UNIVERSITÀ DEGLI STUDI DI MILANO

Department of Physics

PhD Programme in Physics, Astrophysics and Applied Physics
Cycle XXXVIII

Towards Acceleration of Quantum-Classical Orchestration

Disciplinary Scientific Sector FIS/02

Supervisor of the Thesis: Stefano Carrazza

Coordinator of the PhD Programme: Prof. Aniello Mennella

PhD Thesis of:

Matteo Robbiati
Registration number: R13832
ORCID: 0009-0005-8605-234X

A. Y. 2024-2025

Commission of the final examination:

External Referee: Emanuele Angelo Bagnaschi
External Referee: Angelo Bassi

External Member: German Sierra
External Member: Fabio Maltoni
External Member: Angelo Nucciotti

External supervision:

External supervisor: Sofia Vallecorsa
External co-supervisor: Michele Grossi

Final examination:

Date 24 November 2025

Università degli Studi di Milano, Dipartimento di Fisica, Milano, Italy

Cover illustration:

The classical-quantum orchestra. Image generated with DALL-E 3.

MIUR subjects:

FIS/02 - Fisica Teorica, Modelli e Metodi Matematici

PACS:

03.67.Lx, 07.05.Hd, 07.05.Bx, 89.20.Ff

Keywords:

Quantum Computing, Hybrid Computing, High Performance Computing, Quantum Algorithms, Quantum Machine Learning

To my family, and to everyone
who made me feel at home

Abstract

Quantum computing is expected to act as a hardware accelerator within future computing infrastructures, complementing classical processors in the same way GPUs accelerate specific workloads today. While algorithms such as Shor's and Grover's suggest potential advantages, the practical use of quantum devices will occur within hybrid architectures where classical and quantum resources coexist. Artificial intelligence illustrates this balance: classical methods, such as Transformers and their scalable variants, remain dominant, while quantum computing may only contribute to selected subroutines. At the same time, AI methods are increasingly applied to quantum computing, supporting calibration, control, and error mitigation of near-term devices. This two-way relationship underlines the need for hybrid solutions and for efficient orchestration of heterogeneous resources.

This thesis investigates the design and implementation of hybrid classical-quantum solutions, with emphasis on their real-time orchestration within heterogeneous infrastructures.

The work has been carried out within the `Qibo` environment, a fully open-source framework that spans the entire quantum computing stack: from high-level interfaces for algorithm design, to efficient simulation backends on classical hardware, to integration with quantum devices down to the hardware and pulse level on self-hosted systems. This modularity makes `Qibo` a suitable platform to explore orchestration strategies across classical and quantum resources.

Within this framework, we introduce algorithmic and software contributions that accelerate hybrid workloads. We present general-purpose open-source libraries, such as `Qiboml` for hybrid quantum-classical machine learning, and `mpstab` for a hybrid stabilizer-tensor network representation of quantum states. By preserving the familiar interfaces of classical machine learning frameworks, `Qiboml` enables the design of hybrid architectures and their execution on both simulators and quantum hardware.

We also develop algorithmic solutions for real-time quantum error mitigation, implemented in `Qiboml` and relying on `Qibo`. We further discuss applications, including multi-variable integration, probability density estimation, and ground-state search, as well as an application in which quantum systems operate as precision sensors for fundamental physics. These serve as test cases to validate the proposed approaches and to provide practical examples of how the heterogeneous components of a future quantum-classical infrastructure can be effectively orchestrated.

Contents

List of Figures

List of Tables

Introduction

Quantum computing will not replace classical computing, but it will become one component of a heterogeneous computational landscape. Modern computational infrastructures are already heterogeneous, and their evolution points toward even more complex orchestration of diverse resources. Each processor type has developed around its own natural strengths. Central processing units (CPUs) remain the backbone of general-purpose computing, flexible and well suited for sequential or irregular workloads. Graphics processing units (GPUs), with their massively parallel structure, excel in graphics rendering, dense linear algebra, and the large-scale matrix operations that underpin modern machine learning. Field-programmable gate arrays (FPGAs) occupy a different niche, offering reconfigurable acceleration for tasks where low latency and energy efficiency are decisive, such as signal processing and real-time control. Artificial intelligence accelerators, such as Google's tensor processing units (TPUs), are specialized devices designed to maximize the performance of tensor operations, sustaining the training and inference of modern large-scale language models.

Quantum processing units (QPUs) add to this picture a completely different competence, embedded in their very different nature. Among others, they promise speedups for problems such as integer factorization [16], unstructured search [17], the solution of linear systems with the Harrow-Hassidim-Lloyd (HHL) algorithm [18], and the simulation of many-body quantum systems [19, 20], a task considered one of the most natural applications of quantum devices.

The future of computation is therefore not defined by replacement but by integration. Tasks such as rendering graphics or training language models will remain firmly classical. Other problems, particularly in chemistry and physics, may see quantum subroutines playing a central role, for example in the calculation of ground states or reaction pathways [21, 22, 23]. Between these extremes lies a wide territory where the two domains must interact. The challenge is twofold: first, to map problems to the resources best suited for them, and second, to make those resources work together with minimal overhead. This second point, the orchestration of heterogeneous computing elements, is the focus of this thesis.

Quantum machine learning is a good example of both promise and reassessment. When it was first introduced, it was often presented as a field where quantum devices would soon provide significant advantages over classical models. Experience has shown that the reality is more subtle, as challenges such as trainability, noise, and scalability have tempered the initial optimism. Classical machine learning, especially with architectures such as Transformers, has proven extraordinarily effective, and near-term quantum models cannot compete at this level. This does not make QML irrelevant: it shifts the

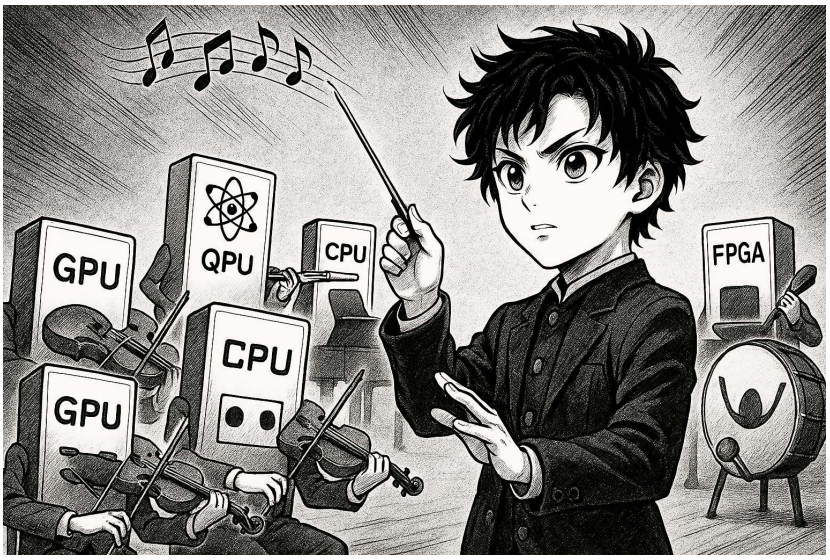


Figure 1: Illustration of orchestration in hybrid quantum-classical computing. Image generated with DALL-E 3.

perspective toward niches where quantum contributions may eventually make sense. In chemistry, for instance, quantum models are expected to encode correlations more naturally than their classical counterparts, potentially providing tools for molecular simulation and electronic-structure problems. The guiding idea in QML is not that quantum resources fail, but that their advantages may arise only when their competences align with the problem, a possibility still under active exploration.

The relationship between artificial intelligence and quantum computing is also reciprocal. AI is already helping quantum technologies, supporting the calibration of devices [24, 25, 26], the characterization of noise [27, 28], and the design of mitigation protocols [29] among others. Machine learning has been applied to circuit compilation [30, 31, 32] and transpilation [33], framing them as optimization tasks, and reinforcement learning has been used to guide adaptive calibration and pulse tuning [34, 35]. These contributions are practical: they help noisy intermediate-scale quantum devices deliver usable results, and they illustrate the importance of classical-quantum cooperation in real infrastructures.

If each computing element has its role, then what remains is to provide the interfaces and abstractions that make them work together. The community as a whole is working to identify the division of competences among classical and quantum devices. This thesis addresses the complementary challenge: to develop open-source software that implements orchestration across the full stack, from algorithms to hardware, and makes hybrid workflows practical. It is also important to recognize that, as will be discussed in Chapter 4, classical computers can sometimes be surprisingly effective in simulating quantum systems, under certain constraints. If either large-scale entanglement or a high number of non-Clifford T gates is absent, then efficient classical simulation is possible, as shown by stabilizer-based methods [36], tensor-network approaches [37], or Clifford+T simulation schemes [38]. These two aspects, entanglement and T-gate complexity, summarize the fundamental obstacles for classical simulation: the first is central in the noise-intermediate scale quantum (NISQ) era [20], the second in the fault-tolerant

regime [39].

The work has been carried out in the context of the `Qibo` ecosystem, a family of fully open-source projects that provide an operating system for quantum computing [3, 2, 40, 6, 7, 41, 42]. `Qibo` itself offers a high-level interface for algorithm design, efficient backends for CPUs and GPUs, and integration with quantum hardware, down to pulse-level control on self-hosted devices. `Qibolab` extends this by connecting to laboratory instruments and enabling control of quantum experiments. `Qibocal` adds systematic procedures for device characterization and calibration. On top of this stack, `Qiboml` introduces hybrid quantum-classical machine learning, including routines for real-time calibration and error mitigation that rely on `Qibo`, while preserving the familiar interface of classical machine learning frameworks. Together, these contributions make orchestration across heterogeneous resources possible, while remaining fully open source.

The thesis is structured to reflect this progression.

Part I. The first part introduces fundamental concepts for the understanding of the thesis, assuming the reader is familiar with basic notions of digital quantum computing, such as qubits, quantum gates, and circuits. Appendix A provides a more detailed introduction to these concepts, for readers less familiar with the field.

After a first historical introduction, Chapter 1 presents the basic concepts of digital quantum computing, and discuss some of the most pressing challenges of the field, both from a technological and a computational perspective.

Chapter 2 presents quantum machine learning, highlighting its connections to classical methods and examining some of its most pressing challenges.

Finally, in Chapter 3, a more realistic picture is painted, introducing one of the most common hardware implementations, superconducting qubits, and the main sources of noise that affect current quantum devices. Consequently to the discussion on noise, quantum error mitigation techniques are introduced, which are essential to obtain useful results from near-term quantum devices.

Part II. The second part presents the `Qibo` ecosystem as a full-stack operating system for quantum computing. It starts with Chapter 4, where `Qibo` itself [3] is introduced as a framework that allows the design and execution of quantum algorithms with backends ranging from CPUs and GPUs to specialized accelerators. In the same chapter also other classical simulation methods are discussed, including tensor networks and stabilizer circuits.

The discussion then moves, with Chapter 5, to `Qibolab` [6], which connects the software layer to experimental hardware, providing open-source control of laboratory instruments and pulse-level access to self-hosted devices. This marks the beginning of the original contributions of this thesis. From this point on, starting with `Qibolab` and including the following chapters, I present frameworks and methods to which I directly contributed.

Part II concludes with Chapter 6 introducing `Qibocal` [7], which systematizes characterization and calibration procedures, ensuring that hardware can be operated reliably. Together, these chapters show how open-source software can span the entire quantum computing stack.

Part III. The third part is devoted to orchestration, that is, to the interaction between classical and quantum resources. Chapter 7 introduces `Qiboml`, which integrates ma-

chine learning into the quantum software stack and provides routines for real-time calibration and error mitigation.

Chapter 9 then presents `mpstab`, a hybrid stabilizer-tensor network representation that extends the scope of classical simulation. This part also includes, with Chapter 8, the development of a real-time quantum error mitigation (RTQEM) [9] protocol, which provides practical techniques to improve the quality of computations on near-term devices. The emphasis here is on building interfaces and methods that reduce the overhead of hybrid computation.

Part IV. The fourth part applies these tools to concrete problems that act as test cases for the stack. It includes studies of multi-variable integration in Chapter 10 [11], and probability density estimation tasks in Chapter 11 [12]. Finally, Chapter 12 presents work on ground-state approximation, focusing in particular on double-bracket quantum algorithms (DBQAs) [14], which offer an alternative to standard variational methods. These applications both validate the developed tools and demonstrate the potential of hybrid infrastructures in realistic scenarios.

Part V. The final part looks outward, considering the role of quantum devices as measurement tools and their possible applications beyond computation strictly defined.

Chapter 13 presents `kifit` [15], a framework for new physics search through isotope shifts data.

Taken together, the chapters follow a coherent trajectory. They begin with theory and background, establish an open-source full-stack framework, build orchestration tools, and apply them in practice. Throughout, the objective is to understand the division of competences among computational resources and to provide software solutions that make their orchestration effective in realistic infrastructures.

Part I
Fundamentals

CHAPTER 1

Introduction to Quantum Computing

This chapter offers an introduction to the fundamental concepts needed to understand the material presented in the following chapters.

After a brief historical overview of quantum computing in Section 1.1.1, we examine the main technological challenges and paradigms that shape the field.

The goal of this chapter is not to provide a complete review of quantum computing, but rather to supply the background and context required for the research discussed in this thesis. Readers new to the subject may find it helpful to consult the appendices dedicated to the basics of quantum computing. The appendices are designed to be self-contained and pedagogical, often including examples and practical implementations based on the `Qibo` framework [3]. Specifically, Appendix A introduces the core ideas of digital quantum computing, Appendix B reviews some key quantum algorithms built on the query model, Appendix C presents the principles of adiabatic quantum computing, an alternative model based on continuous adiabatic evolution rather than discrete unitary gates, and Appendix D discusses Hamiltonian simulation, a central task concerned with reproducing the dynamics of quantum systems governed by a Hamiltonian.

More experienced readers may choose to skip the appendices, though we encourage them to browse these sections if they are interested in a more hands-on perspective on some of the ideas presented here.

For additional background, we recommend standard textbooks on the subject, such as Nielsen and Chuang's *Quantum Computation and Quantum Information* [1]. Readers with a strong computer science background may also appreciate Mermin's *Quantum Computer Science: An Introduction* [43].

1.1 The Birth of Quantum Computing

1.1.1 Historical Context

Almost a century ago, from October 24th to 27th, 1927, the fifth Solvay conference witnessed one of the most significant scientific debates in history [44]. Around the table, prominent scientists including Albert Einstein, Niels Bohr, Werner Heisenberg, Marie Curie, Max Planck, Paul Dirac, Erwin Schrödinger, and Louis de Broglie engaged in discussions about the foundations of quantum mechanics, a theory that would fundamentally transform our understanding of the physical world. During these discussions,

although the participants could not have foreseen the technological implications we observe today, many of the core concepts that underpin quantum computing and this thesis were debated: *(i) superposition*, *(ii) entanglement*, *(iii) the uncertainty principle*, and *(iv) wavefunction collapse*, among others.

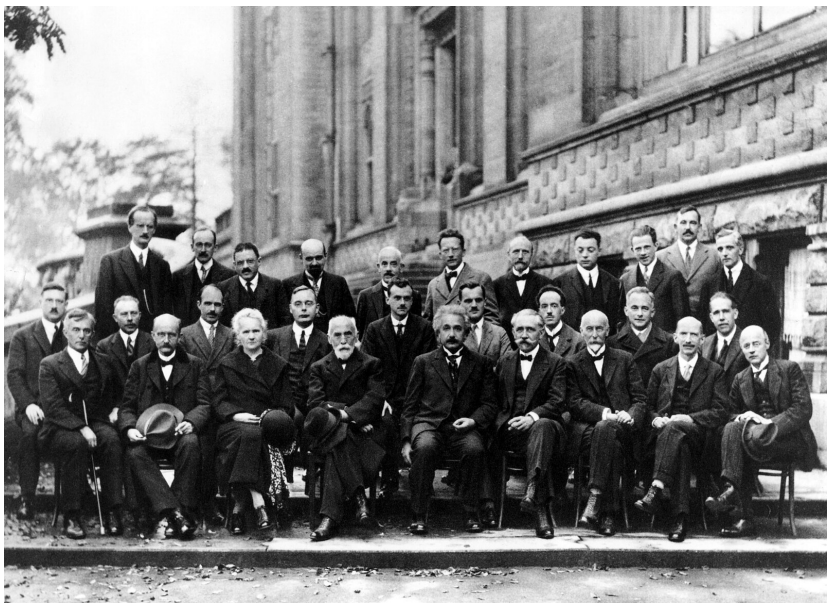


Figure 1.1: The fifth Solvay conference's participants. The photograph was taken by Benjamin Couprie during the event in 1927.

The profound questions raised during this conference catalyzed extensive research activity in the subsequent decades. These discussions later led to more concrete proposals for quantum computation. A pivotal moment came in the early eighties, when first Paul Benioff proposed a quantum mechanical model of Turing machine in 1980 [45] and then, in 1982, Richard Feynman articulated the idea of a quantum computer: a device that could exploit quantum phenomena to perform computations that would be impractical for classical machines [19]. Feynman's central insight was that quantum systems could be employed to simulate other quantum systems more efficiently than their classical counterparts, thereby opening new computational possibilities that had previously been beyond reach.

The idea of using quantum systems to process information began to be explored and was given a more theoretical footing in 1985, when David Deutsch proposed the mathematical concept of a *quantum Turing machine* to model quantum computation.

One year earlier, in 1984, the first fundamental application of quantum mechanics to information processing was proposed. Charles Bennett and Gilles Brassard introduced the concept of quantum key distribution, a method for secure communication that leverages the principles of quantum mechanics to ensure the security of transmitted information [46]. This idea was practically implemented in 1989, marking a significant milestone in the field [47]. A second application, quantum teleportation, was proposed in 1993 by Charles Bennett, Claude Crépeau, Richard Jozsa, Asher Peres, and William Wootters [48]. Quantum teleportation allows the transfer of quantum states between dis-

tant locations without physically moving the particles themselves, relying on the phenomenon of entanglement to achieve this feat. This concept was then experimentally demonstrated in 1997 by a team led by Anton Zeilinger [49].

Focusing on the theoretical aspects of quantum computing, the field required concrete examples of algorithms demonstrating how the model of a quantum computer, formalized by Deutsch in 1985, could be used to solve tasks more efficiently than any other computational paradigm. Such results would strengthen its position within the broader framework of information theory.

Promptly, in 1992, Deutsch himself, together with Richard Jozsa, proposed the first quantum algorithm with provable speedup over classical algorithms, which is indeed called the Deutsch-Jozsa algorithm [50]. This algorithm determines whether a Boolean function is constant or balanced with a single quantum query, whereas any classical algorithm requires exponentially many queries in the worst case.

The theoretical foundations were further strengthened in 1993 when Umesh Vazirani and Ethan Bernstein introduced the Bernstein-Vazirani algorithm [51], which efficiently finds a hidden bit string using a single quantum query, compared to n classical queries for an n-bit string.

In 1994, Daniel Simon introduced an algorithm [52] that solved the hidden subgroup problem for abelian groups exponentially faster than any known classical method. This result showed a clear quantum advantage for a structured problem and gave important insights that influenced later work on quantum algorithms.

The same year was a turning point for quantum computing when Peter Shor developed his factoring algorithm [16]. Shor's algorithm can factor large integers exponentially faster than the best-known classical algorithms, directly threatening the security of widely-used cryptographic systems such as RSA. This result demonstrated that quantum computers could have profound real-world implications, particularly for cryptography and cybersecurity.

In 1996, Lov Grover introduced another fundamental quantum algorithm [53] for searching unsorted databases. Grover's algorithm provides a quadratic speedup over classical search algorithms, finding a marked item in an unsorted database of N elements in approximately $\sqrt{N}$ steps, compared to the classical requirement of $N/2$ steps on average. This algorithm has broad applications beyond database search, including optimization problems and amplitude amplification techniques.

Building on Feynman's original vision, Seth Lloyd formalized the concept of universal quantum simulators in 1996 [54], providing a rigorous theoretical framework for Feynman's intuition that quantum computers could efficiently simulate arbitrary quantum systems. Lloyd's work demonstrated that a universal quantum simulator could be programmed to simulate any quantum system, establishing the theoretical foundation for one of the most promising applications of quantum computing.

The aforementioned algorithms and theoretical advancements set the stage for the development of quantum computing as a field. They not only provided concrete examples of quantum speedup but also established the theoretical framework for understanding quantum computation. These early breakthroughs paved the way for subsequent new research directions, sometimes born from the interplay with other emergent fields, as happened with quantum machine learning, which will be often the theoretical playground explored in this thesis.

1.1.2 A Technological Challenge

The theoretical elegance of quantum algorithms stands in contrast to the formidable challenges of building practical quantum computers. In 2000, David DiVincenzo outlined five essential criteria that any physical system must satisfy to serve as a quantum computer [55]. These criteria, known as the DiVincenzo criteria, highlight the fundamental difficulties in quantum computing implementation:

1. *Scalable qubit systems:* The physical system must support a scalable number of well-characterized qubits. Each qubit must be reliably prepared, controlled, and measured while maintaining its quantum properties.

2. *Qubit initialization:* There must be a reliable method to initialize the qubits to a known state, typically the computational ground state $|0\rangle$.

3. *Universal quantum gates:* The system must allow for the implementation of a universal set of quantum gates[1] with sufficiently low error rates to enable fault-tolerant computation [39]; that is, computation in which errors are actively detected and corrected so the result remains reliable.

4. *Long coherence times:* The quantum states must remain coherent for times much longer than the gate operation times, allowing for meaningful computation before decoherence destroys the quantum information.

5. *Efficient readout:* There must be a reliable method to measure the final quantum states with high fidelity.

Each of these criteria presents significant technical challenges. Quantum systems are inherently fragile, with quantum states being easily disturbed by environmental noise, electromagnetic fields, temperature fluctuations, and even cosmic rays. The requirement for extremely low error rates, typically below 10^{-3} per gate operation for meaningful computation, demands unprecedented precision in control and isolation from the environment.

These challenges have spawned entire research fields dedicated to overcoming the limitations of current quantum hardware. *Quantum error correction* emerged as a theoretical framework for protecting quantum information by encoding logical qubits into multiple physical qubits, allowing for the detection and correction of errors without destroying the quantum state [56]. However, implementing quantum error correction requires thousands of physical qubits per logical qubit, presenting a significant scalability challenge. In parallel, *quantum error mitigation* techniques have been developed to extract meaningful results from noisy quantum devices without full error correction [57]. These methods include techniques such as error extrapolation, symmetry verification, and probabilistic error cancellation, which can improve the reliability of quantum computations on near-term devices. Section 3.3.2 will provide a more detailed discussion of some specific approaches to quantum error mitigation.

The current technological landscape is characterized by two distinct paradigms.

Noisy Intermediate-Scale Quantum (NISQ) devices represent the present state of quantum computing technology [20]. These systems typically contain tens to hundreds of qubits but lack the error correction capabilities necessary for running large-scale quantum algorithms reliably. NISQ devices are limited by noise and decoherence, restricting

[1] We will see, later in the chapter, that we commonly refer to operations on the quantum bits as *gates*.

them to relatively shallow quantum circuits and specialized algorithms designed to be resilient to errors.

The ultimate goal remains the development of *fault-tolerant quantum computers*, which will incorporate sophisticated quantum error correction schemes to protect quantum information indefinitely [39]. These systems will require millions of physical qubits to implement thousands of logical qubits, representing a fundamental scaling challenge that defines much of current quantum computing research. The transition from NISQ to fault-tolerant quantum computing represents one of the most significant technological challenges of our time, requiring advances in materials science, control electronics, and algorithmic design.

Although the prospect of fully fault-tolerant quantum devices is still distant, significant progress can be achieved by focusing on the capabilities of current hardware. In this context, research efforts concentrate on quantum error mitigation techniques, as well as on the development of robust and reliable methods to control quantum processors. Equally important is the design of architectures that integrate quantum computers into larger hybrid infrastructures, where classical and quantum resources work together. These directions are essential to bridge the gap between current noisy devices and future large-scale quantum computers. This thesis belongs to this research area.

1.2 Quantum Computing Challenges

When considering quantum computing, we can distinguish between two main paradigms: *digital quantum computing* and *analog quantum computing*. Digital quantum computing is the most widely known and studied approach, and it serves as the main focus of this thesis. A detailed introduction to its building blocks is provided in Appendix A, while analog quantum computing is briefly discussed in Appendix C.

In what follows, we use the digital paradigm as a concrete example of how quantum algorithms are described and executed. It offers a precise and structured model that allows us to identify challenges, quantify costs, and motivate the need for quantum computers. By working within this framework, we will progressively connect physical limitations, algorithmic constraints, and computational complexity.

Digital quantum computers operate by manipulating qubits through discrete quantum gates, which form quantum circuits. Qubits are two-level quantum systems that can exist in superpositions of their basis states. As shown in Equation 1.5 of Appendix A, the size of the Hilbert space grows exponentially with the number of qubits: representing an n-qubit state requires 2^n complex amplitudes. Quantum circuits provide a convenient way to structure quantum algorithms as sequences of elementary operations. Single-qubit gates manipulate individual states, while multi-qubit gates create and distribute entanglement.

