset. Finally, the model is tested on the test set. The results of each model are reported in Table 6.4, while Figure 6.6 illustrates the predictions on the test set for each dataset, highlighting the errors.

Dataset	**B**	**X**	**ξ**	**T. acc**	**T. AUROC**	**T. AUPCR**	**V. acc**	**V. AUROC**	**V. AUPCR**
MN09	2	3	1	0.9354	0.9839	0.9845	0.9375	0.9877	0.9886
MN38	2	3	1	0.8667	0.9083	0.8391	0.8722	0.9372	0.9067
MN47	2	3	1	0.9354	0.9742	0.9795	0.9014	0.9674	0.9708
MN56	2	2	2	0.8937	0.9675	0.9640	0.8972	0.9648	0.9602

Table 6.3: Best results for hyperparameter tuning on each dataset. The columns B, K, and ξ represent the hyperparameter values that obtained the best AUPCR score. T. (*Train*) and V. (*Validation*) in the column names stand for the training and validation sets. acc, AUROC, and AUPCR are accuracy, the area under ROC, and the area under PCR scores.

Dataset	Precision	Recall	f1-Score
MN09	0.91	0.91	0.90
MN38	0.84	0.80	0.80
MN47	0.97	0.97	0.97
MN56	0.89	0.87	0.87

Table 6.4: This table shows the macro average of Precision, Recall, and f1-score for each dataset on the test set.

The results are consistently positive, showcasing the models' high accuracy in predicting the classes of the datasets. Some errors can be attributed to the limited information contained in only two features, even though these features are the most expressive in Principal Component Analysis.

As hypothesized during formulation, the experiments provide evidence of the effectiveness of this approach and highlight the synergistic power of the two quantum technologies when employed together.

6.3.4 Discussion

The results obtained from our Hybrid Quantum Support Vector Machine (Hybrid QSVM) showcase great promise across various aspects. Remarkably, we achieved favorable outcomes using only two features derived through Principal Component Analysis (PCA) while harnessing both quantum gate-based technology (simulated) and quantum annealing (real hardware).

As previously mentioned, our experimental focus was not centered on seeking the optimal model or comparing its performance against classical approaches. Instead, our primary goal was to experimentally validate the viability of our proposed approach. We deliberately invested minimal effort in performance optimization, aside from a modest hyperparameter tuning step that surpassed the naive approach. Consequently, these results serve as a baseline for potential future enhancements to the model.

During the tuning phase, we concentrated solely on hyperparameters related to the annealing component of the algorithm (specifically, B, K, and ξ), without delving into the gate-based part. This decision is motivated by three main factors.

Firstly, the existing literature on Quantum gate-based Support Vector Machines (QgSVM)

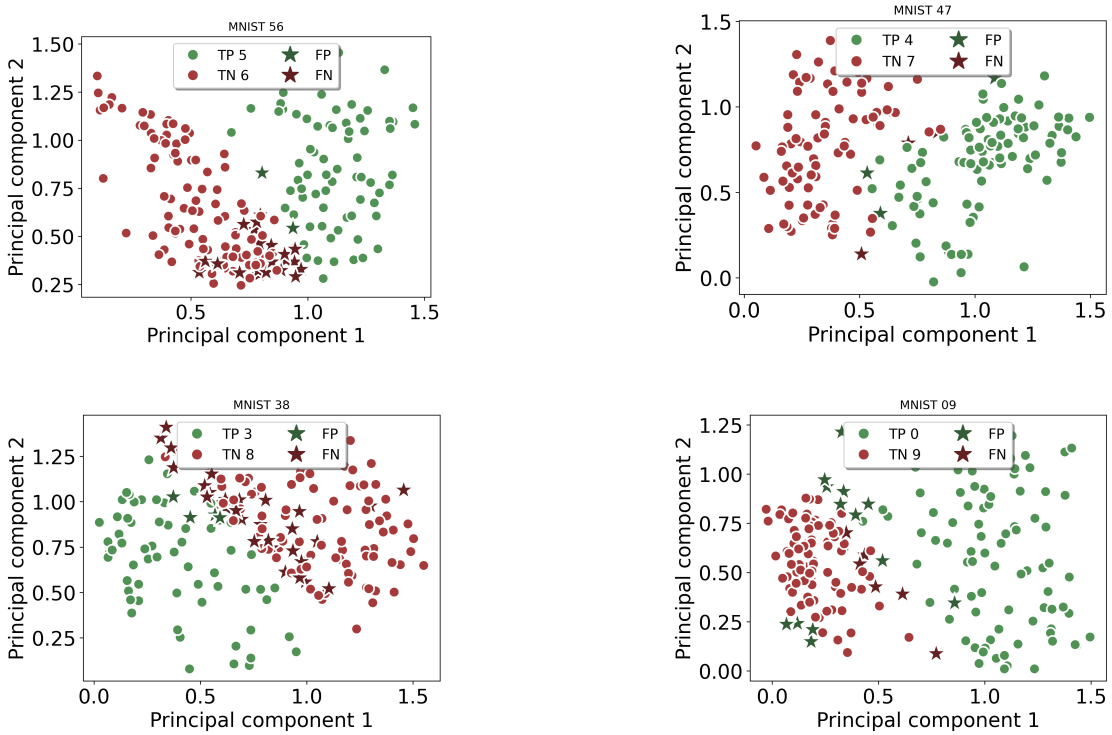


Figure 6.6: The figures illustrate the models' predictions on each dataset. The axes represent the first two component obtained with PCA and used for classification. Circles denote correctly classified instances, while stars indicate misclassifications. The color of the stars represents the class to which they have been erroneously assigned. Each graph considers the lower-numbered class as "Positive" and the higher-numbered class as "Negative".

is more developed compared to Quantum annealing Support Vector Machines (QaSVM), making exploration of the latter more pertinent from a research standpoint.

Secondly, we had access to real annealing hardware through the D-Wave Leap program [143]. Although access to real gate-based hardware is possible through the IBM Quantum Initiative, leveraging both technologies concurrently was impractical due to extended wait times in the hardware queues. Moreover, given our constraint of limiting each data point to two features, a two-qubit gate-based quantum hardware was sufficient for simulating the quantum kernel, further simplifying the process.

Third, and directly linked to the point above, we considered that an easy-to-simulate quantum kernel cannot surpass classical performance. This means that one of the reasons why the QgSVM in our experiment performed sub-optimally is because we could

not access hard-to-simulate quantum hardware. For this reason, optimizing its hyperparameters was deemed futile. The same applies to the circuit design, we decided to keep it fixed for the whole experimental phase, since any 2-qubit circuit (with depth within reason) can be simulated.

It is essential to clarify that the three reasons outlined above pertain to the experimental setup and not the specific model's performance on the datasets. While tuning different hyperparameters or exploring more values of the already optimized ones might enhance performance on specific datasets, it deviates from the primary objective of this paper.

When talking about the goal of this proposal and the advantages it brings, we need to highlight that the quantum annealer is on the verge of addressing industrial optimization challenges [143]. Simultaneously, the inherent variational nature of the QgSVM positions it favorably for the Noisy Intermediate-scale Quantum (NISQ) era. The convergence of these two approaches is nearing completion, offering several potential benefits. QgSVM can leverage the exponentially large Hilbert space $\mathcal{H}$ to more effectively capture similarities and distances between points than classical kernel tricks. This allows it to distinguish between points that are traditionally considered challenging. Moreover, the quantum annealer's speed surpasses classical optimization methods, providing a solution quicker.

Recent studies on Quantum gate-based Support Vector Machines (QgSVM) have highlighted certain theoretical limitations under specific assumptions. In [148], the authors connect the exponential dimension of the feature space to a limitation in the model's ability to generalize. One proposed solution, discussed both theoretically [149] and empirically [150], involves introducing a new hyperparameter to regulate the bandwidth.

We argue that our proposal can further enhance the generalization ability of QgSVM. Building on the findings from the original QaSVM study [81], we know that Quantum annealing Support Vector Machines can outperform the single classifier obtained by classical SVM optimization in the same computational problem, and this can be directly translated to an improvement on the QgSVM since its optimization is done classically. This improvement stems from the ensembling technique used and the intrinsic ability of the annealer to generate a distribution of solutions close to optimal, thereby enhancing generalization (similar results can be found in [151, 152] for different annealing machine learning models). Moreover, since our proposal is not tied to a specific implementation, it

can employ various strategies to mitigate the limitations of QgSVM like the one discussed above.

Future work in this direction could involve a theoretical analysis of these features.

6.3.5 Conclusions

In the realm of quantum computing, the conventional approach involves selecting one technology and working within its framework, often neglecting the potential synergies that could arise from mixing different quantum technologies to harness their respective strengths. In this paper, we propose a novel version of Quantum Support Vector Machines (QSVM) that capitalizes on the advantages of both gate-based quantum computation and quantum annealing.

Our proposal positions itself within the context of Quantum Support Vector Machines (QSVM) from both technological points of view improving on both techniques. We enhance the generalization ability of standard QgSVM through the ensemble technique and leverage the intrinsic capability of Quantum annealing Support Vector Machines (QaSVM) to generate a distribution of suboptimal solutions during optimization within constant annealing time.

The ensemble method, similarly to its classical counterpart, not only enhances generalization but also reduces the computation time of the kernel matrix and the accesses to the quantum computer. For a dataset X with n samples and γ features, the full kernel matrix is composed of $n^2/2$ elements, implying $\mathcal{O}(sn^2)$ calls to the (gate-based) quantum computer where s is the number of shots needed to accurately compute the similarity score. Among other important factors [153], s is strongly linked to the dimensionality of the Hilbert space [149] that in turn depends on the kernel implementation and γ. Our ensemble of m Kernels quadratically reduces the needed number of accesses to $\mathcal{O}(s(n/m)^2)$ while only needing $\mathcal{O}(m)$ accesses to the quantum annealer. In this context, m becomes a new hyperparameter and its value is highly dependent on the specific problem and the implementation in question.
To conclude, we can summarize our contribution in three points.