Although multi-qubit gates acting on more than two qubits exist in principle, they are almost always decomposed into sequences of single-qubit and two-qubit operations [58, 1]. This approach is essential in practice: two-qubit gates already represent the main source of hardware errors and are often the primary bottleneck for circuit depth. Implementing gates acting on more than two qubits would introduce further complexity and additional error channels.

For these reasons, understanding how decompositions behave plays a central role in assessing the cost of quantum algorithms.

All these features illustrate the expressive power of digital quantum computing. At the same time, they highlight substantial challenges that must be addressed before large-

scale, reliable quantum computation becomes feasible. We begin by reviewing these challenges from both a hardware and an algorithmic perspective.

1.2.1 Practical Challenges in Digital Quantum Computing

The practical constraints of digital quantum computing arise from two tightly linked aspects: *i)* the physical behavior and limitations of current hardware platforms, and *ii)* the algorithmic and compilation difficulties involved in turning mathematical circuits into instructions that real devices can execute. This close interplay defines the entire quantum software and hardware stack, and many of the tools presented in this thesis were developed precisely to bridge these layers.

Hardware and Engineering Challenges

Constructing a reliable quantum processor requires extraordinary precision and control. Qubits are extremely sensitive to environmental disturbances, and even small imperfections in fabrication or control can degrade performance. Some of the most important physical obstacles are the following.

Noise, coherence, and gate errors. Quantum states decohere due to unavoidable interactions with the environment. In superconducting qubit platforms, coherence times typically range from a few tens to a few hundreds of microseconds. Within this short time window, a circuit must complete all operations before quantum information is lost. Moreover, quantum gates are imperfect: single-qubit gates can achieve error rates near 10^{-4} or below, but two-qubit gates, which are essential for entanglement, are significantly noisier and often represent the dominant source of errors. As circuits grow deeper, these errors accumulate and eventually corrupt the computation.

Device topology and limited connectivity. The geometry of a quantum device strongly constrains which qubits can interact directly. In superconducting processors, qubits are typically arranged on a two-dimensional grid with fixed coupling patterns. If a circuit requires an entangling gate between two qubits that are not directly connected, a sequence of SWAP gates must be used to move quantum information across the chip. This routing overhead increases circuit depth and, consequently, exposes the computation to additional noise. Connectivity therefore plays a critical role in determining the effective cost of a circuit.

Stability, calibration, and scaling. Qubit parameters such as frequency, anharmonicity, and coupling strength, which will be discussed in greater detail in Chapter 3, are not perfectly stable. They drift on time scales ranging from hours to minutes, making regular calibration essential. Crosstalk between neighboring qubits introduces further complexity, as driving one qubit can unintentionally influence another.

Scaling these systems to hundreds or thousands of qubits demands major advances in cryogenic engineering, microwave routing, on-chip control electronics, and fabrication techniques. At present, these engineering challenges limit the size of practical quantum processors.

Algorithmic and Compilation Challenges

Even if quantum hardware were ideal, implementing a quantum algorithm would still require solving difficult algorithmic and compilation problems. In practice, these are inseparable from hardware considerations.

Gate decomposition and native operations. Quantum devices support a limited set of native gates. Arbitrary operations must therefore be decomposed into sequences of native gates. For example, many hardware platforms implement controlled-NOT operations natively, but others require constructing them from more elementary interactions. These decompositions influence circuit depth, which in turn affects both noise accumulation and simulation cost.

Compilation, routing, and SWAP optimization. Mapping an ideal circuit onto a physical device involves assigning logical qubits to physical qubits[2], scheduling operations so that hardware constraints are respected, and inserting SWAP gates whenever direct interactions are not available. This task is computationally difficult and often contributes substantially to the effective cost of a quantum algorithm.

For these reasons, efficient compilation techniques, together with noise-aware routing strategies, are essential components of the quantum software stack.

These algorithmic considerations play a central role in understanding the practical cost of executing quantum algorithms and will be explored in greater detail in Section 1.2.2.

A Bridge to Computational Complexity

The challenges outlined so far may suggest a pessimistic outlook on digital quantum computing. However, the situation is more nuanced, and two important points balance this perspective. First, from a technological perspective, these challenges create a fertile environment for innovation, especially within hybrid infrastructures where classical and quantum resources cooperate to overcome hardware limitations. Much of the software solutions presented in this thesis (`Qibolab`, `Qibocal`, `Qibotn`, `Qiboml`) is motivated by these needs.

Second, and more importantly, is the observation that classical computers are capable of simulating quantum systems only under certain structural conditions. Understanding these conditions provides a natural bridge to computational complexity. Classical simulation remains efficient mainly in two specific regimes: *i)* when entanglement remains low or grows slowly, as in shallow or geometrically constrained circuits, where tensor-network methods excel; and *ii)* when the number of non-Clifford gates (this quantity is often called *non-stabilizerness*) is small, enabling efficient simulation via stabilizer-based techniques. This discussion will be expanded in Chapter 4, where we review the main classical simulation methods and their limitations.

Quantum computers become necessary precisely when both resources grow simultaneously: high entanglement and large numbers of non-Clifford operations. In this regime, classical approximations break down, and no compression technique is known to avoid the exponential growth in memory and time requirements. When such structure is absent, a classical simulator must store the full 2^n amplitudes of the state vector, which

[2] That is, finding a suitable mapping between the indices used in a theoretical circuit description and the actual qubits laid out on the chip.

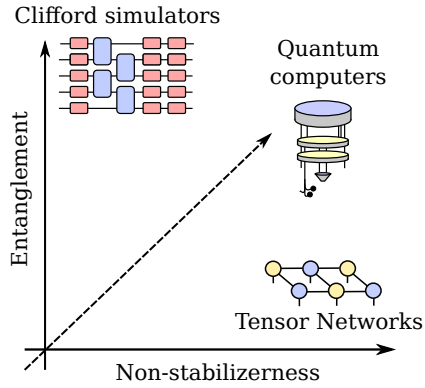


Figure 1.2: Schematic representation of the regimes of classical simulability based on entanglement and non-Clifford resources. Classical simulation is efficient when either entanglement or non-Clifford resources are limited. Quantum computers become necessary when both grow simultaneously.

becomes rapidly infeasible as it will be shown in Section 1.3. A schematic representation of these regimes is provided in Figure 1.2.

This observation motivates not only the interest in quantum devices but also the need to quantify the true cost of digital quantum computation. Before introducing more advanced topics, we therefore present intuitive metrics that characterize the cost of quantum algorithms and discuss the fundamental limits of classical simulation.

1.2.2 The Cost of Digital Quantum Computing, Present and Future

A fundamental aspect of quantum computation is that the cost of an algorithm is typically measured not only in terms of arithmetic operations but, particularly, in terms of quantum gates. The number and type of gates required to implement an algorithm determine its physical feasibility and, in the fault-tolerant setting, its asymptotic scaling.

We will see, in this section, how NISQ applications are limited by the number of two-qubit gates (CZ, CNOT), while fault-tolerant quantum computing is dominated by the cost of T gates.

Single-qubit *rotational gates* (Equation 1.17 in Appendix A) play an important role because they form a universal, continuous family of parametrized operations. However, fault-tolerant quantum computers do not implement arbitrary rotations directly. Instead, they support operations from the *Clifford group*, which preserve the structure of Pauli operators and allow efficient classical tracking. Universality is achieved by adding a single non-Clifford operation, typically the T gate (Section A.5.2). Consequently, T gates become the primary unit of cost for fault-tolerant computation. A more detailed discussion on Clifford gates and non-stabilizerness is provided in Chapter 9.

This fundamental role stems from two limitations: *i)* T gates break the stabilizer structure that enables efficient classical simulation; and *ii)* they cannot be implemented transversally in most error-correcting codes, requiring costly *magic-state distillation* processes whose overhead increases significantly with desired precision [59].

Arbitrary rotations behave analogously: they must be approximated using long fault-tolerant sequences of Clifford and T gates.

On present-day NISQ devices, which operate without error correction, the dominant cost shifts from T gates to *two-qubit gates*. These gates are typically an order of magnitude noisier than single-qubit operations. Moreover, coherence-time limitations constrain the maximum allowed circuit depth, making the reduction of two-qubit gates essential for practical execution. Similar concerns influence classical simulation costs: many simulators treat two-qubit gates as the primary source of entanglement growth.

These principles motivate several goals of this thesis. One example is the `mpstab` package (Chapter 9), which uses stabilizer structure to reduce the number of two-qubit gates in simulations and the tensor network formalism to transform local rotations into matrix product operators. Another is the algorithm presented in Chapter 12, which approximates ground states of Hamiltonians using optimized sequences of Hamiltonian simulation steps designed to minimize two-qubit interactions.

1.3 Can a Classical Computer Simulate a Quantum Computer?

Before moving to more advanced introductory chapters, let us complete the provided picture by asking a fundamental question: to what extent can classical computers simulate quantum computers? And where does this simulability break down? We start with the simplest case: storing and manipulating the full quantum state vector.

Representing the state of n qubits requires 2^n complex amplitudes, each typically stored using 16 bytes in double precision. This quickly leads to an exponential explosion in memory requirements. The following example illustrates this growth concretely.

How many qubits can be represented on a classical computer?

Consider a personal computer with 32 GB of RAM. It can store:

$$\frac{32 \times 2^{30}}{16} \approx 2^{31} \tag{1.1}$$

complex amplitudes, corresponding to the full state of $n \approx 31$ qubits.
Now consider El Capitan, a leading classical supercomputer with about 2 exabytes (2×10^{18} bytes) of memory [60]. It can store:

$$\frac{2 \times 10^{18}}{16} = 1.25 \times 10^{17} \tag{1.2}$$

complex amplitudes, corresponding to $n \approx 56$ qubits.
Even with such enormous resources, simulating a system of only 56 qubits is already at the limit of what is feasible.

This exponential scaling shows why quantum computers are fundamentally different: they can naturally process quantum states that classical machines cannot even store.

However, this picture is not showing all the nuances. Full state-vector simulation is the most complete, but also the most resource-intensive classical simulation technique.

Beyond the full state: structured simulations. Many quantum systems encountered in practice have structure that dramatically reduces simulation cost. If entanglement remains limited, tensor-network approaches such as matrix product states can be used. If a

circuit contains few non-Clifford gates, stabilizer-based techniques simulate it efficiently. These approaches often allow classical computers to simulate far larger systems than the naive 2^n bound would suggest, as discussed in Section 4.

Hybrid strategies and circuit cutting. The presence of these efficient classical regimes motivates hybrid quantum-classical approaches, in which classical simulations handle structured regions of a computation, while quantum devices are used for the complex parts. An additional layer of flexibility is offered by circuit cutting strategies, which decompose a large circuit into smaller pieces that can be simulated on distributed classical infrastructures or executed across multiple quantum processors. Recent work in this direction includes the algorithmic frameworks introduced in Ref. [61].

Overall, understanding when quantum states admit a compact representation and when they require full exponential resources is crucial for appreciating the true computational power of quantum computers. This theme will reappear throughout the thesis, particularly in our discussion of classical simulation techniques and quantum algorithmic optimizations.

The first key message of this section is that simulating quantum systems on a classical computer is feasible only under specific structural conditions. When these conditions are not satisfied, classical simulation quickly becomes intractable, underscoring the unique capabilities of quantum devices. This observation motivates the development of quantum hardware and algorithms designed to operate in regimes beyond classical reach.

A second important point is that one computational paradigm does not replace the other. Instead, classical and quantum computing complement each other, with hybrid approaches drawing on the strengths of both to address complex problems more effectively.

CHAPTER 2

Introduction to Quantum Machine Learning

Quantum machine learning (QML) is an emerging field that combines quantum computing and machine learning with the intention of leveraging quantum systems to enhance the capabilities of machine learning algorithms [62, 63]. Whether an advantage over classical machine learning or, more in general, over classical computing can be achieved is still an open question: in the last years, several works have been investigating the potential of utilizing quantum systems to embed data, map features into more expressive spaces, and perform optimization tasks adopting different approaches.

In what follows, we will introduce the main concepts of quantum machie learning, focusing on a supervised learning setting, where the goal is to learn a function that maps input data to output labels. This can be easily extended to different settings, such as unsupervised learning or reinforcement learning, which will not be discussed since unrelated to the following chapters.

2.1 Supervised Learning

Let's consider a problem where we have a set of input data $\{\boldsymbol{x}_i\}_{i=1}^{M}$. We establish the notation of using boldface letters to denote vectors, while we use the subscript i to denote the i-th element of the set. The goal of a supervised learning algorithm is to learn a function $f(\boldsymbol{x})$ that maps the input data $\boldsymbol{x}$ to a label y, which we consider mono-dimensional for simplicity, but it can be easily extended to multi-dimensional labels. As in classical machine learning, a common approach to learn the function $f(\boldsymbol{x})$ is to consider a parameterized model $m(\boldsymbol{x};\boldsymbol{\theta})$ and optimize the parameters $\boldsymbol{\theta}$ such that m provides a good approximation of the function $\tilde{f}(\boldsymbol{x};\boldsymbol{\theta})$. We denote with tilde the approximated function.

In classical machine learning, a standard choice for the model $m(\boldsymbol{x};\boldsymbol{\theta})$ is a neural network [64], which is a function that can be expressed as a composition of several layers of linear and non-linear transformations. Introducing such complex modeling to tackle learning tasks implies taking into account the ideas of trainability, expressivity, and generalization, which are fundamental concepts in machine learning.

1. **Expressivity** is related to the variety of functions that can be represented by a model. More formally, expressivity characterizes the ability of a model class $\mathcal{M}$ to represent elements of a function space $\mathcal{F}$. More elements of the function space can be represented by a more expressive model class. The extreme case is when the model class is so expressive that it can approximate any function in $\mathcal{F}$ at arbitrary precision. This is known as *universality*.

2. **Trainability** refers to the ability of a model class $\mathcal{M}$ to learn a target function $f \in \mathcal{F}$ with satisfactory accuracy. Trainability characterizes how efficiently and reliably

such representation can be found using finite data and computational resources given an optimization algorithm $\mathcal{A}$. From analytical perspectives, trainability can be studied through the properties of the optimization landscape: the presence of well-conditioned minima, the absence of pathological saddle points or the magnitude and stability of the gradient of the loss function during the training.

3. **Generalization** is the ability of a trained model $m \in \mathcal{M}$ to represent well a target function $f \in \mathcal{F}$ not only on the training data, but also on unseen data, drawn from the same distribution as the training data. If we define $\mathcal{E}_{\text{train}}(m)$ and $\mathcal{E}_{\text{test}}(m)$ respectively as the error of the model m in computing predictions on the training and test data, the generalization error can be defined as

$$\mathcal{E}_{\text{gen}}(m) = |\mathcal{E}_{\text{test}}(m) - \mathcal{E}_{\text{train}}(m)|. \tag{2.1}$$

The three concepts of expressivity, trainability, and generalization are interrelated and play a crucial role in practical machine learning applications. A consequent challenge is to understand which aspects of these concepts are most important for a given task and how they can be balanced effectively. For example, we may be interested in training a model m the best possible way but still ensure that it generalizes well to unseen data. This means avoiding *overfitting*, which occurs when a model starts to learn the statistical fluctuations of the data. A schematic representation of overfitting is shown in left sub-plot of Figure 2.1.

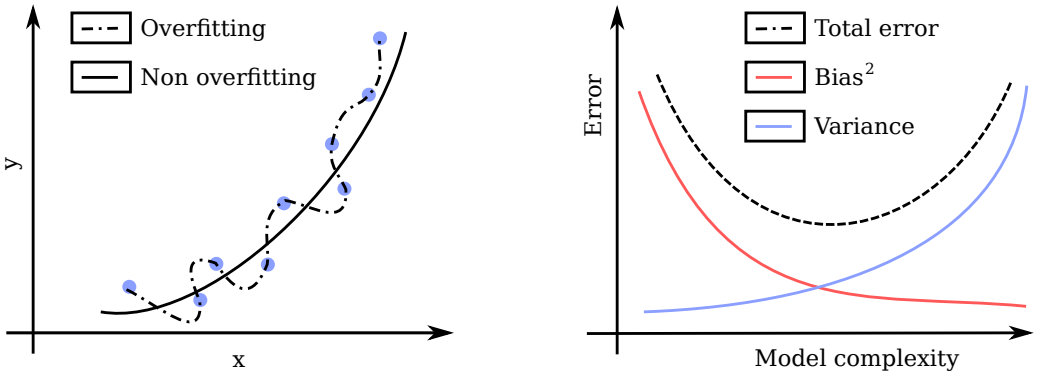


Figure 2.1: Left, schematic representation of the overfitting phenomenon. The dot-dashed line is catching the statistical fluctuations of the data, resulting in a model which is not able to generalize to new data. Right, representation of the bias-variance tradeoff.

A related concept, which can be better formalized, is the *bias–variance tradeoff*, which decomposes the expected error of a model into two complementary sources. Let $f \in \mathcal{F}$ denote the target function, $m \in \mathcal{M}$ a trained model obtained by algorithm $\mathcal{A}$ on a dataset sampled from $\mathcal{D}$, and $\tilde{f}(x) = m(x)$ its prediction. The expected squared error at an input $x \in \mathcal{X}$ can be decomposed as

$$\mathbb{E}_{\mathcal{D},\mathcal{A}}\left[(f(x) - \tilde{f}(x))^2\right] = \underbrace{(f(x) - \mathbb{E}[\tilde{f}(x)])^2}_{\text{Bias}^2} + \underbrace{\mathbb{V}[\tilde{f}(x)]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible noise}},$$

where $\mathbb{E}[\tilde{f}(x)]$ is the average prediction over training sets and initializations, $\mathbb{V}[\tilde{f}(x)]$ is the variance of predictions, and σ^2 denotes the intrinsic noise in the data.

High bias typically arises when $\mathcal{M}$ has low expressivity, leading to underfitting, while high variance often results from overly expressive models that are trainable but sensitive to fluctuations in the training data, leading to overfitting. The bias–variance tradeoff thus highlights the tension between expressivity, trainability, and generalization, and motivates the use of regularization, model selection, and data augmentation to balance these factors.

2.2 Training a Model

To achieve a satisfactory prediction, we have to understand and implement a strategy to train a parametric model. This is usually done by defining a *cost function* $\mathcal{C}(\boldsymbol{\theta})$ that quantifies the difference between the predicted labels $m(\boldsymbol{x}_i;\boldsymbol{\theta})$ and the true labels y_i. A typical choice of cost function in regression problems is the mean-squared error (MSE):

$$\mathcal{C}(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M}(m(\boldsymbol{x}_i;\boldsymbol{\theta}) - y_i)^2, \tag{2.2}$$

but many other choices are possible, depending on the specific task and the nature of the data. For a detailed review of possible cost functions see for example "Training and cost function" section of [65].

Once a cost function is defined, we need an optimization algorithm, to which we refer as optimizer, to minimize it. The goal of the optimizer is to update the parameters of the model in order to find the optimal parameters $\boldsymbol{\theta}^*$ that minimize the cost function:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{C}(\boldsymbol{\theta}). \tag{2.3}$$

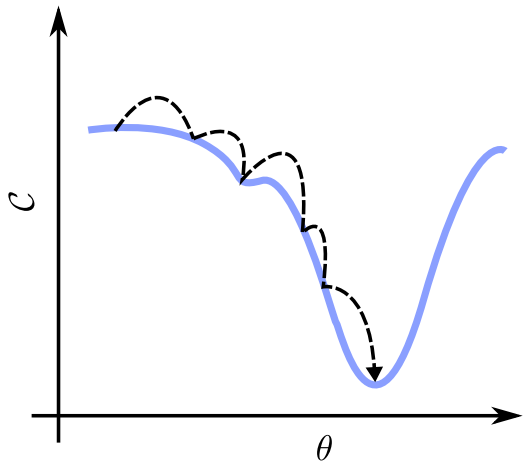


Figure 2.2: Illustration of gradient descent optimization. The blue line represents the cost function, while the black and dashed line represent the updates of the parameters.

A common choice of optimizer in machine learning is gradient descent, which iteratively updates the model parameters in the direction of the negative gradient of the cost function:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\nabla\mathcal{C}(\boldsymbol{\theta}_t), \tag{2.4}$$

where η is the learning rate and t denotes the iteration. An illustrative drawing of a gradient descent optimization is shown in Figure 2.2. For a detailed review of possible optimization algorithms see for example "Faster optimizers" section of [65]. The wide usage of gradient descent in machine learning explains why gradients are so important for studying the trainability properties of models in this context.

2.2.1 Gradients in Classical Machine Learning: Backpropagation

In classical machine learning, the optimization of models $m \in \mathcal{M}$ relies on the efficient computation of gradients of a cost function $\mathcal{C}$ with respect to the model parameters $\theta \in \Theta$. For neural networks, this is achieved through the *backpropagation* algorithm [66], which applies the chain rule of calculus to propagate derivatives backward through the computational graph of the network.

Concretely, consider a neural network represented as a composition of L layers,

$$f(x;\theta) = f_L\big(f_{L-1}(\cdots f_1(x;\theta_1)\cdots;\theta_{L-1});\theta_L\big), \tag{2.5}$$

where $\theta = \{\theta_1, \ldots, \theta_L\}$ are the trainable parameters associated with each layer. The gradient of the cost function $\mathcal{C}(f(x;\theta))$ with respect to a parameter block θ_ℓ can be expressed as

$$\frac{\partial \mathcal{C}}{\partial \theta_\ell} = \frac{\partial \mathcal{C}}{\partial f_L} \cdot \frac{\partial f_L}{\partial f_{L-1}} \cdots \frac{\partial f_{\ell+1}}{\partial f_\ell} \cdot \frac{\partial f_\ell}{\partial \theta_\ell}. \tag{2.6}$$

The efficiency of backpropagation comes from the reuse of *intermediate results* computed during the forward pass. In the forward pass, each layer output $h_\ell = f_\ell(h_{\ell-1};\theta_\ell)$ is stored, where $h_0 = x$. During the backward pass, the algorithm computes the so-called *error signals*

$$\delta_\ell = \frac{\partial \mathcal{C}}{\partial h_\ell},$$

which are obtained recursively using

$$\delta_\ell = \delta_{\ell+1} \cdot \frac{\partial f_{\ell+1}(h_\ell;\theta_{\ell+1})}{\partial h_\ell}.$$

Once δ_ℓ is known, the gradient with respect to the parameters of layer ℓ is given by

$$\frac{\partial \mathcal{C}}{\partial \theta_\ell} = \delta_\ell \cdot \frac{\partial f_\ell(h_{\ell-1};\theta_\ell)}{\partial \theta_\ell}.$$

Thus, each gradient is computed using the stored forward activations $h_{\ell-1}$ and the recursively propagated error signal δ_ℓ. This reuse of intermediate results ensures that the cost of computing all gradients scales only linearly with the number of parameters, rather than exponentially with network depth. This efficiency is one of the main reasons why deep learning has become practically feasible and successful.

In practice, the cost of computing the gradient of the cost function w.r.t the trainable parameters of a neural network is comparable with the cost of a single forward pass of the network itself (the cost of a prediction). We will refer to this cost as *backpropagation cost*.

2.3 From Classical to Quantum Machine Learning

The previous sections have introduced the main ingredients of a machine learning pipeline: *i)* a set of labeled data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{M}$, *ii)* a parameterized model $m(\boldsymbol{x}; \boldsymbol{\theta})$, *iii)* a cost function $\mathcal{C}(\boldsymbol{\theta})$ and *iv)* an optimization algorithm to find the optimal parameters $\boldsymbol{\theta}^*$ minimizing the value of the cost function.

These same components can be interpreted in the context of quantum machine learning, allowing us to explore whether quantum systems can be used to enhance machine learning pipelines or to investigate different approaches to the learning tasks.