First, we introduce a method that combines quantum technologies, leveraging the complementarity between gate-based quantum computation and quantum annealing in the context of Support Vector Machines. While previous research has touched on the integration of these approaches for solving large-size Ising problems [154], to the best of our knowledge, our work is the first to explore this fusion in the context of classification

problems.

Second, we provide experimental validation of our approach. Notably, the annealing component of our experiments is conducted on real quantum hardware, demonstrating the feasibility of our methodology in the early era of Noisy Intermediate-Scale Quantum (NISQ) computation.

Third, we establish a baseline result for future hybrid technology approaches, particularly on one of the most widely used datasets in classical machine learning.

Future work in this area is straightforward. As quantum technologies advance, our approach should undergo testing on quantum hardware to validate its capabilities. Additionally, there is a need for more extensive hyperparameter tuning to achieve optimal results. We recommend exploring and proposing different quantum kernel methods based on the characteristics of the data to further enhance performance. Simultaneously, a rigorous proof of the generalization power of our model is essential to prove the positive interaction of the technologies in a more rigorous manner.

It is crucial to note that the implementation of our methodology is not rigidly tied to the specific formulations described here. Rather, it adapts to the state of the art for each technology. The core of our proposal lies in the fusion of approaches, allowing for flexibility as advancements occur in individual technologies.

Quantum machine learning (QML) currently stands as a focal point of research, offering the potential to revolutionize various applications through the utilization of quantum computational power and innovative algorithmic models like Variational Algorithms. Similar to any scientific field, as it expands, new limitations come to light. However, simultaneously, researchers develop techniques to overcome and mitigate these restrictions. Despite the increasing attention, the field is still in its early stages, necessitating further exploration to unveil its practical benefits.

Bibliography

[1] A. Macaluso, L. Clissa, S. Lodi, and C. Sartori, "A variational algorithm for quantum neural networks," in *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part VI 20*. Springer, pp. 591–604.

[2] M. Mottonen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, "Transformation of quantum states using uniformly controlled rotations."

[3] P. Ehrenfest, "Welche Züge der Lichtquantenhypothese spielen in der Theorie der Wärmestrahlung eine wesentliche Rolle?" vol. 341, no. 11, pp. 91–118. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/andp.19113411106

[4] A. Einstein, "Die Plancksche Theorie der Strahlung und die Theorie der spezifischen Wärme," vol. 327, no. 1, pp. 180–190. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/andp.19063270110

[5] E. Schrödinger, "Quantisierung als Eigenwertproblem," vol. 384, no. 4, pp. 361–376. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/andp.19263840404

[6] M. Giliberti and L. Lovisetti, *Heisenberg's 1925 "Umdeutung" Paper: A Commented Translation from a Physicist's Perspective*, ser. SpringerBriefs in History of Science and Technology. Springer Nature Switzerland. [Online]. Available: https://link.springer.com/10.1007/978-3-031-96921-8

[7] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines," vol. 22, no. 5, pp. 563–591. [Online]. Available: http://link.springer.com/10.1007/BF01011339

[8] C. H. Bennett, "Logical Reversibility of Computation," vol. 17, no. 6, pp. 525–532. [Online]. Available: https://ieeexplore.ieee.org/document/5391327/

[9] R. P. Feynman, "Simulating physics with computers," vol. 21, no. 6–7, pp. 467–488. [Online]. Available: http://link.springer.com/10.1007/BF02650179

[10] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," vol. 26, no. 5, pp. 1484–1509. [Online]. Available: httpsdoi.org10.1137%2Fs0097539795293172

[11] Qureca. Quantum Initiatives Worldwide 2025. [Online]. Available: https://www.qureca.com/quantum-initiatives-worldwide/

[12] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 1st ed. Cambridge University Press. [Online]. Available: https://www.cambridge.org/core/product/identifier/9780511976667/type/book

[13] G. B. Malykin, "Use of the poincare sphere in polarization optics and classical and quantum mechanics. Review," vol. 40, no. 3, pp. 175–195. [Online]. Available: http://link.springer.com/10.1007/BF02676342

[14] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," vol. 299, no. 5886, pp. 802–803. [Online]. Available: https://www.nature.com/articles/299802a0

[15] A. Einstein, B. Podolsky, and N. Rosen, "Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?" vol. 47, no. 10, pp. 777–780. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.47.777

[16] A. Aspect, J. Dalibard, and G. Roger, "Experimental Test of Bell's Inequalities Using Time- Varying Analyzers," vol. 49, no. 25, pp. 1804–1807. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.49.1804

[17] E. Schrödinger, "An Undulatory Theory of the Mechanics of Atoms and Molecules," vol. 28, no. 6, pp. 1049–1070. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.28.1049

[18] D. Willsch, M. Willsch, C. D. Gonzalez Calaza, F. Jin, H. De Raedt, M. Svensson, and K. Michielsen, "Benchmarking Advantage and D-Wave 2000Q quantum

annealers with exact cover problems," vol. 21, no. 4, p. 141. [Online]. Available: https://link.springer.com/10.1007/s11128-022-03476-y

[19] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn *et al.*, "Quantum optimization using variational algorithms on near-term quantum devices," vol. 3, no. 3, p. 030503.