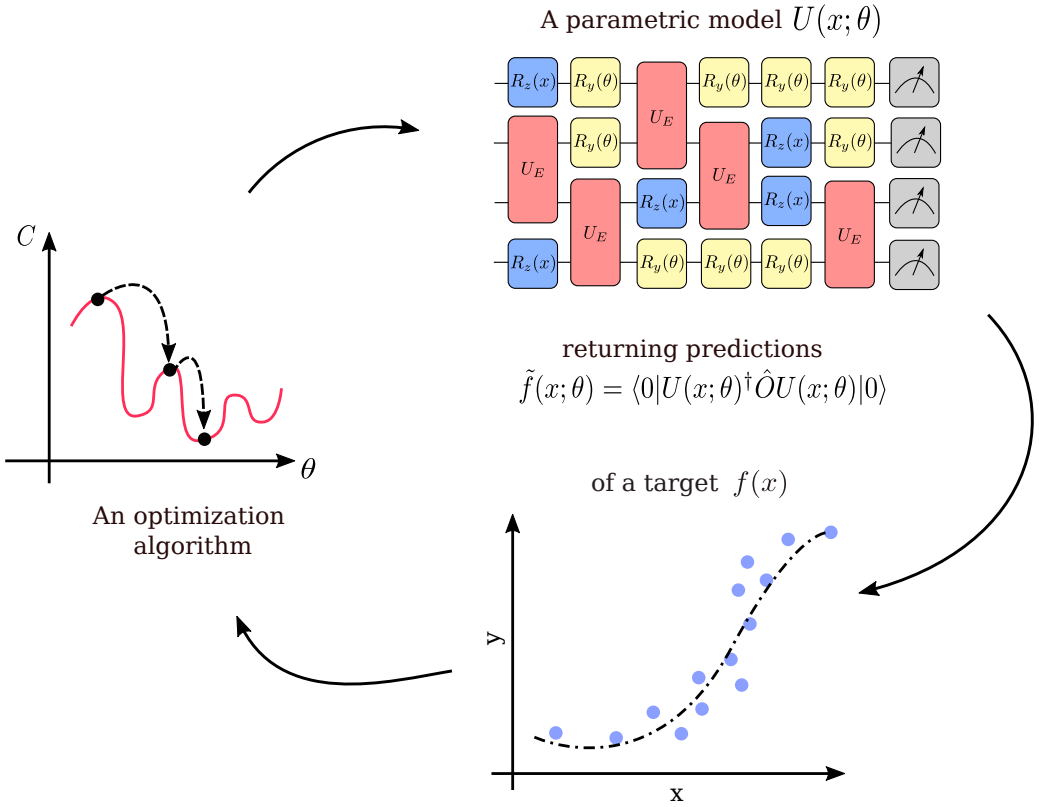


Figure 2.3: Schematic representation of a quantum machine learning pipeline: a parametric circuit is used to approximate a target function and an optimization algorithm is employed to find the optimal parameters of the circuit.

First of all, we can use a parametrized quantum circuit (PQC), $U(\boldsymbol{\theta})$, in place of the parametric model $m(\boldsymbol{x}; \boldsymbol{\theta})$. In fact, as described in Section A.5.1 of Appendix A, there exist parametrized quantum gates that we can use to construct a parametrized circuit. Since we mostly use rotational gates as the ones introduced in Equation 1.17 of Appendix A, we will refer to the trainable parameters of a model as *parameters* and to the parameters of the gates as *angles*. This distinction is important because a single circuit's angle can be constructed as a function of more trainable parameters. For example, we may decide to use a rotation gate $R_z(\alpha) = \exp -i(\alpha/2)\sigma_z$ where alpha is constructed as linear combination of two trainable parameters θ_1 and θ_2: $\alpha = \theta_1 + \theta_2$.

Making a circuit parametric is not the only utility of parametrized gates; in fact, we

can use the same gates to encode data into the state prepared by a quantum circuit. We refer to this encoding strategy as *angle encoding*. If so, when implementing a training procedure, the optimizer will be called to update only the circuit's parameters (which, again, do not correspond to the circuit's angles). When using parametrized gates both for adding trainable parameters and encoding data, we will add $\boldsymbol{x}$ as direct dependence of the unitary operator representing the circuit, *i.e.* $U(\boldsymbol{x};\boldsymbol{\theta})$.

Once an input data $\boldsymbol{x}$ is encoded into the circuit, we need a procedure to compute the corresponding prediction $\tilde{y} = m(\boldsymbol{x};\boldsymbol{\theta}^*)$. This can be achieved by measuring the state of the qubits after executing the quantum circuit. The measurement output can be used to construct an estimation of the target function, for example by computing the expectation value of an observable, as it is done in the illustrative example shown in Figure 2.3. The necessity of processing information in form of expectation values raises two complications: *i)* the necessity of repeating the circuit's execution a certain number of times (or *shots*), and *ii)* the challenge of computing operations in complex Hilbert spaces, which, as will be shown in the next section, introduces one of the biggest obstacles to scalable quantum machine learning applications at the moment.

In summary, what introduced here is one possible way of leveraging the quantum computing formalism to implement machine learning routines. This paves the way for exploring new paradigms in machine learning, where quantum mechanics can be used to rephrase classical concepts and potentially offer advantages in terms of efficiency and performance. On the other hand, playing into a very expressive and complex space also introduces challenges in terms of interpretability and practicality. In the following sections we will introduce two of the most pressing challenges in quantum machine learning: the issue of vanishing gradients and the computational cost of computing gradients of expectation values.

2.3.1 Variational Quantum Eigensolvers

Before moving to the aforementioned challenges, let us describe one of the most promising applications of quantum machine learning: the variational quantum eigensolver (VQE) [21].

VQE is a hybrid quantum-classical algorithm designed to find the ground state energy of a quantum system. It leverages the strengths of both quantum and classical computing, making it a suitable candidate for near-term quantum devices.

We provide this brief introduction to VQE for two reasons. First, VQE is a good example to illustrate a quantum machine learning procedure that is not only emulating the classical counterpart but also leveraging quantum properties. In fact, the cost function used in VQE is a pure quantum object, being the expectation value of a Hamiltonian. Second, VQE is a hybrid quantum-classical algorithm, which fits well with the theme of this thesis. In fact, it is often used as a test case in the next chapters.

In a few words, the core idea behind VQE is to use a parametrized quantum circuit to prepare a trial state, which is then used to compute the expectation value (commonly referred to as *energy*) of a target Hamiltonian H_0:

$$C(\theta) = \langle \psi(\boldsymbol{\theta}) | H_0 | \psi(\boldsymbol{\theta}) \rangle, \tag{2.7}$$

where the variational state $|\psi(\boldsymbol{\theta})\rangle$ does not depend on any input data, since in this case we only need a parametric model of the ground state. Values of the form presented in Equation (2.7) are used within VQE pipelines as cost functions. A classical optimizer is

then employed to adjust the parameters of the circuit in order to minimize the energy expectation value. An illustrative drawing is shown in Figure 2.4.

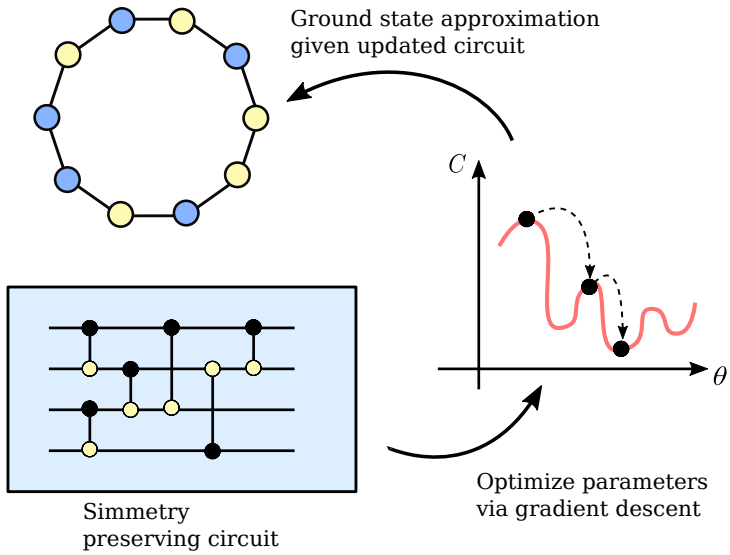


Figure 2.4: Schematic representation of a VQE. A parametrized quantum circuit is used to prepare a trial state, whose energy is measured and minimized using a classical optimizer. Following this minimization procedure, an approximation of the ground state of a target Hamiltonian is prepared.

By construction, the VQE algorithm is thought to provide an approximation of the ground state of a target Hamiltonian H_0. It is still important to note that this approximation is not guaranteed up to arbitrary precision. There are several factors one has to consider, including the expressibility of the ansatz, the trainability problems we will discuss in the next section, and the fact that the target Hamiltonian presents a gap between its lower eigenvalues or not.

To mitigate trainability problems, one can carefully design the ansatz and the optimization procedure. For example, as highlighted in Figure 2.4, one can use a circuit architecture which inherits the simmetries of the target Hamiltonian, reducing the size of the parameter space and potentially alleviating some of the trainability issues.

Despite the mentioned issues, VQEs remain one of the most promising algorithms to be implemented on near-term quantum devices, as they can potentially help solving complicated problems in combinatorial optimization and chemistry.

2.3.2 Barren Plateaus in Quantum Machine Learning

In this section, we introduce the first big challenge in quantum machine learning: the issue of vanishing gradients, also known as barren plateaus (BPs) [67].

To understand the barren plateau phenomenon, we first need to introduce the *Haar measure*. The Haar measure is a probability measure on the group of unitary matrices $U(2^n)$ that allows us to sample unitaries uniformly at random. A crucial property of the Haar measure is its invariance: if U is distributed according to the Haar measure, then for any fixed unitary V, both VU and UV are also Haar-distributed. This invariance makes the Haar measure the natural notion of "uniform randomness" over unitaries.

Connection to Parametrized Quantum Circuits

In quantum machine learning, we often work with parametrized quantum circuits $U(\boldsymbol{\theta})$ acting on an initial state $|\psi_0\rangle$. The circuit produces the state

$$|\psi(\boldsymbol{\theta})\rangle = U(\boldsymbol{\theta})\,|\psi_0\rangle\,. \tag{2.8}$$

The prediction is obtained by measuring an observable O:

$$f(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})|O|\psi(\boldsymbol{\theta})\rangle = \langle\psi_0|U^\dagger(\boldsymbol{\theta})OU(\boldsymbol{\theta})|\psi_0\rangle. \tag{2.9}$$

When the number of parameters in $U(\boldsymbol{\theta})$ is large, the distribution of unitaries generated by the circuit can approximate the Haar distribution. This approximation has profound implications for the behavior of expectation values and gradients.

Concentration of Measure and Lévy's Lemma

The key mathematical tool here is *Lévy's lemma*, which formalizes the concentration of measure phenomenon on high-dimensional spheres. In simple terms, Lévy's lemma states that for a Lipschitz-continuous function f on the unit sphere in $\mathbb{C}^{2^n}$, the value of f is exponentially likely to be close to its mean when the dimension 2^n is large. More precisely, if f has Lipschitz constant η, then for a random state $|\psi\rangle$,

$$\Pr\left[\,|f(|\psi\rangle) - \mathbb{E}[f]| \geq \epsilon\,\right] \leq 2\exp\left(-\frac{C\,2^n\epsilon^2}{\eta^2}\right), \tag{2.10}$$

where C is a positive constant.

In our setting, $f(|\psi\rangle) = \langle\psi|O|\psi\rangle$ is a 1-Lipschitz function (up to a factor depending on $\|O\|$). Thus, Lévy's lemma tells us that for Haar-random states, the expectation values $f(\boldsymbol{\theta})$ are exponentially concentrated around their mean.

An illustrative example of how the values of a function of the unitaries concentrates around the mean value is shown in Figure 2.5.

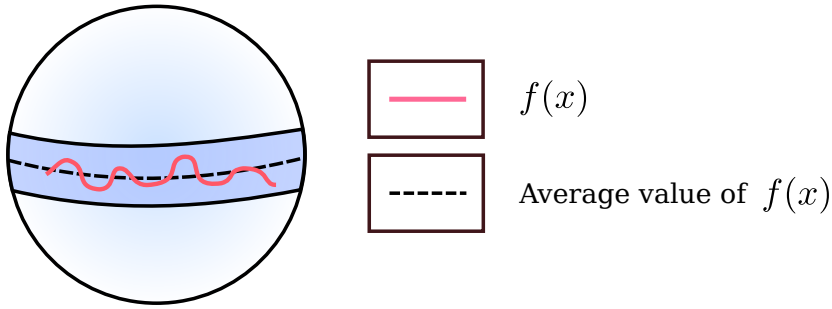


Figure 2.5: Illustration of the concentration of measure phenomenon. The black and dashed line represents the mean value of the function, while the red line represent the values of the function for different unitaries.

Expectation Values and Variance

For Haar-random unitaries, the mean of $f(\boldsymbol{\theta})$ is given by

$$\mathbb{E}_{\text{Haar}}[f(\boldsymbol{\theta})] = \frac{\text{Tr}(O)}{2^n}. \tag{2.11}$$

Moreover, the variance of $f(\boldsymbol{\theta})$ decays exponentially with the system size. For a traceless observable O (as is common in quantum machine learning, *e.g.*, Pauli operators), the variance is

$$\mathrm{Var}_{\mathrm{Haar}}[f(\boldsymbol{\theta})] = \frac{\mathrm{Tr}(O^2) - \mathrm{Tr}(O)^2/2^n}{2^{2n}-1} \approx \frac{\mathrm{Tr}(O^2)}{2^{2n}}. \tag{2.12}$$

Implications for Gradients

The cost function gradients are given by

$$\frac{\partial \mathcal{C}}{\partial \theta_i} = \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_i}. \tag{2.13}$$

Since $f(\boldsymbol{\theta})$ itself concentrates around its mean, Lévy's lemma implies that its derivatives also concentrate. In fact, one can show that

$$\mathrm{Var}\left[\frac{\partial f}{\partial \theta_i}\right] \propto 2^{-n}. \tag{2.14}$$

Thus, the typical magnitude of gradients decreases exponentially with the number of qubits.

The Barren Plateau Phenomenon

This exponential suppression of gradients is known as the *barren plateau* phenomenon [67]. In this regime, the optimization landscape becomes essentially flat: gradients are so small that they are indistinguishable from statistical noise due to finite sampling. As a result, any optimization method struggle in finding optimal parameters.

Barren plateaus therefore represent a fundamental obstacle for scalable quantum machine learning. They suggest that deep parametrized quantum circuits may be inherently difficult to train using standard optimization techniques, unless special circuit structures or initialization strategies are employed to avoid concentration.

In fact, a big effort has been make to identify problems and architectures which suffer less of the barren plateau phenomenon, such as quantum convolutional neural networks [68], or quantum circuits which preserve known simmetries of the solution [69]. It is important to note that it is still an open question whether a model which is not affected by barren plateaus is also non-classically simulable, namely, there is no classical algorithm which can efficiently reproduce the same results of the quantum model [70].

Another proposed approach to allow training in the presence of barren plateaus consist in preparing training *warm starts*, namely initializing the parameters of the quantum circuit close to a known good solution. This can help to escape the barren plateau by providing a better starting point for the optimization process. A pedagogical example of warm start in variational algorithms is presented in [71].

2.3.3 The Cost of Computing Gradients in Quantum Machine Learning

In this section, we discuss the second big challenge in quantum machine learning: the cost of computing gradients of expectation values.

In Section 2.2.1, we have seen that in classical machine learning the cost of computing the gradient of a cost function $\mathcal{C}$ with respect to the trainable parameters $\theta \in \Theta$ is comparable to the cost of a single forward pass of the network. This efficiency is a direct consequence of backpropagation.

In quantum machine learning, however, the situation is fundamentally different. The evaluation of a cost function typically involves the estimation of expectation values of observables on quantum states. Since quantum measurements are inherently probabilistic, any expectation value must be estimated from repeated measurements. The statistical error of such an estimate scales as $1/\sqrt{N}$, where N is the number of shots. Thus, obtaining a reliable estimate of the cost function or its gradient requires a large number of measurements, which can be costly in terms of time and quantum resources.

This motivates the development of quantum-specific gradient estimation techniques, among which the *parameter-shift rule* (PSR) [72, 73, 74] plays a central role.

Circuit Derivatives via Parameter-Shift Rule

The parameter-shift rule provides an exact method to compute derivatives of expectation values with respect to circuit parameters, without relying on finite-difference approximations. Consider an expectation value of the form

$$G(\theta) = \langle \psi_i | U^\dagger(\theta)\, O\, U(\theta) | \psi_i \rangle \,, \tag{2.15}$$

where $U(\theta)$ is a parameterized quantum circuit acting on an initial state $|\psi_i\rangle$, and O is an observable with at most two distinct eigenvalues. Suppose that the parameter θ enters the circuit through a single gate of the form $\exp(-i\theta P/2)$, where P is a Hermitian operator with eigenvalues ± 1.

In this setting, the derivative of $G(\theta)$ with respect to θ can be computed exactly as

$$\partial_\theta G(\theta) = r\Big[G(\theta + s) - G(\theta - s)\Big], \tag{2.16}$$

where s is a fixed shift (in the case of rotations we have $s = \pi/2$) and r is a normalization constant depending on the generator P of the gate. A detailed derivation of the parameter-shift rule can be found in [73].

The key advantage of the parameter-shift rule is that it reduces the problem of computing a derivative to the evaluation of the same expectation value at two shifted parameter settings. Each of these expectation values can be estimated using standard measurement procedures, and the resulting difference yields the exact gradient. This approach avoids the numerical instabilities of finite-difference methods and is compatible with the probabilistic nature of quantum measurements.

The parameter-shift rule was first introduced in [72] and later generalized in [73]. Subsequent work has extended the method to higher-order derivatives and to more general classes of parameterized gates [74].

Shift-Rules' Cost is not Backpropagation Cost

While the parameter-shift rule provides an efficient method to compute gradients in variational quantum circuits, its cost structure differs significantly from backpropagation in classical neural networks.

In classical machine learning, backpropagation allows for the efficient computation of gradients with respect to all parameters in essentially a single pass through the network. This is possible because the computational graph is static and can be traversed in reverse using the chain rule.

In contrast, the parameter-shift rule requires multiple evaluations of the same expectation value at different parameter settings. In the simplest case, each parameter

requires at least two additional circuit executions, though in general more evaluations may be needed depending on the type of parametrized gate. This results in an overhead that scales linearly with the number of parameters, similar in form to classical methods, but with a much larger constant factor because each gradient must be evaluated through fresh circuit runs.

Moreover, gradients must be estimated from repeated measurements (shots), which introduces statistical noise. Achieving accurate gradient estimates therefore requires a substantial number of circuit evaluations, whose cost can grow with both the number of parameters and the depth of the quantum circuit. In practice, this often makes variational training less efficient than classical backpropagation.

It is true that small quantum circuits have often produced intriguing results, and this has been frequently highlighted in demonstrations of quantum machine learning models. However, even if a quantum circuit can achieve comparable predictive performance with fewer parameters than a classical model, the computational cost per *quantum parameter* is typically much higher than that of a *classical parameter*, due to the reasons outlined above.

Can We Get Closer to Backpropagation Cost?

While the parameter-shift rule is currently the standard method for gradient evaluation in quantum machine learning, its reliance on repeated measurements makes it resource-intensive. Two promising directions to improve efficiency are *quantum backpropagation* [75] and *quantum non-demolition (QND) measurements*. The first relies on *shadow tomography*, a technique that estimates many expectation values of a quantum state from far fewer copies than required for full state tomography by learning only observable statistics rather than reconstructing the state itself. This enables a quantum version of backpropagation by providing multiple approximate copies of intermediate states without full state recovery. This allows, in principle, the reuse of quantum information in a way conceptually analogous to caching intermediate activations in classical backpropagation, potentially reducing the cost of gradient estimation. In parallel, the usage of ancillary qubits has been proposed in [76] to enable measuring the derivative of an expectation value with respect to a trainable parameter with a single expectation value computation. Both approaches are still in early development, but they highlight promising paths toward more scalable and resource-efficient training of quantum models.

A Bonus Level of The Orchestration Game

The challenges discussed so far are central to the scalability of quantum machine learning. At the same time, they also suggest new research directions in the design of hybrid quantum-classical algorithms.

Throughout this thesis, we emphasize the importance of coordinating the different components of these algorithms to achieve good performance. This task is meaningful in itself, even without focusing on a specific application. For instance, since the computation of gradients requires multiple evaluations of a quantum circuit, one can investigate strategies such as parallelizing the evaluations, reusing intermediate results, or distributing the computation across several quantum devices, possibly of different architectures. These approaches add a further layer of complexity to the orchestration problem and naturally raise new research questions.

For example: does distributing gradient computations across multiple quantum devices improve the robustness of models against noise? Can the multiple contributions to the gradient be incorporated into adaptive optimization schemes where learning rates and strategies are adjusted during training?

In our work we are already exploring some of these questions, for example through gradient batching in `Qiboml`, and resource orchestration in real-time error mitigation pipelines. We expect that the open problems and limitations of quantum machine learning will continue to motivate new strategies and solutions in the coming years.

CHAPTER 3

Towards Realistic Quantum Computing

In this chapter, we move from the idealized model of quantum computing described in Appendix A to a more realistic description of quantum devices. The objectives are threefold: *i)* to introduce superconducting qubits, one of the leading platforms for building quantum computers, *ii)* to present the basic notions of noise and its representation, from both a practical and theoretical perspective, and *iii)* to discuss strategies for mitigating the impact of noise on quantum computations.

3.1 A Glimpse into Superconducting Qubits

Among the various technologies proposed for qubit construction, superconducting qubits have emerged as one of the most promising candidates. Among their strengths are their relatively long coherence times, fast gate speeds, and the ability to be fabricated using established semiconductor techniques. These advantages make superconducting qubits a leading choice for building scalable quantum processors.

The following section is a selection of concepts and descriptions highlighted from [77]. We strongly recommend the interested reader to refer to the original paper for a more comprehensive understanding of the subject.

To describe the physics of superconducting qubits we will restart from the Schrödinger equation, which governs the dynamics of closed quantum systems:

$$i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle = H|\psi(t)\rangle, \tag{3.1}$$

where $\hbar$ is the reduced Planck constant, $|\psi(t)\rangle$ is the state vector of the system at time t, and H is the Hamiltonian operator, which encodes the total energy of the system. Every time we consider a quantum gate to build an algorithm, we are implicitly assuming that this unitary operation corresponds to a perturbation of the Hamiltonian of our physical system. What is elegantly captured into a quantum circuit notation corresponds, in practice, to a sequence of physical operations that have to be understood and implemented on a real device.

The technology we need is a two-level quantum system whose state can be arbitrarily manipulated. In the case of a superconducting qubit, the physical implementation is inspired by a superconducting circuit composed of a capacitor C and an inductor L. The energy of such a system can be described as a harmonic oscillator, allowing us to access various energy levels.

The Hamiltonian of this system can be expressed as a linear combination of the contributions of the capacitor and the inductor:

$$H = H_C + H_L = \frac{Q^2}{2C} + \frac{\Phi^2}{2L}, \tag{3.2}$$

where Q is the charge operator and Φ is the flux operator, fulfilling the canonical commutation relation $[Q, \Phi] = i\hbar$. In fact, Equation 3.2 is analogous to the one of a harmonic oscillator with mass $m = C$ and resonance frequency $\omega = 1/\sqrt{LC}$. Both the C and L are linear circuit elements, and, once defining the reduced flux $\phi = 2\pi\Phi/\Phi_0$ and the reduced charge $n = Q/2e$, we can rewrite it as

$$H = 4E_C n^2 + \frac{1}{2}E_L\phi^2, \tag{3.3}$$

where $E_C = e^2/(2C)$ is the charging energy and $E_L = (\Phi_0/2\pi)^2/L$ is the inductive energy. This Hamiltonian describes a quantum harmonic oscillator with equidistant energy levels.

3.1.1 The Josephson Junction and the Origin of Anharmonicity

To obtain a true qubit, we need to break the harmonicity of the spectrum. This is achieved by introducing a *Josephson junction*, a nonlinear, dissipationless circuit element formed by two superconductors separated by a thin insulating barrier. The junction supports a supercurrent I depending on the superconducting phase difference ϕ as

$$I = I_c \sin(\phi), \qquad V = \frac{\hbar}{2e}\frac{d\phi}{dt}, \tag{3.4}$$

where the *critical current* I_c is the maximum supercurrent that can flow without generating a voltage drop, i.e., before tunnelling becomes dissipative. The associated potential energy reads

$$U(\phi) = -E_J \cos(\phi), \qquad E_J = \frac{I_c\Phi_0}{2\pi}, \tag{3.5}$$

with $\Phi_0 = h/2e$ the superconducting flux quantum.
Expanding the cosine for small ϕ,

$$U(\phi) \approx -E_J\left(1 - \frac{\phi^2}{2} + \frac{\phi^4}{24} - \dots\right). \tag{3.6}$$

The quadratic term yields a harmonic oscillator, while the quartic term introduces a weak nonlinearity with a negative coefficient, reducing the spacing between $|0\rangle \to |1\rangle$ and $|1\rangle \to |2\rangle$ transitions. This produces a negative anharmonicity

$$\alpha = \omega_{12} - \omega_{01} \approx -E_C, \tag{3.7}$$

where ω_{ij} denotes the transition frequency between levels i and j. This is a defining property of the transmon qubit[1].

[1] A transmon qubit is a charge-noise-insensitive superconducting qubit derived from the Cooper pair box, operated in the regime $E_J/E_C \gg 1$ to reduce charge dispersion while retaining sufficient anharmonicity for selective control.

In the transmon, α is typically 100–300 MHz, enabling a qubit frequency $\omega_q = (\sqrt{8E_J E_C} - E_C)/\hbar$ in the 4–6 GHz range while maintaining $E_J/E_C \gg 1$.