[20] D. Wecker, M. B. Hastings, and M. Troyer, "Progress towards practical quantum variational algorithms," vol. 92, no. 4, p. 042303.

[21] V. Giovannetti, S. Lloyd, and L. Maccone, "Quantum random access memory."

[22] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm."

[23] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, "Evaluating analytic gradients on quantum hardware," vol. 99, no. 3, p. 032331. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.99.032331

[24] G. De Palma, M. Marvian, D. Trevisan, and S. Lloyd, "The Quantum Wasserstein Distance of Order 1," vol. 67, no. 10, pp. 6627–6643. [Online]. Available: https://ieeexplore.ieee.org/document/9420734/

[25] H.-Y. Huang, R. Kueng, and J. Preskill, "Predicting many properties of a quantum system from very few measurements," vol. 16, no. 10, pp. 1050–1057. [Online]. Available: https://www.nature.com/articles/s41567-020-0932-7

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," vol. 323, no. 6088, pp. 533–536.

[27] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli, and S. Woerner, "The power of quantum neural networks," vol. 1, no. 6, pp. 403–409. [Online]. Available: https://www.nature.com/articles/s43588-021-00084-1

[28] F. Tacchino, C. Macchiavello, D. Gerace, and D. Bajoni, "An artificial neuron implemented on an actual quantum processor, zak1998quantum," vol. 5, no. 1, p. 26.

[29] E. Grant, M. Benedetti, S. Cao, A. Hallam, J. Lockhart, V. Stojevic, A. G. Green, and S. Severini, "Hierarchical quantum classifiers," vol. 4, no. 1, pp. 1–8.

[30] W. Huggins, P. Patil, B. Mitchell, K. B. Whaley, and E. M. Stoudenmire, "Towards quantum machine learning with tensor networks," vol. 4, no. 2, p. 024001.

[31] D. Liu, S.-J. Ran, P. Wittek, C. Peng, R. B. García, G. Su, and M. Lewenstein, "Machine learning by unitary tensor network of hierarchical tree structure," vol. 21, no. 7, p. 073059.

[32] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, "Parameterized quantum circuits as machine learning models," vol. 4, no. 4, p. 043001. [Online]. Available: httpsdoi.org10.10882058-9565ab4eb5

[33] V. Havlicek, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, "Supervised learning with quantum enhanced feature spaces."

[34] A. Macaluso, L. Clissa, S. Lodi, and C. Sartori, "Quantum Splines for Non-Linear Approximations," in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, ser. CF '20. Association for Computing Machinery, pp. 249–252. [Online]. Available: httpsdoi.org10.11453387902.3394032

[35] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for solving linear systems of equations."

[36] Y. Cao, G. G. Guerreschi, and A. Aspuru-Guzik, "Quantum Neuron: An elementary building block for machine learning on quantum computers."

[37] W. Hu, "Towards a Real Quantum Neuron," vol. 10, no. 03, pp. 99–109.

[38] M. Schuld, I. Sinayskiy, and F. Petruccione, "Simulating a perceptron on a quantum computer," vol. 379, no. 7, pp. 660–663.

[39] M. Lubasch, J. Joo, P. Moinier, M. Kiffner, and D. Jaksch, "Variational quantum algorithms for nonlinear problems," vol. 101, no. 1, p. 010301. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.101.010301

[40] M. Larocca, P. Czarnik, K. Sharma, G. Muraleedharan, P. J. Coles, and M. Cerezo, "Diagnosing Barren Plateaus with Tools from Quantum Optimal Control," vol. 6, p. 824. [Online]. Available: http://arxiv.org/abs/2105.14377

[41] Y. Beaujeault-Taudière. Systematic improvement of the quantum approximate optimisation ansatz for combinatorial optimisation using quantum subspace expansion. [Online]. Available: http://arxiv.org/abs/2506.18594

[42] X. Lee, X. Yan, N. Xie, D. Cai, Y. Saito, and N. Asai, "Iterative layerwise training for the quantum approximate optimization algorithm," vol. 109, no. 5, p. 052406. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.109.052406

[43] X.-H. Ni, Y.-S. Wu, B.-B. Cai, W.-M. Li, S.-J. Qin, and F. Gao. An Adaptive Mixer Allocation Algorithm for the Quantum Alternating Operator Ansatz. [Online]. Available: http://arxiv.org/abs/2412.19621

[44] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, and D. J. Wineland, "Randomized benchmarking of quantum gates," vol. 77, no. 1, p. 012307. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.77.012307

[45] A. W. Cross, E. Magesan, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Scalable randomised benchmarking of non-Clifford gates," vol. 2, no. 1, p. 16012. [Online]. Available: https://www.nature.com/articles/npjqi201612

[46] A. D. Córcoles, J. M. Gambetta, J. M. Chow, J. A. Smolin, M. Ware, J. Strand, B. L. T. Plourde, and M. Steffen, "Process verification of two-qubit quantum gates by randomized benchmarking," vol. 87, no. 3, p. 030301. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.87.030301

[47] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," vol. 14, no. 6, pp. 595–600. [Online]. Available: https://www.nature.com/articles/s41567-018-0124-x

[48] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," vol. 100, no. 3, p. 032328.