Neglecting higher non-computational states, the qubit Hamiltonian reduces to

$$H = \omega_{\mathrm{q}} \frac{\sigma_z}{2}, \tag{3.8}$$

where σ_z is the Pauli Z operator and ω_q is the qubit transition frequency.

3.1.2 Qubit Control and Single-Qubit Gates

Once we have defined a two-level system, the next step is to learn how to *control* it. In practice, this means being able to implement arbitrary rotations of the qubit state on the Bloch sphere. In superconducting qubits, this is achieved by capacitively coupling the qubit to an external microwave drive. The drive provides an oscillating voltage $V_d(t)$, which couples to the qubit charge degree of freedom. After quantization and truncation to the two lowest levels (see Equation (79) of [77]), the interaction can be written schematically as

$$H_d(t) = \Omega V_d(t)\, \sigma_y, \qquad \Omega = \frac{C_{\mathrm{d}}}{C_\Sigma} Q_{\mathrm{zpf}}, \tag{3.9}$$

where σ_y is a Pauli operator acting on the qubit subspace and the constant Ω depends on the coupling capacitance C_{d}, the total capacitance to ground C_Σ, and the zero-point charge fluctuation $Q_{\mathrm{zpf}} = \sqrt{\hbar/(2Z)}$, which sets the natural quantum scale of charge fluctuations in the ground state of the circuit.

The drive signal. The applied voltage is a microwave pulse of the form

$$V_d(t) = V_0\, s(t) \cos(\omega_d t + \phi), \tag{3.10}$$

where V_0 is the amplitude, $s(t)$ is a slowly varying envelope (the pulse shape), ω_d is the drive frequency, and ϕ is the drive phase. The envelope $s(t)$ is typically generated by an arbitrary waveform generator (AWG), while the fast oscillation at frequency ω_d comes from a stable microwave source. Together, they form a shaped pulse that can selectively address the qubit.

From the lab frame to the rotating frame. In the laboratory frame, the qubit Hamiltonian contains a term $\frac{\omega_q}{2}\sigma_z$, which causes the qubit state to precess around the z-axis at its natural frequency ω_q. If we now add the drive, the Hamiltonian contains fast oscillating terms at frequencies $\omega_q \pm \omega_d$. These terms make the dynamics difficult to interpret directly.

To simplify the description, we move into a *rotating frame* at the frequency $\omega_d = \omega_q$. In this frame, the natural precession of the qubit is removed, so that we only see the relative motion between the drive and the qubit. Applying the *rotating-wave approximation* (RWA), which neglects the very fast oscillating terms that average to zero, the drive Hamiltonian reduces to

$$H_d(t) = \Omega(t) \left(\cos\phi\, \sigma_x + \sin\phi\, \sigma_y\right), \tag{3.11}$$

where $\Omega(t) = \frac{\Omega}{2} V_0 s(t)$ is the *Rabi frequency*, *i.e.* the rate at which the qubit state rotates under the drive.

After defining this framework, our next step is to understand how the drive can be used to implement arbitrary rotations on the Bloch sphere, which correspond to the theoretical quantum gates introduced in Appendix A.

Rotations on the Bloch sphere. Equation (3.11) shows that the drive implements rotations in the equatorial plane of the Bloch sphere. The phase ϕ selects the axis of rotation: $\phi = 0$ corresponds to the x-axis, while $\phi = \pi/2$ corresponds to the y-axis. The strength of the drive determines the *Rabi frequency* $\Omega(t)$, which sets the instantaneous speed of rotation. The total rotation angle is given by the time integral of the drive:

$$\Theta(t) = \int_0^t \Omega(t')\, dt'. \tag{3.12}$$

The corresponding unitary evolution is

$$U(t) = \exp\left[-i\,\frac{\Theta(t)}{2}\,(\cos\phi\,\sigma_x + \sin\phi\,\sigma_y)\right]. \tag{3.13}$$

This compact expression makes the control mechanism transparent: the pulse *amplitude* sets the speed of rotation, the *duration* sets the total angle, and the *phase* sets the axis. In practice, this means that by tuning V_0, $s(t)$, and T, one can calibrate the desired gate. For example, a pulse with $\Theta(T) = \pi$ implements a full X_π or Y_π gate, while $\Theta(T) = \pi/2$ implements a $\pi/2$-rotation. A simple square pulse of duration T can thus be calibrated such that $\Omega V_0 T = \pi$ to realize a π-pulse, while halving the duration (or amplitude) produces a $\pi/2$-pulse.

Virtual-Z gates. Rotations around the z-axis can be implemented virtually by shifting the reference phase of subsequent pulses, a technique known as the *virtual-Z gate* [78]. These gates are effectively instantaneous and error-free, since they require no physical pulse.

DRAG pulses. Because the transmon is only weakly anharmonic, short pulses can inadvertently populate higher levels, leading to leakage errors. The *Derivative Removal by Adiabatic Gate* (DRAG) technique mitigates this by adding a quadrature correction proportional to the derivative of the pulse envelope, reducing both leakage and phase errors [79, 80].

3.1.3 Two-Qubit Gates in Superconducting Circuits

While single-qubit rotations are essential for universal control, a scalable quantum computer also requires reliable *two-qubit gates*. In the superconducting platform, these are typically realized by coupling two transmon qubits through a tunable or fixed interaction channel.

Coupling mechanisms. The most common approach is to connect qubits via a shared circuit element, such as a capacitor or an inductor, which introduces an interaction term in the Hamiltonian of the form

$$H_{\text{int}} = g\,(\sigma_1^+\sigma_2^- + \sigma_1^-\sigma_2^+), \tag{3.14}$$

where $\sigma^{\pm}$ are the raising and lowering operators of the qubits and g is the coupling strength. This exchange-type interaction allows excitations to be swapped between the two qubits. Another option is to couple both qubits to a common resonator, where the interaction arises from virtual excitations of the resonator mode. Finally, tunable couplers based on additional Josephson junctions can be used to switch the interaction g on and off dynamically.

From coupling to gates. To implement a controlled operation, one exploits the interaction either directly or through carefully timed frequency tuning. A basic example is the *iSWAP* gate, obtained by letting the exchange interaction act for a fixed duration, effectively swapping the quantum states of the two qubits.

Another central primitive is the *controlled-phase* (CZ) gate.

This is implemented by bringing the energy levels of the two qubits close to resonance with a higher non-computational state (such as $|11\rangle$ interacting with $|20\rangle$), so that the joint state $|11\rangle$ acquires an additional phase shift. After a calibrated interaction time, this produces the desired CZ gate.

Fidelity and challenges. In practice, two-qubit gates are more error-prone than single-qubit rotations. Their quality is usually quantified by the *average gate fidelity*, defined as

$$F_{\text{avg}}(U, \mathcal{E}) = \int d\psi \; \langle\psi|U^{\dagger}\, \mathcal{E}(|\psi\rangle\langle\psi|)\, U|\psi\rangle, \tag{3.15}$$

where U is the target unitary gate, $\mathcal{E}$ is the noisy channel describing the implemented operation, and the integral runs over all pure input states $|\psi\rangle$. In experiments, this quantity is typically estimated through *randomized benchmarking*, which avoids the need for full process tomography. A fidelity of $F = 0.99$, for example, means that the gate performs as intended 99% of the time. Current superconducting processors achieve two-qubit gate fidelities slightly above 99%, but these gates remain the main bottleneck for scaling up quantum hardware [77].

In Chapter 1 we mentioned that T gates are the usual cost metric for fault-tolerant quantum computing. Since we are now discussing current, real devices, it is important to mention that, in the NISQ era, the real cost metric is the two-qubit gate count, as these gates are typically one to two orders of magnitude noisier than single-qubit gates. This is why, in this thesis, we will use the number of CZ gates as a cost metric for our quantum algorithms (see Chapters 12 as an example).

3.1.4 Qubit Readout.

In addition to controlling a qubit, we must also be able to *measure* its state. For superconducting qubits, this is typically achieved using *dispersive readout*, where the qubit is coupled to a microwave resonator. In the dispersive regime, the qubit induces a small shift of the resonator frequency depending on whether it is in $|0\rangle$ or $|1\rangle$. A common way to visualize this is to sweep the probe frequency and observe that the resonance peak moves depending on the qubit state. In practice, however, readout is performed by sending a short microwave pulse at (or near) the resonator frequency and recording the outgoing signal. The qubit state is inferred from the amplitude and phase of this signal, which map onto the z-axis of the Bloch sphere: the measurement projects the qubit onto either $|0\rangle$ (north pole) or $|1\rangle$ (south pole). High-fidelity readout requires engineering

the resonator to be fast while suppressing unwanted decay (the *Purcell effect*), and amplifying the weak readout signal with near-quantum-limited devices such as Josephson parametric amplifiers.

3.2 Noise Affecting Quantum Devices

When considering a closed quantum system, we can deterministically describe its evolution knowing the initial state and the Hamiltonian. If so, we can retrieve the system's state at any time t by solving the Schrödinger equation (Equation 1.2 of Appendix A). However, in realistic scenarios, quantum systems are not isolated and interact with their environment, leading to noise and decoherence. This complicates the description of their dynamics and poses challenges for quantum computation. An illustrative diagram representing some of the possible noise sources in quantum computing is shown in Figure 3.1.

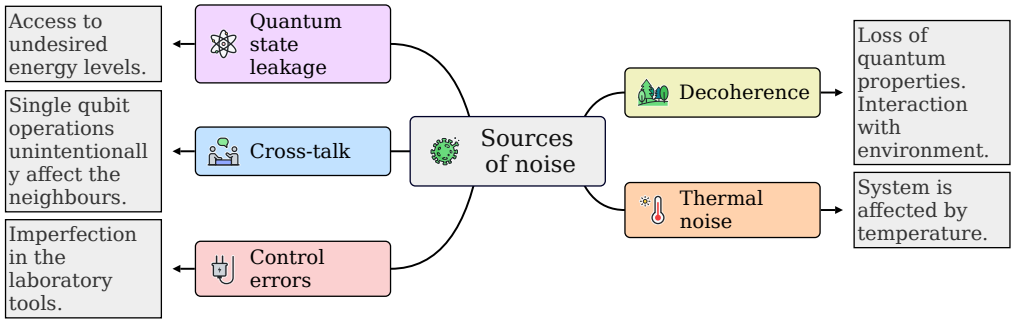


Figure 3.1: Illustrative diagram of some noise sources in quantum computing.

The two main classes of noises are, as per any physical experiment, systematic noises and stochastic noises. The former are usually caused by construction imperfections or calibration errors, while the latter are due to random fluctuations of the system properties, that can be related to interaction with the environment (*e.g.*, thermal noise, electromagnetic interference) or natural random behaviour of a quantum system (*e.g.*, spontaneous emission, photon shot noise).

Once accepted the idea that noise plays a crucial role in nowadays quantum devices, what we can do is from one side understand how to model it and characterize it, and from the other side how to intervene to remove it or, most likely, mitigate it.

In the following, we first introduce a theoretical framework to represent the noise of quantum computers, and then discuss one of the most common approach to deal with it: *quantum error mitigation* (QEM).

3.2.1 Modeling Noise

Bloch Sphere and Open Quantum Systems

To visualize the effects of noise on a quantum state we start by recalling the Bloch sphere representation introduced in Section 1.2 of Appendix A.

In the ideal case we deal with pure quantum states, namely states that cannot be written as probabilistic mixture of other quantum states. These states can be exactly

represented as vectors in the Hilbert space and, in the Bloch sphere notation, correspond to points on the surface of the sphere. We can manipulate the state of a quantum system using unitary operators and this actions can be represented as rotations on the Bloch sphere (left panel of Figure 3.2).

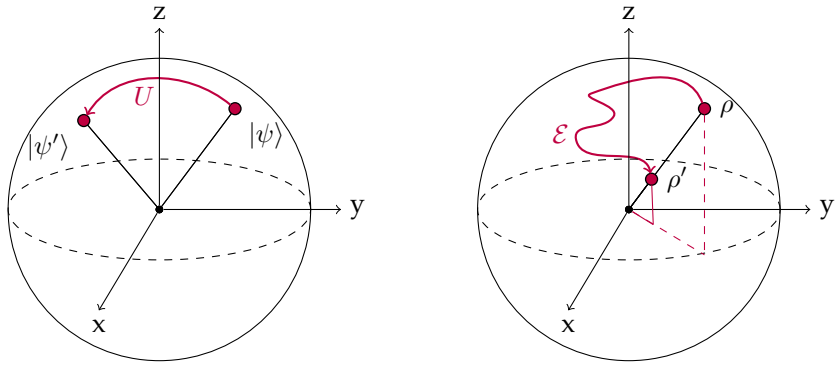


Figure 3.2: Left, Bloch sphere representation of a unitary operation applied to a pure state. U is mapping a point on the surface of the sphere into another point on the surface. Right, Bloch sphere representation of a noisy superoperator applied to a density matrix ρ. If applied to an initial pure state ρ', located on the surface of the sphere, the superoperator $\mathcal{E}$ maps it into a mixed state, represented by a point inside the sphere.

When considering an open system, the state can be affected by environmental interactions, leading to mixed states. Mixed states are commonly represented using the *density matrix* formalism. For example, if we consider a mixed state ρ with is a mixture of k pure states, its density matrix can be written as:

$$\rho = \sum_{i=1}^{k} p_i \left|\psi_i\right\rangle \left\langle\psi_i\right| , \tag{3.16}$$

where p_i is the probability of the system being in the pure state $|\psi_i\rangle$. Note that, if $k = 1$, then we have a pure state.

If statevectors can be manipulated using unitary operators, we usually operate on density matrices with *superoperators*, or *channels*. A superoperator is an operator that acts on the space of operators (such as density matrices) rather than on statevectors directly. As schematically depicted in the right panel of Figure 3.2, the action of a superoperator on an initial pure state can be visualized as a transformation that maps one point of the sphere into a point inside the sphere.

As for pure states, measurements are needed to access information once a state ρ is prepared. We can still use the Born rule and the trace operation to compute the expectation value of an observable O on a density matrix ρ prepared applying a channel $\mathcal{E}$ to an initial state ρ_0:

$$\mathbb{E}[O] = \mathrm{Tr}\big[O\mathcal{E}(\rho_0)\big]. \tag{3.17}$$

Before moving to noise modeling, we also introduce a general expression to describe the density matrix of a qubit where the amplitudes of the state are related by $|\alpha|^2 + |\beta|^2 = 1$:

$$\rho = \frac{1}{2}(I + \vec{a}\cdot\vec{\sigma}) = \frac{1}{2}\begin{pmatrix} 1+\cos\theta & \sin\theta e^{-i\phi} \\ \sin\theta e^{i\phi} & 1+\sin\theta \end{pmatrix} = \begin{pmatrix} \cos^2\frac{\theta}{2} & \sin\theta/2\cos\theta/2 e^{-i\phi} \\ \sin\theta/2\cos\theta/2 e^{i\phi} & \sin^2\theta/2 \end{pmatrix}$$

$$= \begin{pmatrix} |\alpha|^2 & \alpha\beta^* \\ \alpha^*\beta & |\beta|^2 \end{pmatrix}, \tag{3.18}$$

where I is the identity matrix, $\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_z)$ are the Pauli matrices and $\vec{a}$ is the Bloch vector. If ρ is pure, then $\vec{a}$ is unit and $\mathrm{Tr}[\rho^2] = 1$. On the other hand, if ρ is a mixed state, then $\vec{a}$ is not unit and $0 \leq \mathrm{Tr}[\rho^2] < 1$.

The Bloch-Redfield Model

A common approach to model the noise affecting a quantum system is the Bloch-Redfield model, which describes the dynamics of the density matrix of a qubit in the presence of environmental interactions. The Bloch-Redfield equations provide a framework to account for the effects of these interactions on the qubit state, yielding a more realistic description of its time evolution.

The Bloch-Redfield density matrix can be written as:

$$\rho_{\mathrm{BR}}(t) = \begin{pmatrix} 1 + (|\alpha|^2 - 1)e^{-\Gamma_1 t} & \alpha\beta\, e^{i\delta\omega t} e^{-\Gamma_2 t} \\ \alpha^*\beta\, e^{-i\delta\omega t} e^{-\Gamma_2 t} & |\beta|^2 e^{-\Gamma_1 t} \end{pmatrix}, \tag{3.19}$$

which resembles Equation (3.18) but includes additional terms to represent: *i)* longitudinal relaxation, *ii)* pure dephasing, and *iii)* transverse relaxation. In particular, we introduce the longitudinal and transverse decay rates Γ_1 and Γ_2, and the phase accrual term $e^{-i\delta\omega t}$, where $\delta\omega = \omega_q - \omega_d$ accounts for the detuning between the qubit transition frequency ω_q and the rotating-frame frequency ω_d.

Longitudinal relaxation It is described by the parameter $\Gamma_1 \equiv 1/T_1$, which represents the rate of energy exchange between the states $|0\rangle$ and $|1\rangle$. The time constant T_1 is the characteristic decay time, *i.e.* the time after which the excited-state population has decreased to $1/e \approx 37\%$ of its initial value. In practice, we usually assume that Γ_1 is dominated by the decay process $|1\rangle \to |0\rangle$, while $|0\rangle$ is stable once prepared. This behavior is reflected in Equation (3.19): for $t \gg T_1$, the population is entirely relaxed into $|0\rangle$.

Pure dephasing It is described by the parameter Γ_φ, which quantifies the loss of phase coherence between the computational basis states without any exchange of energy with the environment. To see this, recall that the qubit has a natural transition frequency ω_q, which appears in the Hamiltonian of Equation (3.8). If we prepare a superposition state such as $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, its time evolution in the lab frame is

$$|\psi(t)\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{-i\omega_q t}|1\rangle\right). \tag{3.20}$$

In practice, we often move to a rotating frame to *factor out* this deterministic precession, so that the state appears as a fixed point on the Bloch sphere. However, a real qubit is subject to fluctuations in its transition frequency due to broadband environmental noise. These fluctuations cause the relative phase between $|0\rangle$ and $|1\rangle$ to drift stochastically, leading to a decay of the off-diagonal elements of the density matrix. On the Bloch sphere, this corresponds to a randomization of the azimuthal angle and a shrinking of the Bloch vector in the equatorial plane, at a rate Γ_φ.

Unlike energy relaxation, pure dephasing is an *elastic* process: no energy is exchanged with the environment. In principle, this means that the lost phase coherence can be partially recovered through unitary control techniques such as dynamical decoupling [81]. The degree of recoverability depends on the noise spectrum and the speed of control pulses. This should be contrasted with spontaneous energy relaxation, which is an irreversible process: once the qubit emits energy into the environment, the quantum information is irretrievably lost.

Transverse relaxation It is described by the parameter Γ_2, which is related to the previously discussed Γ_1 and Γ_φ through the relation:

$$\Gamma_2 = \frac{\Gamma_1}{2} + \Gamma_\varphi. \tag{3.21}$$

Thus, the transverse relaxation rate Γ_2 is influenced by both energy relaxation and pure dephasing. In practice, while the qubit undergoes natural phase randomization, the $|1\rangle$ component of the state may also decay into $|0\rangle$ as described by Γ_1. This is a phase-breaking event, since we lose all information about the azimuthal orientation of the Bloch vector in the equatorial plane.

Pauli Noise Channel

The Bloch–Redfield model provides a continuous-time description of qubit decoherence, where the density matrix evolves under the influence of longitudinal relaxation (Γ_1), pure dephasing (Γ_φ), and their combined effect in transverse relaxation (Γ_2). This approach is useful when analyzing qubits as open quantum systems interacting with an environment. In quantum information theory, however, it is often more convenient to adopt a discrete description, where noise is represented by quantum channels acting on the qubit state after each gate or time step. In this picture, the Pauli noise channel is one of the simplest models, as it applies Pauli operators with given probabilities. More general channels can also be defined, such as the Kraus channel (arbitrary Kraus operators), the Unitary channel (a probabilistic mixture of unitaries), or multi-qubit and correlated noise models. These channel-based descriptions provide a flexible framework that complements the Bloch–Redfield model. For a practical introduction to channels, we suggest the reader to refer to the channels section of our documentation in `Qibo` [82].

The Pauli channel is defined as

$$\mathcal{E}(\rho) = (1-p)\rho + p_x X\rho X + p_y Y\rho Y + p_z Z\rho Z, \tag{3.22}$$

where X, Y, Z are the Pauli operators and $p_x + p_y + p_z = p$ is the total error probability. This channel models random bit-flip (X), phase-flip (Z), or combined (Y) errors.

Effect on the state. Each Pauli error has a clear geometric interpretation on the Bloch sphere:

- An X error flips the state across the x-axis, exchanging $|0\rangle \leftrightarrow |1\rangle$. This mimics the effect of energy relaxation, where population leaks from $|1\rangle$ into $|0\rangle$.
- A Z error flips the phase of the $|1\rangle$ component, mapping $\alpha|0\rangle + \beta|1\rangle$ to $\alpha|0\rangle - \beta|1\rangle$. This corresponds to pure dephasing, where the relative phase between $|0\rangle$ and $|1\rangle$ becomes randomized.
- A Y error combines both effects, flipping the state across the y-axis and introducing both a bit-flip and a phase-flip.

Repeated application of the Pauli channel drives the qubit state toward the completely mixed state $\rho = I/2$, since the Bloch vector is progressively randomized and shrinks toward the origin.

Decay of expectation values. The action of the Pauli channel can also be understood in terms of expectation values of Pauli operators. For a qubit state ρ, the expectation value of a Pauli operator $P \in \{X, Y, Z\}$ is $\langle P \rangle = \mathrm{Tr}(\rho P)$. Under the Pauli channel, these expectation values decay as

$$\langle P \rangle \longrightarrow (1 - 2p_P)\langle P \rangle, \tag{3.23}$$

where p_P is the probability of applying an error that anticommutes with P. For example, a Z error flips the sign of $\langle X \rangle$ and $\langle Y \rangle$, but leaves $\langle Z \rangle$ unchanged. Averaging over many random errors causes the expectation values of all Pauli operators to decay toward zero, reflecting the loss of information about the qubit state. This is the discrete-channel analogue of the exponential decay of Bloch vector components in the Bloch-Redfield model.

From rates to probabilities. To connect the continuous-time rates to discrete error probabilities, one can consider a short time step Δt. The probability of an error occurring in that interval is approximately

$$p_x \approx \frac{\Delta t}{2T_1}, \quad p_z \approx \frac{\Delta t}{T_\varphi}, \quad p_y \approx 0, \tag{3.24}$$

where the factor of $1/2$ accounts for the fact that T_1 relaxation only affects the excited state population. More generally, the mapping can be refined by expanding the Kraus operators of the amplitude-damping and phase-damping channels to first order in Δt.

Limitations of the Pauli channel. While the Pauli channel provides a simple and widely used abstraction, it does not capture all realistic noise processes. In particular:

- It assumes that errors are instantaneous and discrete, whereas real decoherence is continuous in time.
- It neglects *coherent errors*, such as systematic over-rotations or frequency detunings, which accumulate deterministically rather than stochastically.
- It cannot describe correlated noise across multiple qubits, which is common in realistic devices (*e.g.*, crosstalk or common-mode fluctuations).

- It does not capture non-Markovian effects, where the environment has memory and the noise is not simply random and uncorrelated in time.

For these reasons, the Pauli channel is best viewed as a convenient *effective model* for use in quantum error correction and fault-tolerant analysis, while more detailed models such as Bloch-Redfield or Lindblad master equations are required to capture the full physics of decoherence.

Readout Noise

In addition to decoherence during the coherent evolution of the qubit, imperfections also arise at the measurement stage. In superconducting qubits, for example, the readout is typically performed by coupling the qubit to a resonator and measuring the transmitted or reflected microwave signal. Due to finite signal-to-noise ratio, amplifier noise, and imperfect discrimination between the states, the measurement outcome can be misclassified. This effect is referred to as *readout noise*.

Bit-flip model. A simple and widely used way to model readout noise is through a classical bit-flip channel acting on the measurement outcomes. If the true outcome is $b \in \{0, 1\}$, the reported outcome $\tilde{b}$ is given by

$$\Pr(\tilde{b} = b) = 1 - p_r, \qquad \Pr(\tilde{b} \neq b) = p_r, \tag{3.25}$$

where p_r is the readout error probability. In this model, the measurement device flips the outcome with probability p_r, independently of the quantum state preparation or evolution.

Readout error matrix. More generally, one can describe readout noise using a 2×2 stochastic matrix M that maps the true distribution of outcomes $\vec{p}$ to the observed distribution $\vec{p}_{\text{obs}}$:

$$\vec{p}_{\text{obs}} = M\,\vec{p}, \qquad M = \begin{pmatrix} 1 - p_{1|0} & p_{0|1} \\ p_{1|0} & 1 - p_{0|1} \end{pmatrix}, \tag{3.26}$$

where $p_{1|0}$ is the probability of misclassifying $|0\rangle$ as $|1\rangle$, and $p_{0|1}$ is the probability of misclassifying $|1\rangle$ as $|0\rangle$. The symmetric case $p_{1|0} = p_{0|1} = p_r$ reduces to the simple bit-flip model above.

Impact on expectation values. Readout noise reduces the contrast of measured expectation values. For example, if we measure Z on a qubit prepared in $|0\rangle$, the ideal expectation value is $\langle Z \rangle = +1$. With readout error probability p_r, the observed expectation value becomes

$$\langle Z \rangle_{\text{obs}} = (1 - 2p_r)\langle Z \rangle. \tag{3.27}$$

Thus, readout noise effectively shrinks the measured Bloch vector along the measurement axis, similar to how decoherence shrinks it during evolution.

3.3 Facing Noise: Error Correction and Error Mitigation

The aim of this section is introducing the concepts of quantum error correction (QEC) [56] and quantum error mitigation (QEM) [57], focusing on the second approach. In this thesis, quantum error mitigation techniques have been implemented and tested within quantum machine learning algorithms, with the goal of improving their robustness against noise and errors. QEC has not been investigated, but deserves to be introduced since it is the key to achieving fault-tolerant quantum computation.

Both these approaches offers an incredible playground to explore combined usage of classical and quantum computers, since they are often composed of classical routines combined with operations on the quantum device.

3.3.1 Quantum Error Correction

Quantum error correction is the framework that ultimately need to implement a reliable quantum computation in the presence of noise. The goal of QEC is indeed to correct the results, completely removing errors from the computations.

The central idea is to encode a single logical qubit into a larger number of physical qubits, distributing the quantum information across entangled states so that local errors can be detected and corrected without disturbing the logical state. For example, the simplest three-qubit repetition code encodes $\alpha |0\rangle_L + \beta |1\rangle_L = \alpha |000\rangle + \beta |111\rangle$, allowing single bit-flip errors to be identified and corrected. More sophisticated codes, such as the surface code, can correct both bit-flip and phase-flip errors simultaneously.

By measuring suitable *error syndromes*, one can identify the type and location of errors and apply corrective operations, thereby protecting quantum information from decoherence. The threshold theorem guarantees that arbitrarily long quantum computations are possible if the physical error rates remain below a critical value (typically $\sim 10^{-3}$ for surface codes). However, QEC comes with substantial resource overhead: hundreds or thousands of physical qubits are needed to encode a single logical qubit with sufficient protection.

In the current noisy intermediate-scale quantum (NISQ) era, such large overheads are not yet practical. For this reason, *error mitigation* techniques have emerged as a more near-term approach, aiming to reduce the impact of noise on quantum computations without the prohibitive qubit requirements of full error correction.

3.3.2 Quantum Error Mitigation

If QEC is not feasible with current computational resources, quantum error mitigation stands as a promising alternative for utilizing near-term quantum devices.

The goal of QEM is not to correct errors, but rather to exploit the knowledge we have about noise characteristics and use them to mitigate its effects. What a typical QEM algorithm returns is a post-processed result that exhibits reduced bias compared to the raw noisy outcome, namely, the distance between the true expectation value and the measured value. We recommend the reader consult Reference [83] for a systematic and clear analysis of error statistics and scalability of QEM techniques.

We focus here on QEM techniques whose aim is to mitigate results in the form of expectation values. Several algorithms have been proposed, such as zero-noise extrapolation (ZNE) [84, 85], which reduces the impact of noise by extrapolating results obtained at different noise levels to estimate the ideal outcome, and probabilistic error cancellation (PEC) [84], which uses knowledge about the noise model to probabilistically correct the

results. In this thesis, we focus on *data-driven* techniques, which leverage data sampled from quantum devices to learn a noise model and train a classical algorithm to mitigate it.

We focus on these methods for two main reasons. First, we aim to implement them in the context of quantum machine learning algorithms, in the form of real-time error mitigation routines. These techniques have been identified as promising candidates for handling noise in variational tasks [86]. Second, they provide a natural setting to explore the combined use of classical and quantum resources, since they often rely on classical routines interleaved with operations on the quantum device.

Learning-Based Quantum Error Mitigation

To describe how learning-based error mitigation techniques work, we consider a target quantum circuit C_0, which prepares a state on which we want to compute the expectation value of an observable O. We assume that C_0 is a circuit we must execute on a real quantum computer, namely involving a large number of qubits, considerable entanglement, and a large number of T gates.

We also know that the quantum computer is noisy, and the best we can do by executing our circuit on the device is to obtain a noisy expectation value

$$\langle O_0 \rangle_{\text{noisy}} = \text{Tr}\big[O\mathcal{E}[C_0(\rho_0)]\big], \tag{3.28}$$

where $\rho_0 = |0\rangle\langle 0|^{\otimes n}$ is the initial state of the system and $\mathcal{E}$ represents the noise process affecting the circuit.

We can now consider a set of circuits $\{C_i\}_{i=1}^N$ which, when executed on the quantum computer, are representative of the same noise characteristics as C_0: same number of qubits, similar circuit structure, and comparable gate count.

What is crucial is that these circuits are classically simulable. In practice, we use a classical computer to simulate the calculation of the ideal expectation value

$$\langle O_i \rangle_{\text{exact}} = \text{Tr}\big[OC_i(\rho_0)\big] \tag{3.29}$$

and the quantum computer to obtain the corresponding noisy expectation value

$$\langle O_i \rangle_{\text{noisy}} = \text{Tr}\big[O\mathcal{E}[C_i(\rho_0)]\big]. \tag{3.30}$$

The circuits $\{C_i\}_{i=1}^N$, which we refer to as *training circuits*, are usually constructed by reducing the amount of magic (T gates) in the circuit, making them amenable to Clifford or quasi-Clifford simulators for computing expectation values. A more detailed discussion of these classical simulation techniques can be found in Section 4.3.2. In the scheme presented in Figure 3.3, we consider a classical simulation technique known as *hybrid stabilizer-MPO* representation, which will be discussed in Section 9.1.

Once the *training data* $\{\langle O_i \rangle_{\text{noisy}}\}_{i=1}^N$ and $\{\langle O_i \rangle_{\text{exact}}\}_{i=1}^N$ are available, we can use them to learn a mitigation map ℓ that takes noisy expectation values as input and returns mitigated values:

$$\langle O_0 \rangle_{\text{mit}} = \ell\big(\langle O_0 \rangle_{\text{noisy}}\big). \tag{3.31}$$

Bias-variance tradeoffs in learning-based QEM. The effectiveness of learning-based error mitigation involves important bias-variance tradeoffs. When properly optimized, these algorithms can significantly reduce the bias in expectation value estimates. However, this bias reduction typically comes at the cost of increased variance in the results.

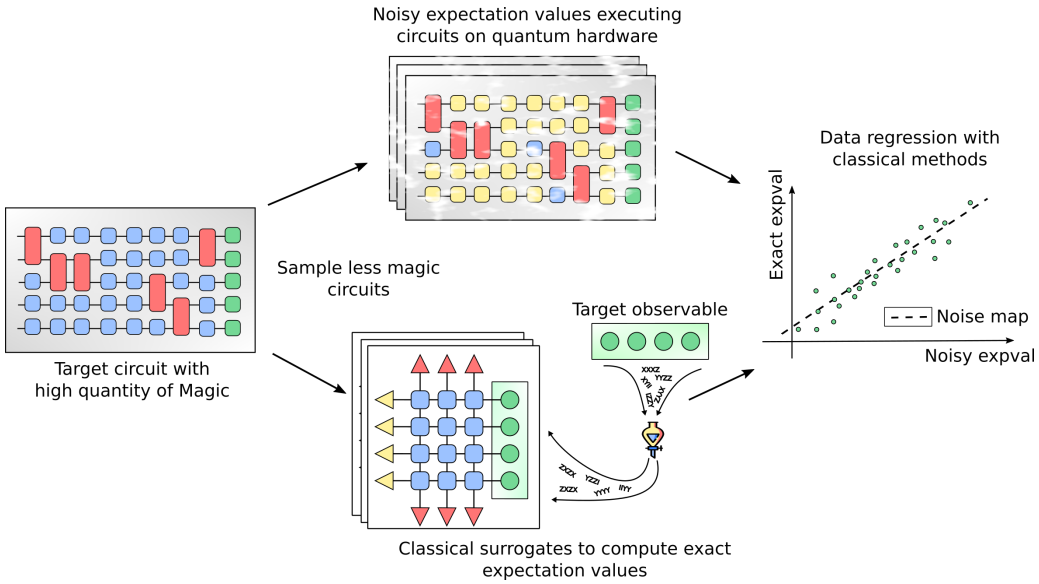


Figure 3.3: Illustrative scheme of a data-driven error mitigation algorithm. A training set of circuits is sampled from the original circuit by reducing the amount of magic (T gates), and then training data are produced using classical simulators and real quantum computers. Finally, the data are used to perform classical post-processing.

The degree of this tradeoff depends on several factors: the accuracy of the noise model learned from training data, the representativeness of the training circuits relative to the target circuit, and the stability of the mitigation map under statistical fluctuations. Recent theoretical analysis suggests that for optimized protocols, the residual bias scales favorably with circuit size ($\sim \sqrt{N}$), making error mitigation increasingly beneficial for larger quantum circuits, despite the variance penalty (see "*Bias in the global depolarising model*" section of [86]).

3.4 Real-Time Orchestration for Real-World Quantum Devices

Working with real quantum devices involves challenges at different levels. Even when noise is relatively low, operations require calibration, tuning, and careful control. When noise becomes more pronounced, these tasks interact in non-trivial ways and directly affect the outcome of computations.

At the same time, this complexity is also an opportunity. It motivates the development of protocols that link classical and quantum resources in real time, reducing latency and enabling fast feedback. Real-time error mitigation is one such approach, and it already shows promise in variational tasks [9]. Looking ahead, the same principles can extend to real-time error correction, where continuous adaptation will be even more critical.

Another important aspect is orchestration. Calibration, characterization, and device tuning are essential for reliable operation. These routines are usually handled separately, but in practice they must be integrated into the broader pipeline that runs algorithms. In the future, we may see low-level operations become part of high-level workflows, for example by embedding calibration or noise characterization directly into the training

loop of a machine learning algorithm.

Overall, the management of noise and control in quantum devices opens several directions for research. It requires technical advances in hardware and infrastructure, but also new algorithmic strategies that adapt to the device as it operates. This dual perspective will be central in the transition from proof-of-principle experiments to practical quantum computing.

Part II

A Full-Stack Open-Source Quantum Operating System

CHAPTER 4

Classical Simulation of Quantum Computers

Alongside the development of quantum technologies and the exploration of algorithms and solution seeking for a quantum advantage, classical computers still play a crucial role when implementing techniques to simulate quantum systems. In fact, classical simulation of quantum computers is not only useful for validating algorithms and protocols while the quantum devices are noisy and limited, but they are also extremely competitive in specific scenarios. The following sections present the challenge of simulating quantum computing operations on a classical device, followed by an introduction to `Qibo`, an open-source quantum computing simulator. Section 4.3 then discusses alternative approaches to classical simulation of quantum systems in more detail.

4.1 State-Vector Simulation

When simulating quantum circuits on a classical computer, the most straightforward approach is to represent the quantum state as a vector and apply unitary operations using standard linear algebra. For a system of n qubits, the quantum state $|\psi\rangle$ can be written as a complex vector with $N = 2^n$ components, and any quantum gate acting on $t \leq n$ qubits corresponds to a $2^t \times 2^t$ unitary matrix.

To simulate the effect of a gate acting on a subset of qubits, we conceptually split the full n-qubit system into two parts: *i)* $\boldsymbol{\tau}$, a bitstring of length t, representing the qubits that the gate acts on; and *ii)* $\boldsymbol{q}$, a bitstring of length $n - t$, representing the remaining qubits.

The action of the gate updates the quantum state as:

$$|\psi'(\boldsymbol{\tau}, \boldsymbol{q})\rangle = \sum_{\boldsymbol{\tau}'} U(\boldsymbol{\tau}, \boldsymbol{\tau}') \, |\psi(\boldsymbol{\tau}', \boldsymbol{q})\rangle \, , \tag{4.1}$$

where the sum runs over all possible bitstrings $\boldsymbol{\tau}'$ of length t.

This expression means that to compute the updated amplitude for the configuration $(\boldsymbol{\tau}, \boldsymbol{q})$, we apply the unitary matrix U to the relevant t-qubit subsystem (indexed by $\boldsymbol{\tau}$), while keeping the rest of the system (indexed by $\boldsymbol{q}$) unchanged. The simulation proceeds by applying such updates sequentially for each gate in the quantum circuit.

Although this simulation technique is exact and very intuitive, it carries a big cost in terms of memory consuption. In fact, as discussed in Section 1.3, any classical (even super) computer struggles in representing more than some dozens of qubits [20]. Even restricting ourselves to system of qubits of size $n < 30$, which is typically executable on a personal computer, the cost of simulating a sequence of gates as shown in Equation (4.1) scales in time and memory as $\mathcal{O}(2^n)$.

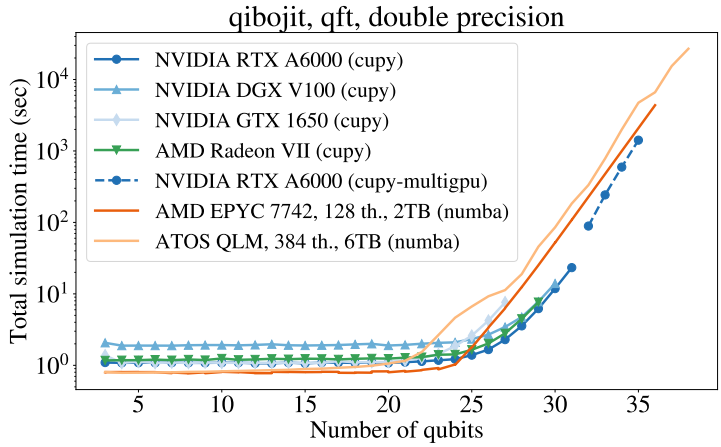


Figure 4.1: Total simulation time for executing the quantum Fourier transform algorithm [1] on different devices using the `Qibo`'s just-in-time compiled backends provided by `Qibojit` [2].

This is indeed the behavior one can evince when trying to simulate quantum circuits considering an increasing number of qubits. In Figure 4.1 is shown the time to execute a quantum Fourier transform [1] in double precision using a state-vector simulation backend of `Qibo` on different devices.

This limitation indeed pushes us to improve quantum technologies and to find alternative simulation techniques, or to develop smarter ways to perform this kind of simulation.

4.2 `Qibo`: an Open-Source Quantum Computing Simulator

The `Qibo` package was introduced in 2021 to provide a Python tool to simulate quantum circuits through state-vector simulation approach [3, 40]. The package is constructed to be modular, namely, once a quantum algorithm is written using `Qibo`'s application programming interface (API), the actual execution of the quantum circuits is delegated to a chosen *backend*.

A backend is a Python class implementing the operations necessary to perform the circuit execution using primitives of a third-part package. For example, a `NumPy` [87] backend is provided with the default `Qibo` installation and allows to execute circuits of size $n \leq 20$ in a moderate time.

`Qibo`'s architecture includes several specialized backends that cater to different simulation needs. The native backends are complemented by provider-specific implementations. Among the others, `Qibotn` enables tensor networks simulations, while `Qiboml` provides automatic differentiation capabilities using frameworks like `TensorFlow` [88], `Jax` [89], and `PyTorch` [90], making it particularly suitable for hybrid quantum-classical machine learning applications where quantum circuits parameters need to be optimized. Common applications include variational quantum algorithms and quantum neural networks, where classical optimization routines are combined with quantum circuit evaluations. For high-performance computing, `Qibojit` [2] offers just-in-time compiled executions on CPUs and GPUs, including multi-GPU support. Additionally, `Qibolab` [6] facilitates interfacing with real quantum hardware through self-hosted quantum computers. `Qibo` also supports execution on cloud-based quantum hardware, such as IBM

Quantum devices [91], AWS Braket [92], and IonQ [93], through dedicated backends.

Figure 4.2 provides a comprehensive overview of `Qibo`'s supported backends and their interactions within the ecosystem.

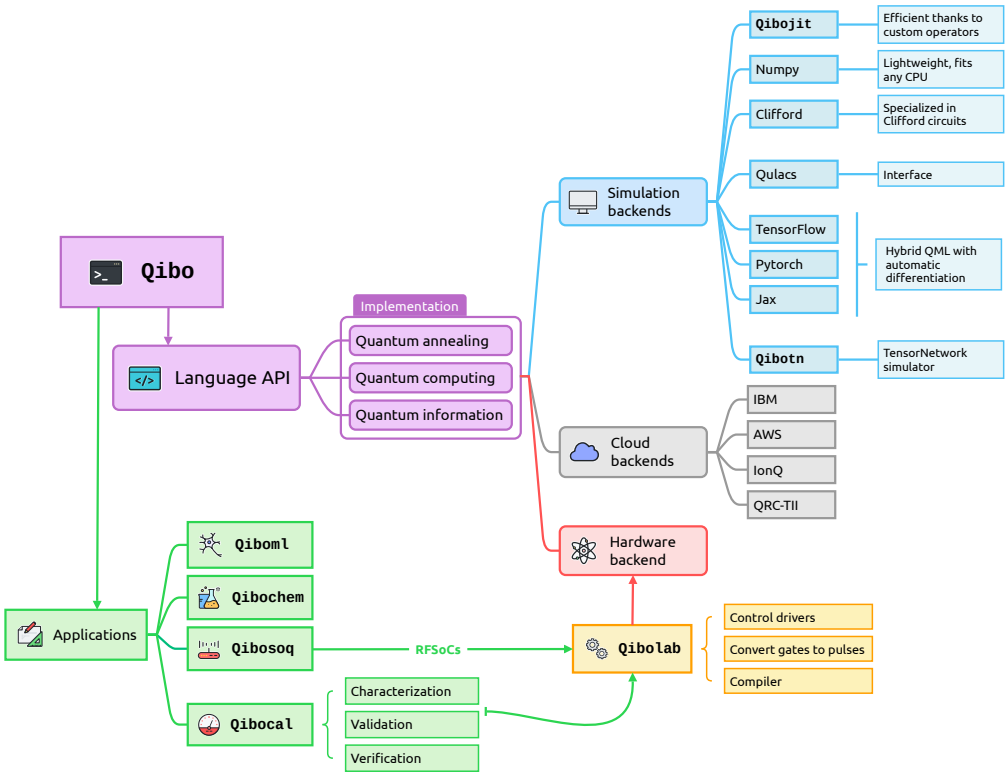


Figure 4.2: Schematic representation of the current status of the `Qibo` ecosystem.

`Qibo` implements the digital quantum computing paradigm presented in Chapter 1, but also consent the implementation of analog quantum computing simulations. More specifically, it provides adiabatic evolution [94] simulation tools, which rely on Susuki-Trotter (see time-evolving block decimation method in [95]) decomposition of the evolution unitary, constructing a digitalized (and arbitrary approximated) reprentation of the theoretical evolution. `Qibo` not only allows writing and executing algorithm with the two mentioned approaches, but is also equipped with pre-computed quantum computing routines such as the quantum Fourier transform [1], the Grover search algorithm [17] or variational quantum eigensolvers [21] among others.

Several `Qibo`-based applications have been developed to address specific computational challenges. `Qiboml` specializes in hybrid classical-quantum machine learning implementations and training [96], while `Qibochem` focuses on quantum chemistry applications [97]. For hardware control, `Qibosoq` [98] enables customized management of self-hosted devices through radio frequency system on chip (RFSoC) integration, and `Qibocal` [7] provides tools for quantum hardware characterization and calibration.

`Qibolab` serves as the bridge between `Qibo` and physical quantum hardware. It provides a comprehensive set of tools for controlling and interfacing with quantum comput-

ers, including pulse-level control, calibration routines, and hardware abstraction layers. The package implements low-level quantum control features such as pulse shaping, timing control, and readout optimization, making it essential for users who want to execute quantum algorithms on real quantum devices. When integrated with Qibocal, it forms a complete stack for quantum hardware control, from device calibration to experiment execution.

4.2.1 Accelerating State-Vector Simulation with Qibojit

Given the computational cost of executing state-vector simulation, Qibojit has been introduced in 2022 [2] to accelerate the simulation by means of *in-place* updates of the states, custom operators and just-in-time compilation.

Rather than copying states around as in NumPy's einsum, in-place updates directly modify the quantum state in memory. This is achieved through custom operators that implement alternative methods to avoid creating copies and, when needed, take advantage of sparse matrix operations, particularly useful for 1- and 2-qubit gates. These optimizations boost the simulation speed and let run circuits with around 30 qubits on a regular laptop.

Just-in-time (JIT) compilation is a runtime optimization technique that compiles frequently executed code sections into optimized machine code during program execution, rather than beforehand. The JIT compiler identifies computational "hot spots" and generates specialized machine code for these regions, caching the results for subsequent use. This approach combines the flexibility of high-level languages with near-native performance for repeated operations. In quantum circuit simulation, Qibojit applies JIT compilation to optimize gate operations, which constitute the dominant computational cost in state-vector simulation. Since gates are applied repeatedly throughout a circuit, the compilation overhead is amortized across many operations, resulting in significant performance improvements over interpreted implementations. As illustrated

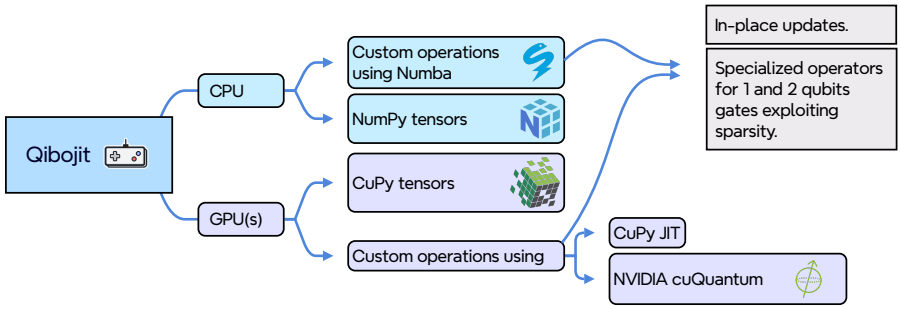


Figure 4.3: Schematic representation of Qibojit's engines.

in Figure 4.3, Qibojit implements two distinct acceleration strategies. For CPU operations, it enhances NumPy arrays using Numba [99] compilation. For GPU and multi-GPU operations, it leverages CuPy [100] primitives through two approaches: custom CUDA [101] kernels implemented in C++ via CuPy's RawKernel, and high-performance primitives from NVIDIA's cuQuantum library [5].

4.2.2 `Qibo` as Playground for Quantum-Classical Orchestration

As highlighted by the variety of backends and applications in the `Qibo` ecosystem, `Qibo` allows the user to choose the most suitable tool for their specific needs, while keeping the same interface. For example, one may want to keep unchanged its implementation of the QFT circuit mentioned in Figure 4.3, and test how a tensor network backend performs with respect to a state-vector one. In that case, the user will obtain similar results to the ones shown in Figure 4.4, where state-vector and tensor network backends are compared when executing the QFT circuit in double precision on various classical hardware.

`Qibo` is also designed to facilitate the integration of classical and quantum computing resources. This capability is particularly relevant in the context of hybrid quantum-classical protocols, where the strengths of both paradigms can be leveraged to tackle complex problems. More importantly, `Qibo` offers a completely open-source platform that allows researchers to experiment with and develop new methods involving any combination of classical and quantum resources.

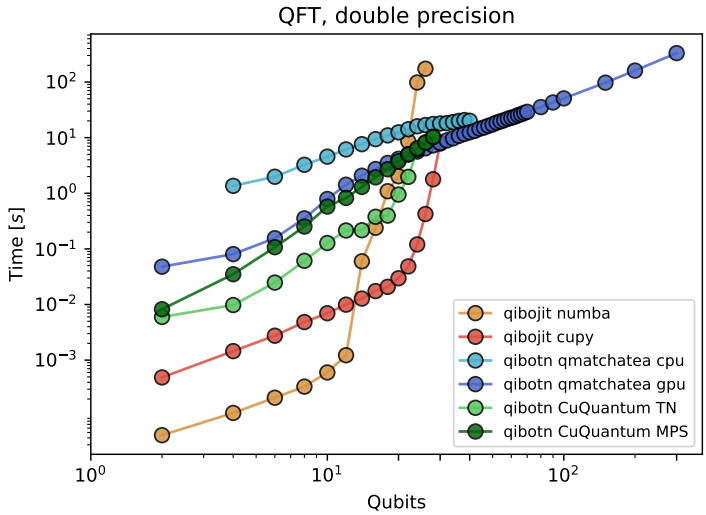


Figure 4.4: Total simulation time for executing the quantum Fourier transform algorithm [1] in double precision on different devices using the `Qibo`'s tensor network backend provided by `Qibotn` [3]. State-vector simulators are adopted through `Qibojit` [2], both on CPU (orange line) and GPU (red line). Tensor network simulations are performed using `Qibotn` as backend provider and `qmatchatea` [4] primitives (light blue and blue lines for CPU and GPU respectively), and NVIDIA's `cuQuantum` library [5] (light green for CPU and dark green for GPU). These results were produced when I was contributing to the `Qibotn` package, integrating a simulation backend based on `qmatchatea`.