[49] A. Erhard, J. J. Wallman, L. Postler, M. Meth, R. Stricker, E. A. Martinez, P. Schindler, T. Monz, J. Emerson, and R. Blatt, "Characterizing large-scale quantum computers via cycle benchmarking," vol. 10, no. 1, p. 5347. [Online]. Available: https://www.nature.com/articles/s41467-019-13068-7

[50] T. Lubinski, S. Johri, P. Varosy, J. Coleman, L. Zhao, J. Necaise, C. H. Baldwin, K. Mayer, and T. Proctor, "Application-Oriented Performance

Benchmarks for Quantum Computing," vol. 4, pp. 1–32. [Online]. Available: http://arxiv.org/abs/2110.03137

[51] S. Martiel, T. Ayral, and C. Allouche, "Benchmarking quantum co-processors in an application-centric, hardware-agnostic and scalable way," vol. 2, pp. 1–11. [Online]. Available: http://arxiv.org/abs/2102.12973

[52] E. Pelofske, "Comparing Three Generations of D-Wave Quantum Annealers for Minor Embedded Combinatorial Optimization Problems," vol. 10, no. 2, p. 025025. [Online]. Available: http://arxiv.org/abs/2301.03009

[53] R. Blume-Kohout and K. C. Young, "A volumetric framework for quantum computer benchmarks," vol. 4, p. 362. [Online]. Available: https://quantum-journal.org/papers/q-2020-11-15-362/

[54] J. R. Finžgar, P. Ross, L. Hölscher, J. Klepsch, and A. Luckow, "QUARK: A Framework for Quantum Computing Application Benchmarking." [Online]. Available: https://arxiv.org/abs/2202.03028

[55] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing," vol. 7, p. 1062. [Online]. Available: https://quantum-journal.org/papers/q-2023-07-20-1062/

[56] F. Barbaresco, L. Rioux, C. Labreuche, M. Nowak, N. Olivier, D. Nicolazic, O. Hess, A.-L. Guilmin, R. Wang, T. Sassolas, S. Louise, K. Snizhko, G. Misguich, A. Auffèves, R. Whitney, E. Vergnaud, and F. Schopfer. BACQ – Application-oriented Benchmarks for Quantum Computing. [Online]. Available: http://arxiv.org/abs/2403.12205

[57] M. Clerc, "Graph colouring: A polynomial complexity quantum algorithm." [Online]. Available: https://rgdoi.net/10.13140/RG.2.2.36154.77760

[58] H. Buhrman, R. Cleve, J. Watrous, and p. u. family=Wolf, given=Ronald, "Quantum fingerprinting," vol. 87, no. 16, p. 167902. [Online]. Available: http://arxiv.org/abs/quant-ph/0102001

[59] B. Koczor, "Exponential Error Suppression for Near-Term Quantum Devices," vol. 11, no. 3, p. 031057. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.11.031057

[60] J. C. Garcia-Escartin and P. Muñoz-Escoí, "Swap test for an arbitrary number of quantum states," vol. 13, no. 1–2, pp. 0019–0030.

[61] J. Knörzer, D. Malz, and J. I. Cirac, "Cross-platform verification in quantum networks," vol. 107, no. 6. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.107.062424

[62] L. Cincio, Y. Subaşı, A. T. Sornborger, and P. J. Coles, "Learning the quantum algorithm for state overlap," vol. 20, no. 11, p. 113022. [Online]. Available: http://dx.doi.org/10.1088/1367-2630/aae94a

[63] J. C. Garcia-Escartin and P. Chamorro-Posada, "The SWAP test and the Hong-Ou-Mandel effect are equivalent," vol. 87, no. 5, p. 052330. [Online]. Available: http://arxiv.org/abs/1303.6814

[64] M. Kada, H. Nishimura, and T. Yamakami, "The efficiency of quantum identity testing of multiple states," vol. 41, no. 39, p. 395309. [Online]. Available: https://iopscience.iop.org/article/10.1088/1751-8113/41/39/395309

[65] J. Biamonte and P. Wittek, "Learning the quantum algorithm for state overlap," vol. 549, no. 7671, pp. 195–200.

[66] A. Romito and L. Cincio, "Quantum autoencoders for efficient compression of quantum data," vol. 97, no. 6, p. 062309.

[67] B. Koczor, "Rigorous noise reduction with quantum autoencoders," vol. 104, no. 1, p. 012411.

[68] L. Cincio, M. A. W, and P. Wittek, "Noise-Assisted Quantum Autoencoder."

[69] B. Koczor, "A Universal Training Algorithm for Quantum Deep Learning."

[70] L. Cincio, M. A. W, and P. Wittek, "On compression rate of quantum autoencoders: Control design, numerical and experimental realization," vol. 97, no. 6, p. 062309.

[71] Z. Cai, R. Babbush, S. C. Benjamin, S. Endo, W. J. Huggins, Y. Li, J. R. McClean, and T. E. O'Brien, "Quantum Error Mitigation," vol. 95, no. 4, p. 045005. [Online]. Available: http://arxiv.org/abs/2210.00921

[72] X. Zhang *et al.*, "Generic detection-based error-mitigation using quantum autoencoders," vol. 103, no. 4, p. 040403.

[73] B. Koczor, "Quantum Error Correction with Quantum Autoencoders," vol. 7, p. 942.

[74] A. Rizzo *et al.*, "Quantum Autoencoders for anomaly detection to identify Gamma-Ray Bursts."

[75] A. Sireesh, A. Alhajri, and S. Kais, "Dimensionality Reduction with Variational Encoders Based on Subsystem Purification," vol. 10, no. 18, p. 3469.

[76] Y. Yu *et al.*, "QOBRA: A Quantum Operator-Based Autoencoder for De Novo Molecular Design."

[77] Y. Huang and Z. Chen, "Optimized Quantum Autoencoder."

[78] T. Suzuki, T. Hasebe, and T. Miyazaki. Quantum support vector machines for classification and regression on a trapped-ion quantum computer. [Online]. Available: http://arxiv.org/abs/2307.02091

[79] M. Schuld and F. Petruccione, *Supervised Learning with Quantum Computers*, 1st ed. Springer Publishing Company, Incorporated.

[80] M. Schuld, "Supervised quantum machine learning models are kernel methods," arXiv.

[81] D. Willsch, M. Willsch, H. De Raedt, and K. Michielsen, "Support vector machines on the D-Wave quantum annealer," vol. 248, p. 107006.

[82] C. Headquarters, "Technical Description of the D-Wave Quantum Processing Unit," Academic Press.

[83] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," in *Proceedings of the 23rd International Conference on Machine Learning - ICML '06*. ACM Press.

[84] A. Delilbasic, B. L. Saux, M. Riedel, K. Michielsen, and G. Cavallaro, "A Single-Step Multiclass SVM based on Quantum Annealing for Remote Sensing Data Classification," arXiv.

[85] B. Dema, J. ARAI, and K. HORIKAWA, "Support vector machine for multiclass classification using quantum annealers," in *Proceedings of the DEIM Forum*.

[86] "Improvement of K-means Clustering Algorithm Based on Quantum State Similarity Measurement," vol. 9, no. 2. [Online]. Available: https://www.clausiuspress.com/article/15471.html

[87] T. Chen, A. Ray, A. Seshadri, D. Herman, B. Bach, P. Deshpande, A. Som, N. Kumar, and M. Pistoia. Provably faster randomized and quantum algorithms for k-means clustering via uniform sampling. [Online]. Available: https://arxiv.org/abs/2504.20982

[88] J.-N. Zaech, M. Danelljan, T. Birdal, and L. Van Gool. Probabilistic Sampling of Balanced K-Means using Adiabatic Quantum Computing. [Online]. Available: https://arxiv.org/abs/2310.12153

[89] M. B. Jain, K. G. Ratan, and R. Benni, "Quantum-Inspired Cluster Optimization: K-Means Versus Quantum K-Means," in *Fifth Congress on Intelligent Systems*, S. Kumar, E. A. Mary Anita, J. H. Kim, and A. Nagar, Eds. Springer Nature Singapore, vol. 1275, pp. 265–282. [Online]. Available: https://link.springer.com/10.1007/978-981-96-2694-6_18

[90] A. Zeguendry, Z. Jarir, and M. Quafafou, "Quantum Machine Learning: A Review and Case Studies," vol. 25, no. 2, p. 287. [Online]. Available: https://www.mdpi.com/1099-4300/25/2/287

[91] S. Kazarlis, A. Bakirtzis, and V. Petridis, "A genetic algorithm solution to the unit commitment problem," vol. 11, no. 1, pp. 83–92. [Online]. Available: http://ieeexplore.ieee.org/document/485989/

[92] I. Abdou and M. Tkiouat, "Unit Commitment Problem in Electrical Power System: A Literature Review," vol. 8, no. 3, p. 1357. [Online]. Available: http://ijece.iaescore.com/index.php/IJECE/article/view/9675

[93] D. Simopoulos, S. Kavatza, and C. Vournas, "Unit Commitment by an Enhanced Simulated Annealing Algorithm," in *2006 IEEE PES Power Systems Conference and Exposition*. IEEE, pp. 193–201. [Online]. Available: http://ieeexplore.ieee.org/document/4075743/

[94] F. Zhuang and F. Galiana, "Unit commitment by simulated annealing," vol. 5, no. 1, pp. 311–318. [Online]. Available: http://ieeexplore.ieee.org/document/49122/

[95] L. Montero, A. Bello, and J. Reneses, "A Review on the Unit Commitment Problem: Approaches, Techniques, and Resolution Methods," vol. 15, no. 4, p. 1296. [Online]. Available: https://www.mdpi.com/1996-1073/15/4/1296