This is why `Qibo` has been chosen as the main tool to implement and test the techniques presented in this thesis, and it is often referred to as the playground for quantum-classical orchestration in our discussion.

4.3 Alternative Classical Simulators

While state-vector simulation offers exact quantum state representation, its exponential scaling in both memory and computation time necessitates alternative approaches. Var-

ious specialized techniques have emerged, each optimized for specific classes of quantum systems and scenarios. As illustrated in Figure 4.5[1], these methods can be highly effective in their targeted domains, as briefly described in the following subsections.

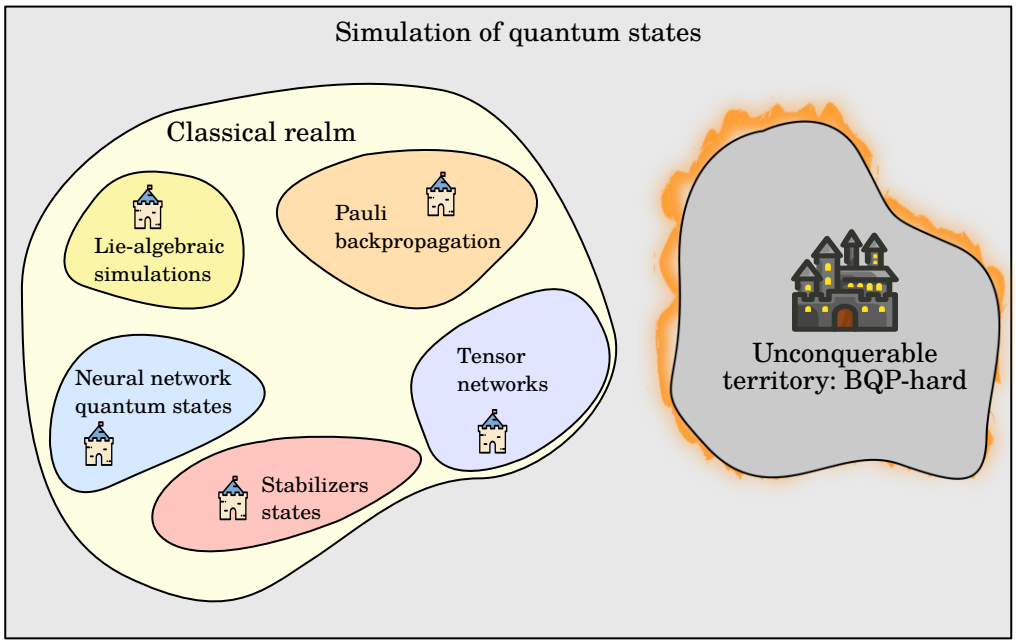


Figure 4.5: Schematic representation of various classical simulation methods for quantum systems, each of them is optimal within specific scenarios.

4.3.1 Tensor Networks

Tensor Networks are efficient tools to represent states and operations involving a limited amount of entanglement [103, 37]. They decompose the full quantum state into a network of lower-rank tensors, with the network structure reflecting the entanglement patterns in the system. Usually, tensor networks are intuitively represented as a graph, where nodes represent tensors and edges represent the connections between them. An illustrative example of possible building blocks is shown in Figure 4.6.

This same representation allows intuitive representation of the usual operations we perform on tensors, such as contraction, tracing, etc. An illustrative example of some of these operations is shown in Figure 4.7.

In this section, one of the simplest yet successful tensor network representations is introduced, namely the Matrix Product State (MPS) representation. We strongly recommend the interested reader to refer to the review [104] for a more comprehensive introduction to the topic.

MPS is particularly effective for representing one-dimensional quantum systems, where the entanglement entropy typically obeys an area law. To construct an MPS rep-

[1] About the figure notation: a problem is BQP-hard if it is at least as difficult (under efficient reductions) as the hardest problems that a bounded-error quantum computer can solve in polynomial time (the class BQP). See [102] for more details.

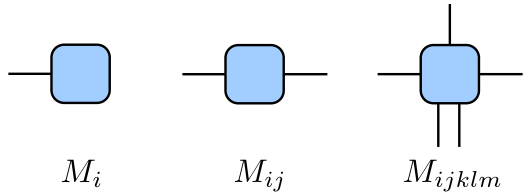


Figure 4.6: Example of possible building blocks of a tensor network. From left to right we have representations of a vector, a matrix and a tensor of rank five.

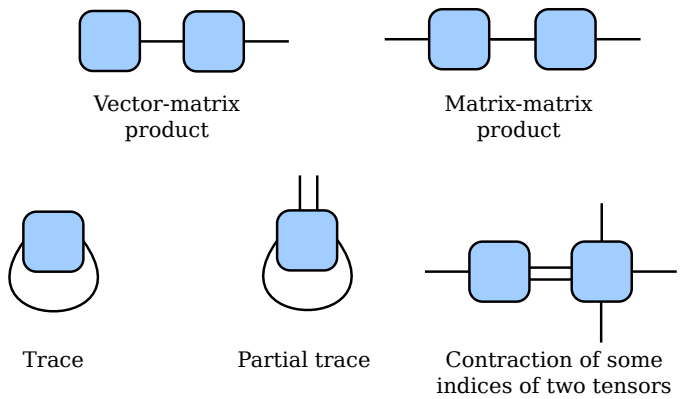


Figure 4.7: Some examples of possible operations on tensors represented as networks.

resentation we can start from a general state of n qubits

$$|\psi\rangle = \sum_{i_1,\ldots,i_n} c_{i_1,\ldots,i_n} |i_1, i_2, \ldots, i_n\rangle \,, \tag{4.2}$$

where the tensor $c_{i_1,\ldots,i_n}$ represents the amplitudes of the state $|\psi\rangle$ in the computational basis. This same tensor, in a state-vector simulation, is represented as a 2^n-dimensional vector of complex numbers. Starting from this, the building block of the tensor network representation is the fact that we can manipulate the shape of the tensor $c_{i_1,\ldots,i_n}$ using index grouping and singular value decomposition (SVD). In particular, SVD allows us to decompose a tensor M into a product of matrices $U\Sigma V^\dagger$ such that U and V are unitary matrices and Σ is a diagonal matrix containing the singular values. A schematic representation of the SVD decomposition is shown in Figure 4.8.

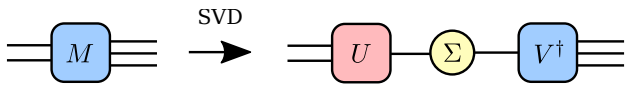


Figure 4.8: Schematic representation of the SVD decomposition of a matrix M into a product of matrices U, Σ and $V^\dagger$.

By doing so, we decomposes an n-qubit state as:

$$|\psi\rangle = \sum_{i_1,\ldots,i_n} \mathrm{Tr}(A^{[1]}_{i_1} A^{[2]}_{i_2} \cdots A^{[n]}_{i_n}) \, |i_1, i_2, \ldots, i_n\rangle \,, \tag{4.3}$$

where each $A_{i_k}^{[k]}$ is a rank-3 tensor associated with qubit k. Each of these tensors has a physical index i_k (with dimension $d = 2$ for qubits) and two auxiliary indices that connect to neighboring tensors in the network. The auxiliary indices represent the bond dimensions.

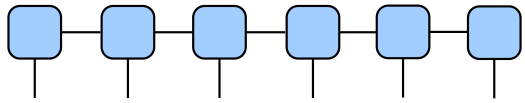


Figure 4.9: Schematic representation of a Matrix Product State (MPS) for six qubits. Each tensor in the middle of the chain has a physical index (the vertical leg) and two auxiliary indices, representing the bond dimensions.

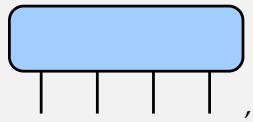
The dimension of these matrices is determined iteratively applying the SVD during the representation process and it can be described in terms of *bond dimension*, which is the maximum dimension of the matrices $A_{i_k}^{[k]}$. The bond dimension is an important parameter that controls the level of approximation in the MPS representation, and so the amount of entanglement that can be captured. An example of how to write the MPS representation for a state of four qubits is illustrated in the following box.

MPS representation of a 4-qubit system

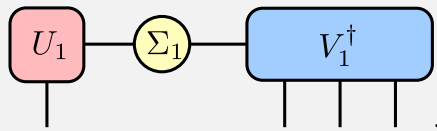
We consider a system of four qubits, whose full state can be represented as a vector of 16 complex numbers, *i.e.* C_{i_1,i_2,i_3,i_4}. As introduced in Equation (4.2), we can write this state as:

$$|\psi\rangle = \sum_{i_1,i_2,i_3,i_4} C_{i_1,i_2,i_3,i_4} |i_1\rangle \otimes |i_2\rangle \otimes |i_3\rangle \otimes |i_4\rangle \,, \tag{4.4}$$

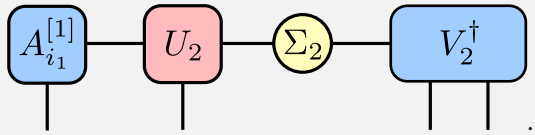
where the rank-4 tensor C_{i_1,i_2,i_3,i_4} represents the amplitudes of the state $|\psi\rangle$ in the computational basis. In usual state-vector simulation, this information is stored into a vector of shape $(16,)$.
Starting from an initial general state that we can represent as



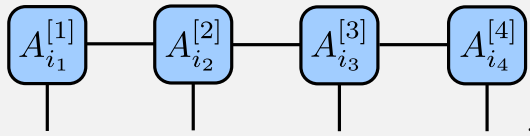
we reshape the tensor into a matrix of shape $(2, 8)$, preparing the tensor for SVD decomposition and isolating the first physical index. We thus decompose the matrix using SVD as $U_1 \Sigma_1 V_1^\dagger$, with U_1 and V_1 being unitary matrices and Σ_1 a diagonal matrix containing the singular values:



Since we start from a matrix of shape $(2, 8)$, the resulting matrices will have shapes: $U_1 : (2, 2)$, $\Sigma_1 : (r_1, r_1)$ and $V_1 : (8, 2)$, with $r_1 = 2$.
In the previous step we have introduced a new index r_1, which is used to restrict Σ_1 to only non-zero singular values.
We keep apart U_1 and Σ_1, and those are going to compose the first tensor in the MPS representation, *i.e.* $A^{[1]}_{i_1}$ of Equation (4.3).
Now we start with a new SVD. We take the matrix $V_1^\dagger$, which for now has shape $(2, 8)$, and group the bond indices with the ones of the second qubit, obtaining a matrix of shape $(4, 4)$. We decompose this matrix using SVD as $U_2\Sigma_2V_2^\dagger$, and this time the matrices will have shapes $U_2 : (4, 2)$, $\Sigma_2 : (r_2, r_2)$ and $V_2 : (4, 4)$, with $r_2 = 4$:



As previously done, U_2 and Σ_2 are kept apart, and they will compose the second tensor in the MPS representation, *i.e.* $A^{[2]}_{i_2}$. We repeat this process until we reach the last qubit, obtaining $A^{[3]}_{i_3}$ and $A^{[4]}_{i_4}$:



The number of bonds in the MPS representation is bounded by the lower dimension of the matrices obtained during the SVD decomposition, and it is intuitive to see that it increases exponentially with the number of qubits in the center of the chain.

In the previous box it is shown how to write a full MPS representation of a system of qubits, when we keep all the non-zero singular values. Although this is a valid representation, it is not always necessary to keep all the values. In fact, these bonds are representing the entanglement between the qubits, and in many cases it is possible to truncate the bond dimension to a smaller value, which is often sufficient to represent the state with a good approximation. This is particularly true for states with low entanglement, such as ground states of one-dimensional quantum systems. In these cases, the bond dimension can be reduced to a small value without significantly affecting the accuracy of the representation.

In any case, the bond dimension χ of a representation is a parameter that can be tuned to balance the accuracy of the representation and the computational cost. The computational complexity scales with the bond dimension χ of the network, typically as $\mathcal{O}(n\chi^3)$ for MPS operations, where n is the system size. However, their efficiency

breaks down for states with volume-law entanglement, requiring exponentially large bond dimensions to maintain accuracy.

Several software packages implement tensor network methods, including `Quimb` [105], `qmatchatea` [4] and `TeNPy` [106] in Python, and `ITensor.jl` [107] for Julia and C++.

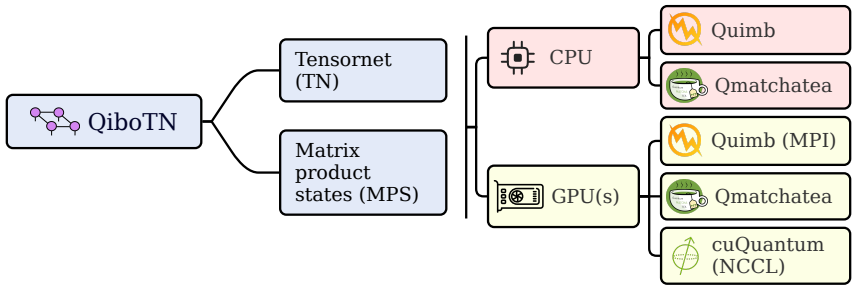


Figure 4.10: Schematic representation of the `Qibotn` available backends.

In `Qibo`, we have implemented a tensor network backend provider, `Qibotn`, which wraps around some of the aforementioned packages, allowing the user to choose the most suitable one for their specific needs, while keeping the same interface. A diagram of the `Qibotn` available backends is shown in Figure 4.10.

4.3.2 Clifford and Quasi-Clifford Simulation

Classical simulation of quantum circuits typically requires exponential resources, but when circuits consist primarily of Clifford gates with few non-Clifford operations, we can exploit special mathematical structure to achieve polynomial-time simulation. Clifford [36] and quasi-Clifford [108] methods leverage the stabilizer formalism, which represents quantum states through a set of commuting Pauli operators that "stabilize" the state. A stabilizer state $|\psi\rangle$ is the unique common eigenstate of a set of commuting Pauli operators $\{S_1, S_2, \ldots, S_k\}$, meaning $S_i|\psi\rangle = s_i|\psi\rangle$ where $s_i \in \{+1, -1\}$ are the eigenvalues. The key insight is that Clifford gates preserve the stabilizer structure: if P is a Pauli operator and C is a Clifford gate, then $CPC^\dagger$ is also a Pauli operator (up to a global phase). Importantly, Clifford operations can change both the Pauli operators and their eigenvalues, but the overall stabilizer structure is preserved. This closure property enables classical tracking of stabilizer evolution without storing exponentially large state vectors.

The widely-used Aaronson-Gottesman tableau representation [36] encodes stabilizer generators for an n-qubit system in a $(2n \times 2n)$ binary matrix plus phase information. Each row represents a stabilizer generator, with the first n columns encoding X operators and the next n columns encoding Z operators, plus a phase bit tracking the ± 1 eigenvalue of that stabilizer on the current state.

Stabilizer tableau for Bell state $|\Phi^+\rangle$

Consider the 2-qubit Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, which is stabilized by XX and ZZ with eigenvalues $+1$. The tableau representation encodes these stabiliz-

ers as:

X_1	X_2	Z_1	Z_2	phase
1	1	0	0	0
0	0	1	1	0

(4.5)

When applying Clifford gates, both the Pauli structure and eigenvalues can change. For example, applying Z_1 would flip the phase of the first stabilizer (changing XX to $-XX$), while Hadamard and CNOT gates update the tableau through row operations that preserve the stabilizer relationships.

For pure Clifford circuits, simulation complexity scales as $\mathcal{O}(n^2)$ per gate operation versus $\mathcal{O}(2^n)$ for state-vector simulation, providing exponential speedup that makes simulation feasible for hundreds of qubits. When non-Clifford gates (such as T gates) are present, the stabilizer formalism cannot represent the state exactly, as these gates create superpositions beyond stabilizer states. Quasi-Clifford methods address this by approximating the state as a weighted sum $|\psi\rangle = \sum_{i=1}^{r} \alpha_i |\psi_i\rangle$ of r stabilizer states. The simulation cost becomes $\mathcal{O}(r \cdot n^2)$, where r grows exponentially only in the number of non-Clifford gates, making these methods practical for circuits with sparse non-Clifford operations. Implementations include `Stim` [109] for Clifford simulation and `PauliBackpropagation.jl` [110] for quasi-Clifford methods, forming the basis for efficient simulation of quantum error correction codes and quantum algorithms with limited non-Clifford resources.

In `Qibo`, we have implemented a Clifford backend, which allows the user to efficiently simulate circuits composed of Clifford gates [111].

4.3.3 Neural Network Quantum States

Neural network quantum states (NNQS) provide a variational framework to approximate quantum states using machine learning architectures [112]. These approaches parameterize quantum wavefunctions using neural networks such as restricted Boltzmann machines (RBMs) [113], deep feedforward networks [114], convolutional neural networks [115], or transformer architectures [116].

The quantum wavefunction is parameterized as $\psi(x) = \mathcal{N}_\theta(x)$, where $\mathcal{N}_\theta$ is a neural network with trainable parameters θ, and x represents a computational basis state. For an n-qubit system, the network maps binary configurations $x \in \{0,1\}^{n}$[2] to their corresponding complex amplitudes $\psi(x)$.

The original NNQS approach used RBMs, which define the wavefunction through marginalization over hidden units:

$$\psi(x) = \sum_h e^{\sum_i a_i x_i + \sum_j b_j h_j + \sum_{ij} W_{ij} x_i h_j} \tag{4.6}$$

where x_i are visible units (physical qubits), $h_j \in \{0,1\}$ are binary hidden units, and $\{a_i, b_j, W_{ij}\}$ are trainable parameters. The hidden layer captures quantum correlations beyond simple product states.

[2] We use the digital quantum computing convention, but the approach generalizes to other basis representations.

NNQS are optimized using variational Monte Carlo (VMC) to minimize the energy expectation value $\langle H \rangle = \langle \psi | H | \psi \rangle / \langle \psi | \psi \rangle$ for a given Hamiltonian H. The process involves sampling basis states x with probability $|\psi(x)|^2$ using Markov chain Monte Carlo (MCMC), computing local energy estimates, and performing gradient-based parameter updates. The local energy is defined as:

$$E_{\text{loc}}(x) = \sum_{x'} \frac{H_{x,x'} \psi(x')}{\psi(x)} \tag{4.7}$$

For an RBM with M parameters, each network evaluation scales as $\mathcal{O}(M)$, while full training requires $\mathcal{O}(M \cdot N_{\text{samples}} \cdot N_{\text{epochs}})$ operations. Although modern architectures may contain millions of parameters, this polynomial scaling remains tractable compared to exponential exact methods.

While NNQS cannot represent arbitrary quantum states exactly, they excel at capturing structured correlations in ground states of local Hamiltonians with area-law entanglement, states with built-in symmetries (translational, rotational, etc.), and quantum many-body systems near critical points where correlations are strong but organized.

Recent advances include autoregressive models for direct sampling from $|\psi(x)|^2$ [117] and transformer-based networks that capture long-range correlations more effectively than traditional RBMs [118].

4.3.4 Lie-Algebraic Methods

Lie-algebraic simulation, known as $\mathfrak{g}$-sim, enables efficient classical simulation of certain quantum circuits [119]. The key idea is simple: instead of tracking exponentially large quantum states, we track how a small number of observables evolve over time.

The method works when quantum circuits have polynomial-sized dynamical Lie algebras (DLAs). Consider a parametrized quantum circuit:

$$U(\theta) = \prod_{l=1}^{L} e^{-i\theta_l H_l}, \tag{4.8}$$

where H_l are gate generators (like Pauli operators) and L is the circuit depth. The DLA $\mathfrak{g}$ contains all nested commutators of these generators. For example, starting with $H_1 = X_1$ and $H_2 = Z_1 Z_2$, we compute $[H_1, H_2] = 2iY_1 Z_2$, then $[H_1, [H_1, H_2]]$, and so forth until no new independent operators appear.

Most quantum circuits generate exponentially large algebras, but special cases exist where $\dim(\mathfrak{g}) = d \in \mathcal{O}(\text{poly}(n))$ for n qubits. Examples include the transverse-field Ising model ($d = n^2$), systems with particle number conservation, and certain QAOA instances.

The simulation algorithm tracks expectation values instead of state amplitudes. Let $\{h_\alpha\}_{\alpha=1}^{d}$ be an orthonormal basis for the DLA $\mathfrak{g}$. For any observable $O = \sum_\alpha w_\alpha h_\alpha$ where w_α are real coefficients, we start with initial expectation values with respect to the initial state $\rho^{(0)}$:

$$(e^{(0)})_\alpha = \text{Tr}[h_\alpha \rho^{(0)}]. \tag{4.9}$$

The crucial insight comes from using the cyclic property of the trace. When we apply a unitary U to transform the state, we can shift the evolution onto the observable: $\text{Tr}[h_\alpha U \rho^{(0)} U^\dagger] = \text{Tr}[U^\dagger h_\alpha U \rho^{(0)}]$. The key breakthrough is the *adjoint identity*, which tells

us that when $U = e^{-i\theta h_\mu}$ is generated by a DLA element $h_\mu \in \mathfrak{g}$, the transformed observable stays within the DLA:

$$e^{i\theta h_\mu} h_\alpha e^{-i\theta h_\mu} = \sum_\beta e^{-i\theta f^\mu_{\alpha\beta}} h_\beta, \tag{4.10}$$

where $f^\mu_{\alpha\beta}$ are structure constants defined by $[h_\alpha, h_\beta] = \sum_\gamma f^\gamma_{\alpha\beta} h_\gamma$.

This means expectation values evolve linearly. Under a single gate, they transform as:

$$(e^{(1)})_\alpha = \sum_\beta e^{-i\theta f^\mu_{\alpha\beta}} (e^{(0)})_\beta. \tag{4.11}$$

For multi-gate circuits, this becomes a product of matrix exponentials:

$$e^{(\text{out})} = \left(\prod_{l=1}^{L} e^{-i\theta_l \bar{H}_l} \right) e^{(\text{in})}, \tag{4.12}$$

where $e^{(\text{in})} \equiv e^{(0)}$, $e^{(\text{out})}$ is the final expectation vector, and $\bar{H}_l$ are $d \times d$ matrices with entries $(\bar{H}_\mu)_{\alpha\beta} = f^\mu_{\alpha\beta}$ - the structure constants of the algebra.

The final expectation value is:

$$\langle O(\theta) \rangle = w^T \cdot e^{(\text{out})}. \tag{4.13}$$

This transforms quantum simulation into classical linear algebra. While standard simulation requires $\mathcal{O}(2^n)$ memory and time, $\mathfrak{g}$-sim requires $\mathcal{O}(d^2)$ memory to store the adjoint representations and $\mathcal{O}(Ld^3)$ time for matrix exponentiations over L layers. When $d \in \mathcal{O}(\text{poly}(n))$, this provides exponential speedup.

For the transverse-field Ising model on n qubits, $d = n(2n-1)$, so memory scales as $\mathcal{O}(n^4)$ and time as $\mathcal{O}(Ln^6)$: polynomial rather than exponential. This enables exact classical simulation of variational circuits with polynomial-sized algebras, as detailed in Table 4.1.

The method is particularly valuable for pre-training variational algorithms, understanding barren plateaus, and identifying the quantum-classical computational boundary. However, it is restricted to the special class of circuits whose generators form polynomial-dimensional Lie algebras, representing a complementary approach to other classical simulation techniques.

4.4 Summary of Classical Simulation Complexities

To conclude, Table 4.1 summarizes the computational complexity and typical applicability of the main classical simulation techniques discussed in this chapter.

As shown in this chapter, while state-vector simulation is the most general simulation method, its exponential scaling limits its applicability to small systems. Alternative methods such as tensor networks, Clifford/quasi-Clifford techniques, neural network quantum states, and Lie-algebraic approaches provide polynomial-time simulation for specific classes of quantum systems. The choice of method depends on the structure of the quantum circuit, the amount of entanglement, and the presence of symmetries or constraints. These specialized techniques enable the study of larger systems and more complex phenomena than would be feasible with brute-force state-vector methods alone.

Classical Simulation Methods			
Method	**Memory**	**Time**	**Notes**
State-vector	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$	Universal, exact; $n \lesssim 30$
Tensor Net (MPS)	$\mathcal{O}(n\chi^2)$	$\mathcal{O}(n\chi^3)$	1D, low entanglement
Clifford	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	Clifford-only circuits
Quasi-Clifford	$\mathcal{O}(rn^2)$	$\mathcal{O}(rn^2)$	Few non-Clifford gates
NNQS	$\mathcal{O}(M)$	$\mathcal{O}(MSE)$	VMC, ML-based
Lie-Algebraic	$\mathcal{O}(d^2)$	$\mathcal{O}(d^3)$	Polynomial algebra dim.

Table 4.1: Summary of computational complexities and applicability for classical simulation methods. n: number of qubits, χ: bond dimension, r: stabilizer rank, M: number of parameters in the neural network, S: samples, E: epochs, d: dimension of the Lie algebra.

We would like to mention that these methods can be seen as complementary rather than competitive. In fact, hybrid approaches that combine multiple techniques are an active area of research. This same idea falls into the scope of this thesis, where we explore hybrid quantum-classical orchestration strategies that leverage the strengths of both quantum hardware and classical simulation methods to tackle complex problems more efficiently. A practical example of how combining different simulation methods can be beneficial is presented in Chapter 9, where we present a software package that implements an hybrid tensor network and stabilizer simulation approach.

CHAPTER 5

`Qibolab`: Controlling Quantum Devices

From a theoretical perspective, Chapter 1 and Appendix A introduce a universal computing paradigm, which consent implementing any operation on a system of qubits. In the most-widely used model of quantum computation, digital quantum computing, we can describe any reversible operation in terms of quantum gates, and we can access this information through measurements. From a more practical perspective, we have seen there exist programming tools to facilitate the implementation of quantum algorithms. Among others, we focused on `Qibo`, which offers a simple Python interface to the digital quantum computing paradigm. Still, `Qibo` alone is opening the door to quantum computing but, in order to execute algorithms on a real quantum device, an extra layer of abstraction is needed. This is where `Qibolab` comes into play. In this chapter `Qibolab` is presented: an open-source framework for quantum hardware control.

> This chapter follows the structure and work from Reference [6].
>
> The `qibolab` package is open-source: https://github.com/qiboteam/qibolab.

5.1 Open-Source Middleware for Quantum Computing

With *middleware* we refer to a software layer that acts as a bridge between different applications, systems or components, enabling them to communicate and share data despite being built with different technologies.

The need of a robust and open-source middleware is particularly relevant in the quantum computing context, where classical computers host the high-level interface of any application and we need to deal with several tools, instruments and technological layers [120, 121].

This is even highlighted if we consider the idea, core-business of this thesis, of exploring and optimizing the full pipeline, orchestrating classical and quantum parts composing this hybrid setup. An illustrative diagram of this working context is provided in Figure 5.1.

`Qibolab` was introduced to contribute filling this gap, in a context where other interesting works have been done, such as `QCOdes` [122], `pycqed` [123], and `Labber` [124]. Those are effective in their contexts, but still not flexible enough to offer all the tools and features necessary to effortlessly incorporate all the necessary features that a full-stack solution requires.

Moreover, `Qibolab` is designed to be perfectly compatible with the high-level language `Qibo`, which is used to formalize the quantum algorithms, and with `Qibocal`,

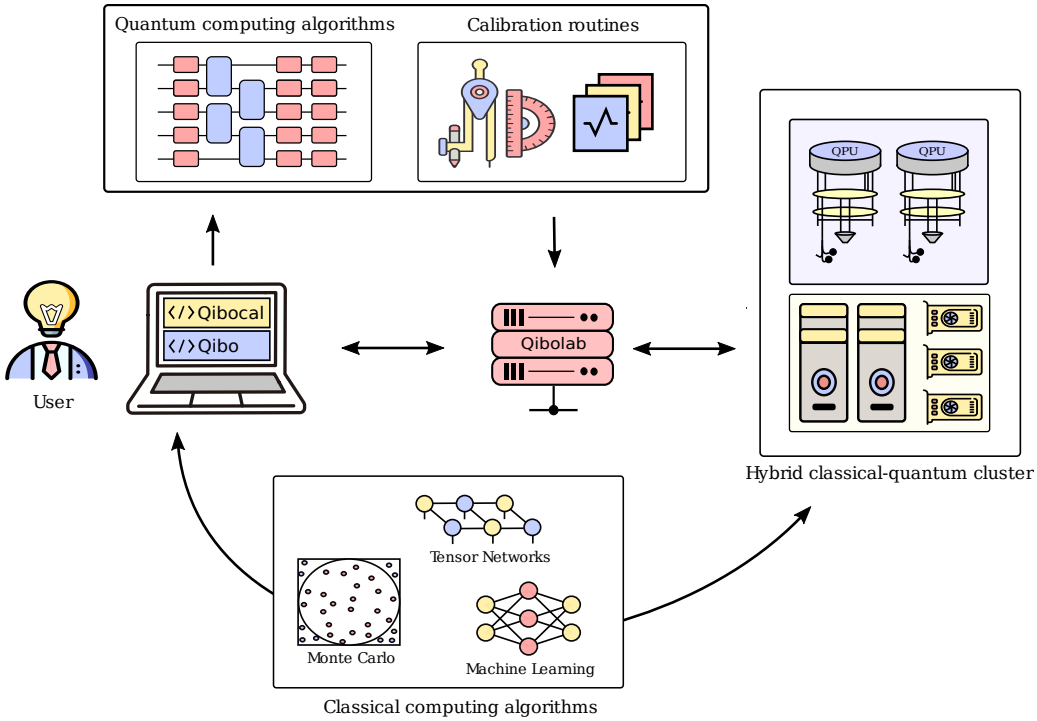


Figure 5.1: Overview of the full-stack hybrid setup within which `Qibolab` contributes as middleware building block. What is conceived as an algorithm from a user perspective, requires orchestration of several components: an high-level language to formalize the algorithm (`Qibo`), some classical nodes to execute parts of it, some quantum node to execute other parts. The whole orchestration is possible thanks to software layers which consent controlling (`Qibolab`) and calibrating (`Qibocal`) the quantum devices.

another open-source software for characterization and calibration of quantum devices. The three of them, together with all the application packages which populate the `Qibo` ecosystem, form a complete and variegated stack for quantum computing.

5.2 Software Design

In this section, we provide a description of how `Qibolab` is structured and designed to deploy quantum algorithms on superconducting quantum devices. We stress that, although the original design of `Qibolab` was tailored to this specific use case, the framework is flexible and modular enough to be adapted to other quantum computing platforms and technologies, such as trapped ions or neutral atoms among others.

`Qibolab`'s interface comprises two main components: the *platform API* and the *experiment API*. The first can be used to prepare an abstract representation of the laboratory setup, including qubits, instruments and connections. Once a platform is defined, the second can be used to arrange an experiment, preparing the sequence of operations to be executed on (and by) the platform.

5.2.1 Experiment API

The Qibolab experiment API provides a set of pulses that can be composed into pulse sequences implementing experiments on the quantum device. Qibolab abstracts the low-level details of hardware control we discussed in Section 3.1.2.

A Pulse encodes the parameters required to generate a physical waveform, such as amplitude, frequency, phase, start time, and duration. Different pulse types are available to implement common operations, including readout, single-qubit rotations, flux tuning, and two-qubit interactions. Waveforms of various shapes (*e.g.* rectangular, Gaussian, DRAG) are supported. Multiple pulses can be combined into a PulseSequence, with flexible scheduling that allows overlapping signals, enabling features such as multiplexed readout.

Once defined, pulse sequences can be deployed on hardware through a Platform, which orchestrates the instruments connected to a given quantum chip. The platform supports execution of single sequences, batches of sequences, or parameter sweeps directly on the control electronics. These capabilities are particularly useful for calibration and characterization, where fast repetition and real-time parameter updates are essential.

5.2.2 Platform API

A Qibolab's platform is an abstract representation of a quantum computing setup and can be composed using a variety of building blocks. A schematic overview of the platform architecture is provided in Figure 5.2.

Qubit. The Qubit objects are representation of the physical qubits of the device. They contain information about their parameters, such as frequency, anharmonicity, or coherence times introduced in Section 3.2.1. These parameters are obtained through a series of characterization and calibration experiments which can be executed using Qibocal, as described in Chapter 6.

QubitPair. The QubitPair objects represent neighboring pairs of qubits in the chip. The topology of a device can be extracted from the Qubit and QubitPair objects and it is used by Qibo to perform transpilation procedures.

Instrument. The Instrument objects can be used to mirror the physical instruments used to control the quantum devices. They contain low-level drivers to match the providers' APIs. For several of the most commonly used manufacturer, Qibolab offers a pre-coded set of drivers implementations.

Channel. The Channel objects represent the communication channels between qubits and instruments. Through the Port objects, they also consent operating on instrument parameters. The channels are used to route the signals from the instruments to the qubits and vice versa.

To operate with Qibolab, one needs to define a platform which accurately represents the physical quantum computing setup. This can be done as follows:

1. add Instrument objects for each physical instrument in the setup;
2. add Qubit objects for each qubit in the device;

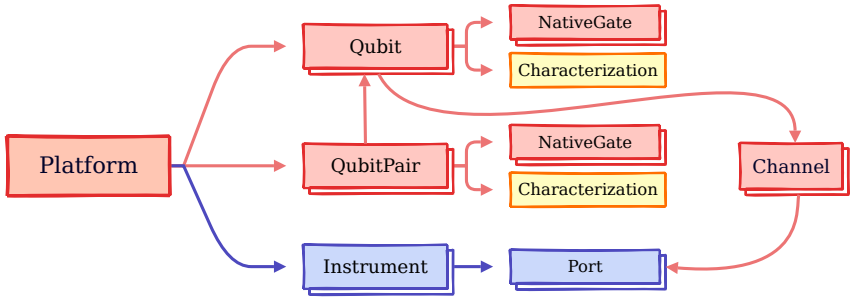


Figure 5.2: Hierarchy of the objects composing the `Qibolab`'s platform. Image reproduced from [6].

3. define the connectivity between qubits and instruments using `Channel` objects and map the connections to the corresponding instrument `Port`s;
4. assign the applicable channels (readout, feedback, drive, flux, etc.) to the qubits according to their roles in the experiment.

When defining a `Qibolab` platform, it is important to distinguish between *static* and *dynamic* parameters. Static parameters include, for example, the mapping between qubits, channels, instruments, and their IP addresses. These are typically fixed for a given laboratory setup and are hard-coded during the platform generation. In contrast, dynamic parameters (such as amplitudes, phases or duration of the pulses submitted through a drive line) evolve during calibration routines and must therefore be handled separately. In practice, dynamic parameters are loaded as external data, and can be updated and serialized to disk after calibration, ensuring that the platform reflects the most recent device characterization. The Platform interface abstracts the process of uploading these parameters to the respective instruments through their vendor-specific APIs.

The execution of a program can be implemented in two ways: a high-level approach consists in using the `Qibo` Circuit API, while a low-level approach consists in using the `Qibolab` Pulse API. The second approach provides finer control over the timing and shape of the pulses applied to the qubits, and it is particularly useful for tasks that require precise control over the pulse sequences (characterization and calibration). The first approach, instead, can be adopted for higher-level tasks, where the focus is on the algorithmic aspects rather than the low-level pulse details. Even if the circuit interface is chosen, the theoretical operations must still be mapped to the underlying pulse sequences. To do so, `Qibo` first uses the connectivity information reconstructed from the `QubitPair`s to transpile the original circuit into one which respects the connectivity. Secondly, the gates are compiled into native gates supported by the device, which are then mapped to the corresponding pulse sequences.

This flexible execution flow is particularly useful in hybrid quantum–classical computing, where different parts of the computation may require different levels of control and abstraction. It also enables higher-level applications: a quantum machine learning user may design a hybrid architecture using `Qiboml` and execute it on a quantum device through `Qibo` and `Qibolab` without modifying the original design. On the same device, an alternative choice could be to train pulse parameters directly with a machine learning routine. In this sense, `Qibolab` provides the flexibility to move seamlessly between algorithmic execution and pulse-level optimization within a unified framework.

The Qibolab port abstraction ensures compatibility between the compiled pulse sequences and the input/output formats of the specific devices. Available result formats include binary outcomes (shots), integrated or demodulated voltage signals, and raw waveform data. Each format can be returned either as single-shot measurements or averaged values, depending on the experimental requirements. This design allows Qibolab to support both algorithmic applications and fine-grained experimental control in a coherent way.

5.3 Supported Drivers

Qibolab is designed to be modular and extensible, allowing easy integration of new instruments and technologies. The framework currently supports a variety of control electronics commonly used in superconducting quantum computing setups. These include Qblox [125], Quantum Machines [126], Zurich Instruments [127] and self-installed radio frequency system on chip FPGAs (RFSoCs) supported by Qick [128] and Qibosoq [41]. A more detailed description of the supported drivers by Qibolab 0.1.0 follows.

Qblox The support is centered on the modular Qblox Cluster, which can host up to 20 instruments in a 19" rack. In our setup, Qibolab controls superconducting qubits using QRM-RF modules for readout [129], QCM-RF modules for microwave drive [130], and QCM modules for flux control [131], all synchronized via the SYNQ protocol [132]. High-level access is provided through the qblox-instruments [133] and QCoDeS libraries [134], while low-level sequencing relies on Q1ASM [135]. This configuration enables control of multiple flux-tunable qubits with scalable synchronization.

Quantum Machines Qibolab integrates with clusters of OPX+ [136] controllers, each offering multiple analog and digital I/O ports and synchronized through OPT devices. The driver leverages the QUA programming language [137], which exposes low-level pulse scheduling, conditional logic, and feedback operations. While OPX+ controllers require external mixers and local oscillators due to limited internal upconversion and bandwidth, they provide powerful real-time feedback and support for large multi-qubit systems.

Zurich Instruments Qibolab supports setups combining SHFQC devices [138] for qubit drive and readout, HDAWGs [139] for flux control, and a PQSC [140] for synchronization via ZSync. These instruments provide high-fidelity signal generation with internal IQ mixing, wide bandwidth, and low-latency feedback. The integration is handled through the Python-based LabOneQ library [141], enabling control of multi-qubit systems with tunable coupler interactions.

Self-programmed RFSoCs Finally, for RFSoCs, Qibolab supports Xilinx boards such as the RFSoC4x2 [142], ZCU111 [143], and ZCU216 [144], which feature direct RF synthesis up to $\sim$9.8 GHz, reducing the need for external mixers and oscillators. The drivers interact with the open-source Qick firmware and the on-board Qibosoq server, providing a cost-effective and flexible solution for small-scale experiments. While these platforms are limited in scalability and synchronization compared to commercial systems, they offer an accessible entry point for new laboratories.

Feature	RFSoCs	Qblox	QM	Zhinst
Arbitrary pulse sequences	✓	✓	✓	✓
Arbitrary waveforms	✓	✓	✓	✓[1]
Multiplexed readout	✓	✓	✓	✓
Hardware classification	✗	✓	✓	✓
Fast reset	💻	💻	💻	💻
Device simulation	✗	✗	✓	💻
RTS frequency	✓[2]	✓	✓	✓
RTS amplitude	✓	✓	✓	✓
RTS duration	✗	✓	✓	✓
RTS start	✓	✓	✓	✓
RTS relative phase	✓	✓	✓	✓
RTS 2D any combination	✓	✓	✓	✓
Sequence unrolling	💻	💻	💻	💻
Hardware averaging	✓	✓	✓	✓
Singleshot (No Averaging)	✓	✓	✓	✓
Integrated acquisition	✓	✓	✓	✓
Classified acquisition	✓	✓	✓	✓
Raw waveform acquisition	✓	✓	✓	✓

Table 5.1: Capabilities and limitations of the main drivers supported in `Qibolab 0.1.0`. Features marked with "✓" are supported, those marked with "✗" are not supported, and those marked with "💻" are currently under development.

In addition to control and readout electronics, `Qibolab` also integrates auxiliary instruments such as local oscillators, which are essential for frequency conversion and traveling-wave parametric amplifier (TWPA) pumping. Since these devices must be calibrated and dynamically controlled during experiments, version `0.1.0` includes drivers for Erasynth [145] and Rohde&Schwarz sources [146], ensuring seamless operation within the same framework.

Table 5.1 summarizes the features supported by Qibolab 0.1.0, described below:

- **Arbitrary pulse sequences:** execution of user-defined pulse sequences, independent of waveform shape.
- **Arbitrary waveforms:** support for custom waveform shapes; if not available, rectangular, Gaussian, and DRAG pulses are still provided.
- **Multiplexed readout:** simultaneous play and acquisition of multiple pulses on a shared readout line, useful for multi-qubit chips.
- **Hardware classification:** single-shot state discrimination performed *during* sequence execution.
- **Fast reset:** active reset of a qubit to the ground state after measurement, requiring hardware classification and enabling faster repetitions.
- **Device simulation:** preview of pulse execution without using hardware.
- **RTS frequency:** real-time sweeping of pulse frequency for faster qubit characterization.
- **RTS amplitude:** real-time sweeping of pulse amplitude.
- **RTS duration:** real-time sweeping of pulse duration.
- **RTS start:** real-time sweeping of pulse start time.
- **RTS relative phase:** real-time sweeping of pulse phase.
- **RTS 2D:** combined sweeping of two parameters simultaneously.
- **Sequence unrolling:** merging subsequences into longer sequences to reduce compilation and communication overhead.
- **Hardware averaging:** repeated execution with results averaged directly on the device.
- **Singleshot (No Averaging):** retrieval of all non-averaged measurement results.
- **Integrated acquisition:** demodulation and integration of IQ signals over the measurement window.
- **Classified acquisition:** binary state classification after integrated acquisition.
- **Raw waveform acquisition:** access to non-integrated IQ waveform data.

5.4 Examples of Experiments and Applications

In this section, we present some experiments we have performed to test Qibolab's capabilities. The idea behing our experiments is twofold: on one hand, we have focused on benchmarking the performance of different electronics while executing various calibration routines. This is something we manage to achieve thanks to Qibolab's modularity and flexibility, and offers a neutral perspective on the capabilities of the underlying hardware controllers. On the other hand, we have executed some quantum algorithms to demonstrate how the Qibolab framework can be exploited from a higher-level perspective.

5.4.1 Benchmarking Experiments

In this section we present speed benchmarks performed with Qibolab on the different control devices supported by its drivers. Using a common interface makes it easy to compare performance across instruments and to evaluate their efficiency. The results provide useful data for researchers when choosing hardware, while also showing the level of support already available in Qibolab.

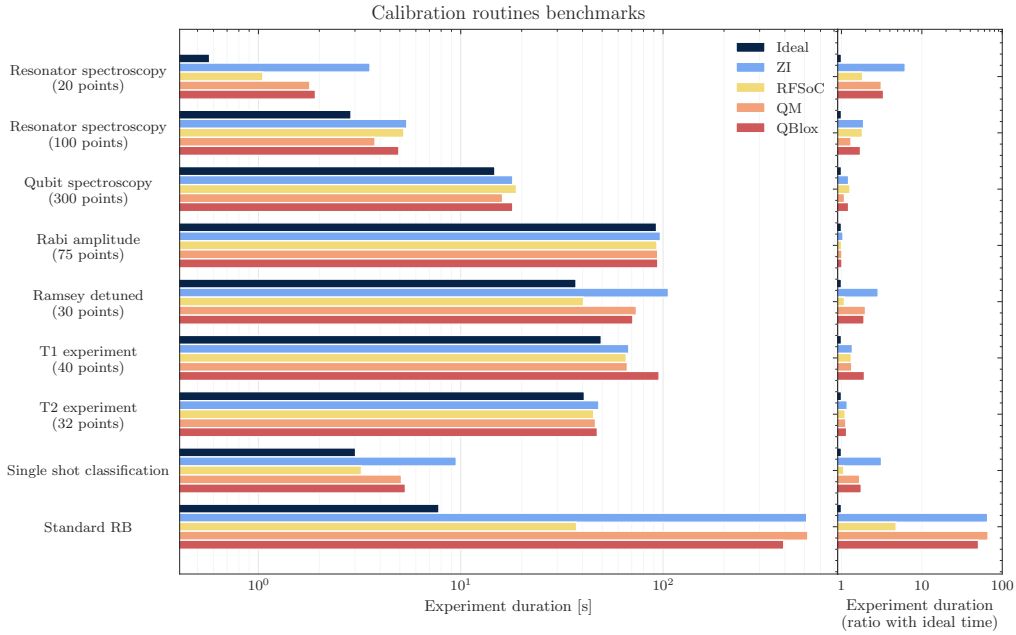


Figure 5.3: Execution times of qubit calibration routines on different control electronics. The left panel shows absolute times (in seconds) for each experiment, with the black bar indicating the ideal minimum time the qubit is actively used. The right panel shows the ratio of actual to ideal time. Real-time sweepers are applied whenever supported, except in the *Ramsey detuned* and *Standard RB* experiments. Image reproduced from [6].

The benchmark routines represent the minimal set of experiments needed to calibrate a single qubit. They also highlight different execution modes: *Single shot classification* uses fixed pulse sequences, while *Spectroscopies* involve parameter sweeps. For a more detailed description of calibration experiments we invite the reader to consult Reference [147]. Figure 5.3 compares execution times for these routines on different devices. The black bar shows the ideal time, defined as

$$\text{ideal} = n_{\text{shots}} \sum_i (T_{\text{sequence},i} + T_{\text{relaxation}}), \tag{5.1}$$

where $T_{\text{sequence},i}$ is the sequence duration at sweep point i, $T_{\text{relaxation}}$ the qubit reset time, and n_{shots} the number of repetitions. This represents the time the qubit is actually used. Real execution times are longer due to compilation and communication overhead. Profiling shows that the overhead from the `Qibolab` backend is negligible, so we approximate

$$\text{real} \approx T_{\text{inst}} + \text{ideal}, \tag{5.2}$$

with T_{inst} the overhead from the control electronics.

Performance strongly depends on how sweeps are executed. If sweeps run in real-time on the control hardware, overhead is small. If they run on the host computer, frequent communication and recompilation add significant delays. This is visible in *Ramsey detuned* and *Randomized Benchmarking (RB)*, where real-time sweepers are not yet available. RB is further slowed because it requires many random sequences, but will improve once sequence unrolling is implemented.

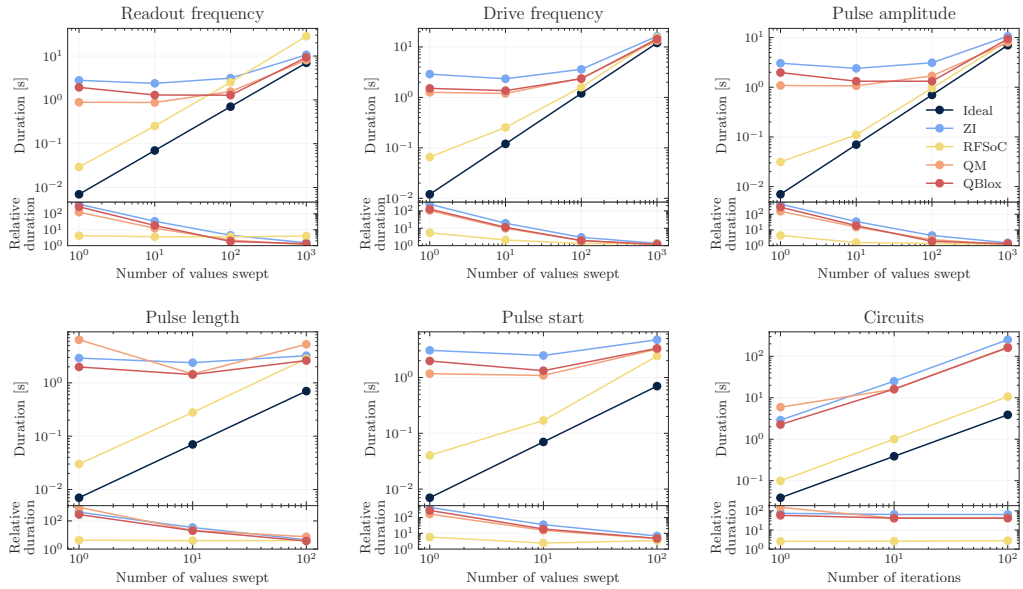


Figure 5.4: Execution time scaling as a function of the number of points in a sweep. The bottom plots display the ratio between the actual execution time on different instruments and the minimum ideal time. In all cases, real-time sweepers are used, except in the final *Circuits* plot, where a standard RB experiment is employed to generate the specified number of random circuits to be executed. Image reproduced from [6].

Communication with the host is another limiting factor, involving both network transfer and compilation on the instrument. RFSoC boards controlled with `Qibosoq` show an advantage here, especially in *Ramsey detuned* and *Single shot classification*, likely due to their simpler single-board setup compared to multi-controller clusters. Other instruments show similar performance across benchmarks.

Figure 5.4 shows how execution time scales with the number of sweep points. RFSoCs are faster for short sweeps due to lower overhead, but the difference decreases beyond 100 points. `Qick` does not yet support real-time sweeping of readout frequency or pulse length, so these sweeps are slower for large point counts. Real-time sweepers are used in all other cases except *Circuits*. Ongoing work on sequence unrolling will further reduce communication overhead and improve runtime.