[96] P. Halffmann, P. Holzer, K. Plociennik, and M. Trebing, "A Quantum Computing Approach for the Unit Commitment Problem," in *Operations Research Proceedings 2022*, O. Grothe, S. Nickel, S. Rebennack, and O. Stein, Eds. Springer International Publishing, pp. 113–120. [Online]. Available: https://link.springer.com/10.1007/978-3-031-24907-5_14

[97] B. Salgado, A. Sequeira, and L. P. Santos. A hybrid classical-quantum approach to highly constrained Unit Commitment problems. [Online]. Available: https://arxiv.org/abs/2412.11312

[98] N. Padhy, "Unit Commitment—A Bibliographical Survey," vol. 19, no. 2, pp. 1196–1205. [Online]. Available: http://ieeexplore.ieee.org/document/1295033/

[99] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*. Springer series in statistics New York, vol. 1, no. 10.

[100] K. Hornik, M. Stinchcombe, H. White *et al.*, "Multilayer feedforward networks are universal approximators." vol. 2, no. 5, pp. 359–366.

[101] S. Aaronson, "Read the fine print," vol. 11, no. 4, pp. 291–293. [Online]. Available: https://www.nature.com/articles/nphys3272

[102] J. A. Smolin and D. P. DiVincenzo, "Five two-bit quantum gates are sufficient to implement the quantum Fredkin gate," vol. 53, no. 4, pp. 2855–2856. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.53.2855

[103] M. Schuld, A. Bocharov, K. Svore, and N. Wiebe, "Circuit-centric quantum classifiers."

[104] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press.

[105] J. S. Judd, *Neural Network Design and the Complexity of Learning*. MIT press.

[106] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of Quantum Logic Circuits."

[107] T. Goto, Q. H. Tran, and K. Nakajima, "Universal Approximation Property of Quantum Feature Map."

[108] C. Bravo-Prieto, R. LaRose, M. Cerezo, Y. Subasi, L. Cincio, and P. J. Coles, "Variational quantum linear solver: A hybrid algorithm for linear systems."

[109] C. De Boor, *A Practical Guide to Splines*. Springer-Verlag New York, vol. 27.

[110] V. Markov, C. Stefanski, A. Rao, and C. Gonciulea, "A Generalized Quantum Inner Product and Applications to Financial Engineering."

[111] M. Maronese, C. Destri, and E. Prati, "Quantum activation functions for quantum neural networks," arXiv. [Online]. Available: httpsarxiv.orgabs2201.03700

[112] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, "Quantum algorithms revisited," vol. 454, no. 1969, pp. 339–354. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1998.0164

[113] J. R. Rice, "A Theory of Condition," vol. 3, no. 2, pp. 287–310. [Online]. Available: https://doi.org/10.1137/0703023

[114] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, "Quantum Amplitude Amplification and Estimation."

[115] A. Macaluso, F. Orazi, M. Klusch, S. Lodi, and C. Sartori, "A Variational Algorithm for Quantum Single Layer Perceptron," in *Machine Learning, Optimization, and Data Science*. Springer Nature Switzerland, pp. 341–356.

[116] A. Macaluso, M. Klusch, S. Lodi, and C. Sartori, "MAQA: A quantum framework for supervised learning," vol. 22, no. 3.

[117] M.-C. Yang, J.-L. Li, and C.-F. Qiao, "The decompositions of Werner and isotropic states," vol. 20, no. 8.

[118] S. Sim, P. D. Johnson, and A. Aspuru-Guzik, "Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms," vol. 2, no. 12. [Online]. Available: http://dx.doi.org/10.1002/qute.201900070

[119] Z. Holmes, K. Sharma, M. Cerezo, and P. J. Coles, "Connecting ansatz expressibility to gradient magnitudes and barren plateaus," vol. 3, no. 1, p. 010313. [Online]. Available: https://link.aps.org/doi/10.1103/PRXQuantum.3.010313

[120] D. G. Utkarsh Azad, "PennyLane spin datasets." [Online]. Available: https://pennylane.ai/datasets/ising-model

[121] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, "Barren plateaus in quantum neural network training landscapes," vol. 9, no. 1. [Online]. Available: https://www.nature.com/articles/s41467-018-07090-4

[122] L. Brasco, "Lipschitz functions," in *UNITEXT*. Springer Nature Switzerland, pp. 261–287.

[123] B. T. Kiani, G. D. Palma, M. Marvian, Z.-W. Liu, and S. Lloyd, "Learning quantum data with the quantum earth mover's distance," vol. 7. [Online]. Available: https://api.semanticscholar.org/CorpusID:248811263

[124] F. Glover, G. Kochenberger, and Y. Du. A Tutorial on Formulating and Using QUBO Models. [Online]. Available: http://arxiv.org/abs/1811.11538

[125] G. J. Oyewole and G. A. Thopil, "Data clustering: Application and trends," vol. 56, no. 7, pp. 6439–6475. [Online]. Available: https://link.springer.com/10.1007/s10462-022-10325-y

[126] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-Means Clustering Algorithm," vol. 28, no. 1, p. 100. [Online]. Available: https://www.jstor.org/stable/10.2307/2346830?origin=crossref

[127] G. Hamerly and C. Elkan, "Alternatives to the k-means algorithm that find better clusterings," in *Proceedings of the Eleventh International Conference on Information and Knowledge Management*. ACM, pp. 600–607. [Online]. Available: https://dl.acm.org/doi/10.1145/584792.584890

[128] A. Verma, M. Lewis, and G. Kochenberger, "Efficient QUBO transformation for Higher Degree Pseudo Boolean Functions," arXiv. [Online]. Available: http://arxiv.org/abs/2107.11695

[129] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *ACM-SIAM Symposium on Discrete Algorithms*. [Online]. Available: https://api.semanticscholar.org/CorpusID:1782131

[130] C. Elkan, "Using the triangle inequality to accelerate k-means," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ser. ICML'03. AAAI Press, pp. 147–153.

[131] A. Papavasiliou, S. S. Oren, and B. Rountree, "Applying High Performance Computing to Transmission-Constrained Stochastic Unit Commitment for Renewable Energy Integration," vol. 30, no. 3, pp. 1109–1120. [Online]. Available: http://ieeexplore.ieee.org/document/6872597/

[132] R. Mahroo and A. Kargarian, "Hybrid Quantum-Classical Unit Commitment," in *2022 IEEE Texas Power and Energy Conference (TPEC)*, pp. 1–5. [Online]. Available: http://arxiv.org/abs/2201.03701

[133] A. Ajagekar and F. You, "Quantum computing for energy systems optimization: Challenges and opportunities," vol. 179, pp. 76–89. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360544219308254

[134] B. Schölkopf, "Learning with kernels: Support vector machines, regularization, optimization, and beyond," MIT Press.

[135] W. H. Press, Ed., *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge University Press.

[136] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," vol. 2, no. 2, pp. 121–167.

[137] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A Practical Guide to Support Vector Classification." [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/papers.html

[138] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, and D. Preda, "A Quantum Adiabatic Evolution Algorithm Applied to Random Instances of an NP-Complete Problem," vol. 292, no. 5516, pp. 472–475.

[139] S. Muthukrishnan, T. Albash, and D. A. Lidar, "When Diabatic Trumps Adiabatic in Quantum Optimization."

[140] E. Crosson, E. Farhi, C. Y.-Y. Lin, H.-H. Lin, and P. Shor, "Different Strategies for Optimization Using the Quantum Adiabatic Algorithm."

[141] P. Cortes, J. Larrañeta, and L. Onieva, "A Genetic Algorithm for Controlling Elevator Group Systems," in *Artificial Neural Nets Problem Solving Methods*. Springer Berlin Heidelberg, pp. 313–320.

[142] M. Schuld and N. Killoran, "Quantum machine learning in feature Hilbert spaces," vol. 122, no. 4, p. 040504.

[143] D-Waves, "D-Waves ocean documentation." [Online]. Available: https://www.dwavesys.com/learn/featured-applications/

[144] H. Abraham and e. al, "Qiskit: An Open-source Framework for Quantum Computing."

[145] H.-C. Kim, S. Pang, H.-M. Je, D. Kim, and S. Y. Bang, "Constructing support vector machine ensemble," vol. 36, no. 12, pp. 2757–2767. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320303001754

[146] Y. LeCun, C. Cortes, and g.-i. family=Burges, given=CJ, "MNIST handwritten digit database," vol. 2.

[147] K. Pearson, "On lines and planes of closest fit to system of points in space, Philos," pp. 33–41.

[148] H.-Y. Huang, M. Broughton, M. Mohseni, R. Babbush, S. Boixo, H. Neven, and J. R. McClean, "Power of data in quantum machine learning," vol. 12, no. 1. [Online]. Available: http://dx.doi.org/10.1038/s41467-021-22539-9

[149] A. Canatar, E. Peters, C. Pehlevan, S. M. Wild, and R. Shaydulin, "Bandwidth Enables Generalization in Quantum Kernel Models," arXiv. [Online]. Available: https://arxiv.org/abs/2206.06686

[150] R. Shaydulin and S. M. Wild, "Importance of Kernel Bandwidth in Quantum Machine Learning." [Online]. Available: https://arxiv.org/abs/2111.05451

[151] A. Mott, J. Job, J.-R. Vlimant, D. Lidar, and M. Spiropulu, "Solving a higgs optimization problem with quantum annealing for machine learning," Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/nature24047

[152] R. Y. Li, R. Di Felice, R. Rohs, and D. A. Lidar, "Quantum annealing versus classical machine learning applied to a simplified computational biology problem," vol. 4, no. 1. [Online]. Available: http://dx.doi.org/10.1038/s41534-018-0060-8

[153] S. Thanasilp, S. Wang, M. Cerezo, and Z. Holmes, "Exponential concentration and untrainability in quantum kernel methods."

[154] C.-Y. Liu and H.-S. Goan, "Hybrid Gate-Based and Annealing Quantum Computing for Large-Size Ising Problems."