All benchmark code is available in a public repository [42].

5.4.2 Higher-Level Experiments

After showing how `Qibolab` can be used to benchmark different control electronics, we now demonstrate its capabilities in higher-level quantum experiments. In particular we showcase three different applications: *i)* the execution of a standard randomized benchmarking protocol, *ii)* the implementation of the Clauser-Horne-Shimony-Holt (CHSH) experiment to measure quantum entanglment in the hosted device and *iii)* a quantum machine learning application.

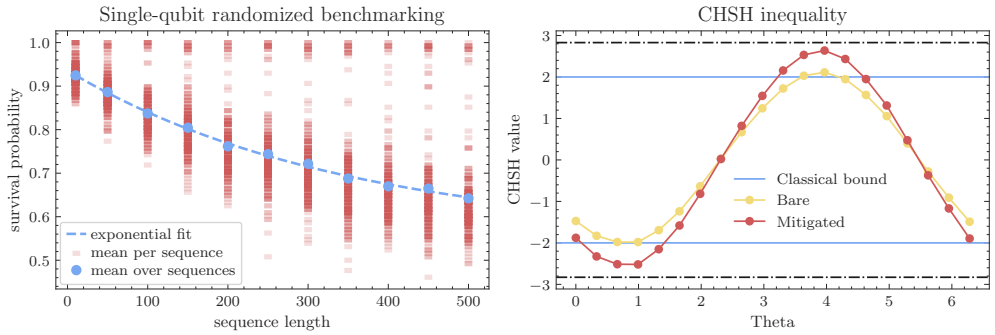


Figure 5.5: Left: results of a single-qubit randomized benchmarking experiment implemented in `Qibo` and executed with `Qibocal` on a 5-qubit IQM device controlled via `Qibolab` 's Zurich Instruments drivers. The survival probability is plotted for random Clifford sequences of varying length, together with the mean over multiple randomizations and an exponential fit yielding average single-qubit gate fidelities above 99.8%. Right: results of a Bell-state experiment performed on two qubits, showing a sweep over the Bell angle on a 5-qubit QuantWare device controlled with `Qibolab` 's Qblox drivers. Readout error mitigation significantly improves the CHSH values, pushing the inequality beyond the classical bound. Figure reproduced from [6].

Randomized Benchmarking Standard randomized benchmarking (RB) with the Clifford group is a widely used method to assess the accuracy of single-qubit gates [148, 149, 150, 151, 152, 153]. The protocol applies random sequences of Clifford gates of varying length, ending with a gate that ideally returns the qubit to its initial state. In the absence of errors, the qubit should be measured in state 0 with probability one, independent of sequence length. Gate fidelities are then extracted from the decay of this average survival probability.

RB provides a holistic test of both hardware and software. In our case, the protocol is defined with the `Circuit` API of `Qibo` using U3, RX, RY, and RZ gates. When executed with the `Qibolab` backend, these are transpiled to native gates, compiled into `PulseSequence` objects, and run on the chosen `Platform`. An example RB experiment on a 5-qubit IQM chip controlled via Zurich Instruments drivers is shown in the left panel of Figure 5.5.

CHSH experiment A quantum software framework should support experiments at different levels of abstraction. To illustrate this, we implement a CHSH inequality test [154] between two qubits using three approaches available in `Qibo` and `Qibolab`: direct pulse-level programming, native gate definitions, and logical gate circuits with automatic transpilation. We also apply readout error mitigation [155], executed on hardware before the experiment, showing the benefit of a framework that integrates all abstraction layers.

The CHSH inequality, originally proposed to test local hidden-variable theories and prove Bell's theorem [156], requires preparing a maximally entangled two-qubit state and measuring both qubits with two possible settings. Each qubit can be measured in two perpendicular bases (*e.g.* X, Z), with a relative angle θ. The resulting correlations are combined as

$$S = E(a, b) - E(a, b') + E(a', b) + E(a', b'), \tag{5.3}$$

which satisfies $|S| \leq 2$ under local hidden-variable models, but can reach $2\sqrt{2}$ in quantum mechanics.

In our case, the CHSH experiment serves as a validation of the chip and control electronics rather than a fundamental test of nonlocality. We performed the experiment on two connected qubits of a 5-qubit QuantWare chip controlled with `Qibolab` 's Qblox drivers. As shown in the right panel of Figure 5.5, the bare CHSH values only slightly exceeded the classical bound, but with readout error mitigation they clearly surpassed 2. This indicates that readout errors, rather than gate control, were the dominant source of noise in the experiment.

Quantum machine learning Finally, we demonstrate a quantum machine learning application. Adopting the same approach presented in Chapter 2, we train a parametric quantum circuit (PQC) to approximate a relevant one-dimensional function in high-energy physics. In particular, we target the up quark parton distribution function $uf(x)$, which represents the probability of finding an up quark inside a proton carrying a fraction x of the proton's momentum. This function is crucial for understanding the internal structure of protons and the dynamics of quantum chromodynamics (QCD). We use as reference labels the predictions produced by the NNPDF4.0 PDF grid [157].

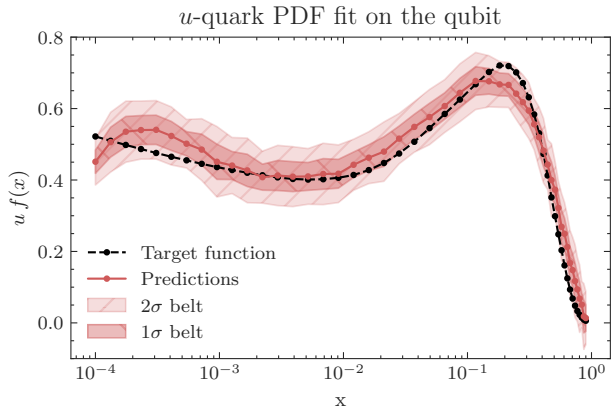


Figure 5.6: Predictions of the up quark parton distribution function obtained training a parametric quantum circuit (PQC) defined with `Qibo` and executed on a self-hosted superconducting qubit controlled by RFSoC through `Qibosoq`. Estimations and uncertainties are obtained as mean and standard deviation of fifty samplings. Image reproduced from [6].

To challenge `Qibolab` and explore the possibilities of our hybrid environment, we propose an hybrid training of the PQC proposed in [158] starting with an initial optimization in exact simulation mode using `Qibo` and proceeding with sixty iterations of training performed on a single superconducting qubit produced by our foundry [159] and controlled by RFSoC through `Qibosoq`.

This exercise is particularly interesting because it fits within a context where classical and quantum nodes can be seamlessly integrated, allowing for a more flexible approach to quantum machine learning.

To do so, we implement an Adam optimizer [160] compatible with quantum hardware, where gradients of the cost function are estimated using parameter-shift rules as described in Section 2.3.3.

The results are presented in Figure 5.6, where we show the average predictions produced by the PQC after the training process. The prediction error is calculated repeating the sampling of the estimations fifty times and computing the standard deviation of the

results. Adam parameters where $\eta = 0.1$, $\beta_1 = 0.85$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$. The accuracy of the prediction has been evaluated computing a mean-squared error value of $\text{MSE} = 0.0021$.

This experiment is clearly a dummy exercise, involving a single qubit circuit and a one-dimensional target, but it is useful to demonstrate how the `Qibo` ecosystem can be leveraged for hybrid quantum-classical machine learning tasks.

5.5 The Importance of Open-Source Quantum Middleware

`Qibolab` adds an important layer of abstraction bridging the gap between quantum hardware and high-level software, offering a flexible tool for developing and executing low-level experiments or higher-level quantum algorithms. Having the possibility of operating at low level (*e.g.* directly manipulating pulses) allows researchers to fine-tune their experiments and optimize performance, ultimately leading to more accurate and reliable quantum computing solutions. The seamless integration with an high-level software stack like `Qibo`, which is in turn integrated with state-of-art libraries like `TensorFlow` and `PyTorch`, paves the way for exploring new hybrid solutions, involving both classical and quantum routines and, potentially, benefitting from the potentialities of both paradigms.

CHAPTER 6

`Qibocal`: Characterization and Calibration of Quantum Devices

In Chapter 4, we presented `Qibo`, a framework for classical simulation of quantum circuits. We showed how it offers a simple interface to quantum computing, allowing users to define and test quantum algorithms on classical hardware. A further step towards a comprehensive quantum computing framework has been presented in Chapter 5, where `Qibolab` has been introduced as a middleware layer for controlling quantum devices. Offering an open-source tool to bridge the gap between theoretical operations and low-level manipulation of real qubits is a crucial step towards realizing practical quantum computing applications. However, the successful deployment of quantum algorithms on real hardware requires a deep understanding of the devices' characteristics and behaviors. In particular, a robust characterization and calibration of the hosted devices is essential for ensuring optimal performances and reliable operation. This is where `Qibocal` comes into play, providing a set of tools and interfaces for the systematic characterization and calibration of quantum devices.

In this chapter, we introduce `Qibocal`, a framework designed to facilitate the characterization and calibration of self-hosted quantum devices.

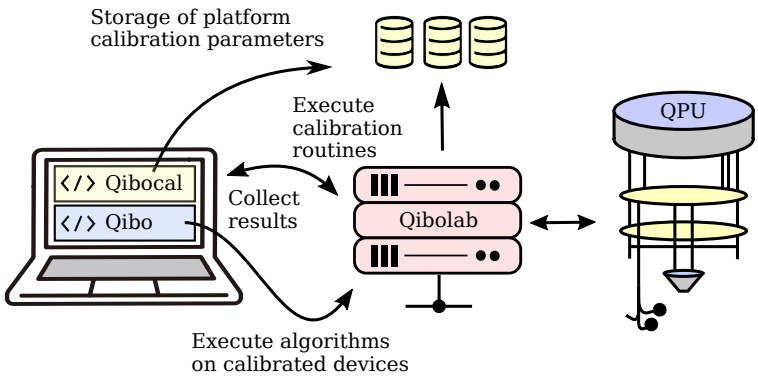


Figure 6.1: Illustration of `Qibocal`'s role within the full-stack quantum computing infrastructure. Image reproduced from [7].

The discussion will follow the structure of our original publication [7].

The `qibocal` package is open-source: https://github.com/qiboteam/qibocal.

6.1 Characterization and Calibration of Quantum Devices

In Chapter 3, we introduced basic notions of superconducting qubits and their role in quantum computing. We discussed the challenges and limitations of current superconducting qubit technology, and presented the physical mechanisms underlying superconducting qubit's operations. For example, we saw how single qubit gates are implemented sending microwave pulses through drive lines coupled with the qubits composing the devices (Section 3.1.2).

Even though these mechanisms are well understood and we know the form of the drive Hamiltonian implementing a specific, theoretical, operation, to achieve an optimal performance we need to account for the imperfections and noise present in real devices. This requires a systematic approach to characterize and calibrate the quantum hardware, ensuring that the actual operations closely match the idealized models. This is the primary goal of `Qibocal`.

`Qibocal` offers a suite of experiments that can be used to define custom calibration programs with a user-friendly Python interface. In particular, we can divide these experiments into three categories: *i) characterization experiments*, which can be used to extract individual parameters of the system, *ii) calibration experiments*, which are fine-tuning experiments optimizing pulses' parameters and *iii) generic experiments*, which cannot be classified into the previous categories. We will refer to *i)* and *ii)* as *protocols*.

6.2 Software Design

In `Qibocal`, a protocol is implemented as a `Routine` class which, in abstraction, corresponds to a generic experiment one wants to perform. A protocol usually involves two operative procedures: *data acquisition* and *data processing*. The first procedure is responsible for collecting raw data from the quantum device, while the second one focuses on analyzing and interpreting the acquired data to extract meaningful information about the system's behavior.

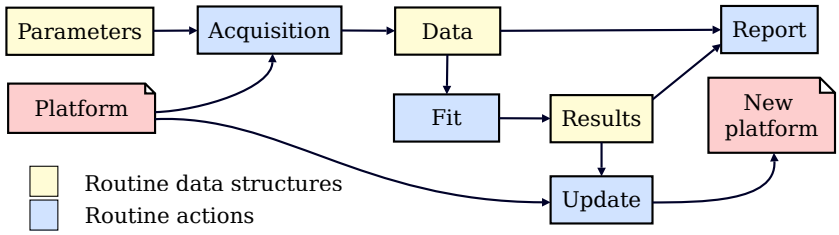


Figure 6.2: Scheme of the dependencies of a `Routine` in `Qibocal`. Image reproduced from [7].

If the user is satisfied by the results of an experiment, the new device parameters can be saved updating the current recorded `Platform` instance as showed in the red box on the right side of Figure 6.2. We remind that the concept of `Platform` has been introduced in Section 5.2.2.

The objects involved in a typical `Routine` execution pipeline are presented in Figure 6.2: the acquisition procedure consumes a `Parameters` instance, and, after performing the defined experiment, it produces a `Data` instance. The processing procedure then takes a `Data` instance and, after applying the defined analysis, it produces a `Results`

instance. Data are usually saved in binary format for efficient storage and retrieval, while additional parameters are stored in JSON format, but one can customize the storage adopting any Python serialization library. The content of `Result` can be used to update the current recorded `Platform` instance.

6.2.1 Execution Modes

`Qibocal` provides two complementary ways to execute protocols: *i)* command line interface and *ii)* calibration programs.

Command line interface (CLI). The simplest way to run experiments is through the CLI, which is summarized in Figure 6.3. All information required for execution (parameters and QPU configuration) is serialized into a YAML file, called a *runcard*. The runcard is then deserialized by an `Executor`, which instantiates the corresponding `Routine`, dispatches the task, and stores the resulting data and post-processing outputs. The CLI allows the user to run acquisition and processing together (`qq run` command) or separately (`qq acquire`, `qq fit` commands). Multiple experiments can be included in a single runcard and executed sequentially, with the `Platform` updated after each protocol. This makes the CLI particularly suited for monitoring tasks and short recalibration programs.

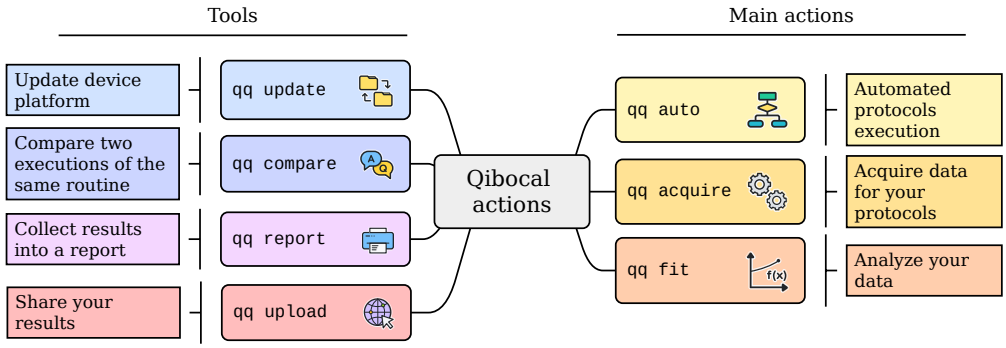


Figure 6.3: `Qibocal`'s command line interface. Image reproduced from [7].

Calibration programs. For more complex workflows, sequential execution is not sufficient. In these cases, protocols are exposed as callable functions within a high-level programming language, currently Python. The `Executor` is realized as a Python object, and each protocol is mapped to one of its methods, which accepts parameters as input and returns processed results. This effectively creates an embedded domain specific language (eDSL) for calibration, where the user can combine protocols with standard programming constructs such as conditionals, loops, or recursion.

This approach allows one to express non-sequential execution schemes, runtime conditionals, and adaptive strategies that cannot be captured by static directed acyclic graphs (DAGs). While DAGs are useful to represent well-defined dependency structures, they cannot handle cyclic dependencies or runtime decisions. Embedding protocols in a programming language overcomes these limitations, enabling scalable and adaptive calibration programs.

6.2.2 Extensibility and Integration

The design of `Qibocal` emphasizes extensibility. New protocols can be added simply by registering them in the `Executor`, making them available at runtime. Although the current implementation is in Python, the same interface could be exposed in other languages through bindings, in line with the broader `Qibo` interoperability effort. This flexibility not only simplifies the specification of calibration workflows, but also facilitates integration with external tools, such as monitoring systems, data providers, or specialized analysis libraries.

Finally, while each protocol typically bundles acquisition, processing, update, and reporting, these stages can be decoupled if needed. For instance, one may process previously acquired data, or postpone report generation to a later stage. This modularity ensures that `Qibocal` can adapt to both simple recalibration tasks and complex, full-chip calibration workflows.

6.2.3 Tools

`Qibocal` also provides a series of tools aimed at facilitating the deployment and interpretation of protocols. At the end of a program launched through the CLI, a web page is produced with a comprehensive summary of all executed protocols. The content of this report is customizable through the `report` function of the `Routine` interface, which can generate plots, tables, or even custom HTML strings. Reports can be easily shared by uploading them to a dedicated server using the `qq upload` command, and two reports can be graphically compared with `qq compare`, producing a combined HTML page that highlights differences between runs.

Finally, although `Qibocal` can generate an updated `Qibolab` platform configuration after an experiment, the actual replacement of the installed configuration is performed as a separate stage using the `qq update` command. This separation ensures that updates are explicit and controlled.

A summary of these commands is shown in the left branch of the diagram in Figure 6.3.

6.3 Calibration Protocols

The code base introduced above can be harnessed to deploy calibration protocols developed using `Qibolab` primitives. `Qibocal` includes an extensive suite of protocols, not only as tools for calibrating superconducting qubit devices, but also as templates for developing new, customized routines. Figure 6.4 shows a gallery of representative protocols.

6.3.1 Single Qubit Calibration

The first step towards full control of a qubit is the calibration of single-qubit gates. `Qibocal` provides protocols for resonator and qubit spectroscopy, Rabi experiments for π-pulse calibration, and standard T_1, T_2^*, and T_2^{Echo} measurements. Additional routines include Ramsey experiments for fine-tuning the drive frequency, and "flipping" sequences to amplify amplitude errors.

Single-shot readout calibration is also supported, with tools to maximize readout fidelity. Beyond standard thresholding, `Qibocal` can train machine learning models to classify states in the IQ plane and benchmark their performance.

More advanced protocols include DRAG pulse calibration to reduce leakage, and optimization of the readout frequency to maximize state discrimination.

A comprehensive description of the available protocols can be found in the Qibocal documentation [161] or in dedicated reviews [147].

6.3.2 Two-Qubit Gates Calibration

Qibocal also supports calibration of two-qubit interactions, particularly for flux-tunable qubits and coupler-based architectures. Protocols include flux spectroscopy to identify interaction points, avoided crossing measurements, and Chevron experiments to extract the parameters of controlled interactions such as iSWAP and CZ gates. Right column of Figure 6.4 shows examples of two-qubit calibration protocols.

Further refinements include correction of dynamical single-qubit phases and mitigation of leakage to higher levels. For coupler-based devices, dedicated protocols identify the operational points where the interaction is switched on or off.

6.3.3 Qubit Benchmarking

Benchmarking protocols are integrated into Qibocal to quantify the performance of calibration. These include randomized benchmarking, interleaved benchmarking, and tomographic routines. Benchmarking can be interleaved with calibration to track improvements and can also serve as cost functions in optimization loops.

6.3.4 Experiments

Beyond calibration, Qibocal can be used to run more complex experiments. Examples include CHSH and Mermin inequality tests to quantify entanglement, which serve both as validation of calibration and as templates for connecting experiments through the Qibocal pipeline. Any experiment implemented in Qibo/Qibolab can be integrated into a calibration stack, enabling automatic pre-calibration before execution.

6.3.5 Automatic Recalibration

The modularity of Qibocal allows calibration protocols to be chained into automatic recalibration workflows. Parameters can be passed between routines sequentially or through more complex logic, with stopping conditions or thresholds coded directly in Python. Verification tools ensure the reliability of fitted parameters, enabling robust automated recalibration.

6.4 Qibocal in Action

We now illustrate Qibocal in practical scenarios.

6.4.1 Coherence at Different Bias Points

As an example of a custom experiment, we measured T_1, T_2^*, and readout fidelity for a qubit at different flux bias points. At each point, the qubit was recalibrated through a sequence of spectroscopy, Rabi, Ramsey, and single-shot classification routines. Results (Figure 6.5) show the expected dependence of coherence and readout fidelity on flux detuning.

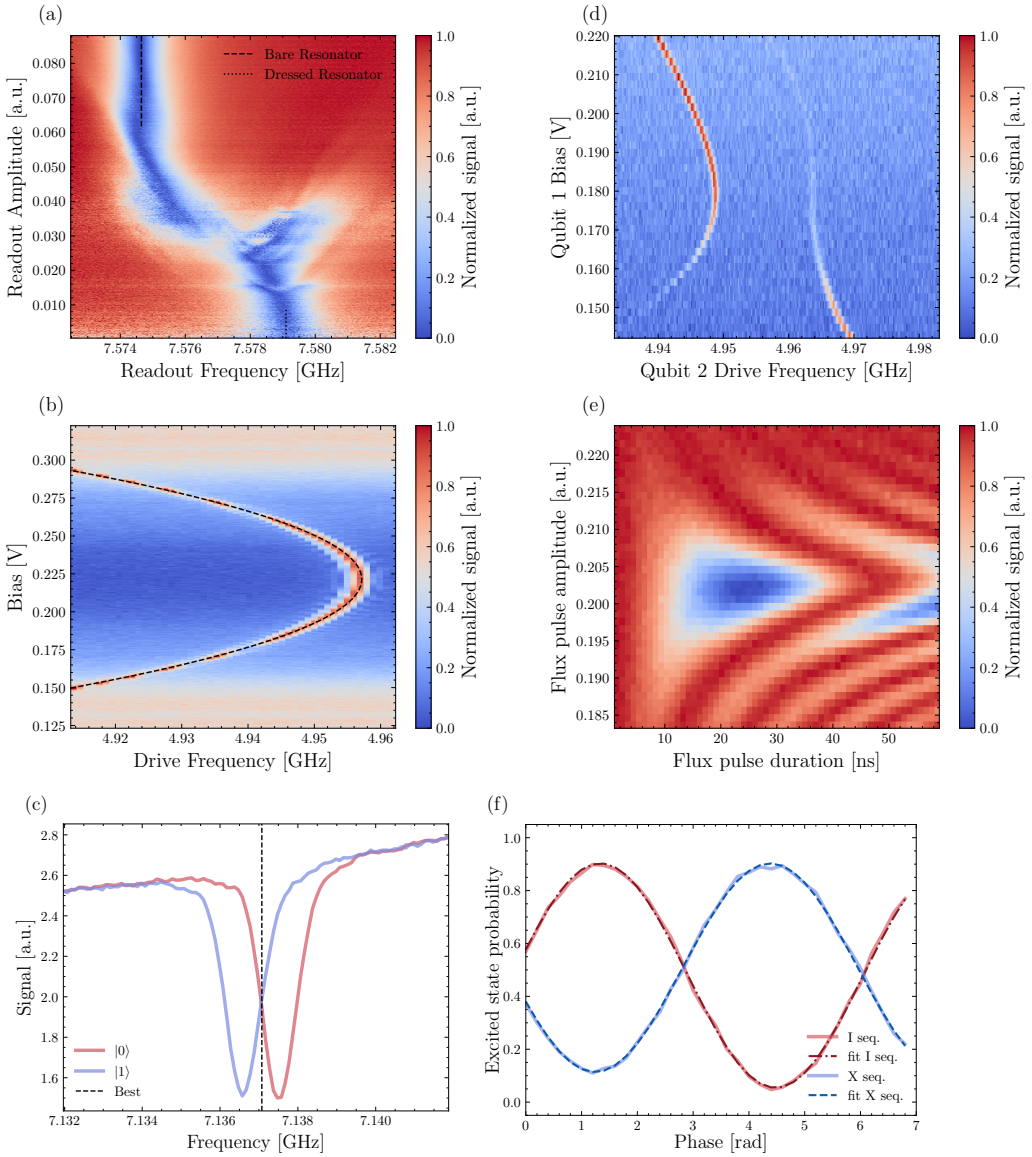


Figure 6.4: Six representative `Qibocal` protocols. *(a)* Measurement of bare and dressed resonator frequencies. *(b)* Qubit frequency as a function of the external bias line, with the dashed line indicating the expected frequency. *(c)* Transmission coefficient as a function of readout frequency for a qubit prepared in state $|0\rangle$ (red) or $|1\rangle$ (blue); the dashed black line marks the frequency that maximizes the separation between the two states. *(d)* Measurement of an avoided crossing between two qubits. *(e)* Chevron pattern observed during calibration of a CZ gate. *(f)* Compensation of dynamical single-qubit phases induced by a flux pulse implementing a CZ gate [8]. Image reproduced from [7].

6.4.2 Pulse Optimization with Randomized Benchmarking

`Qibocal` can be combined with optimization loops. As a demonstration, we optimized the amplitude and DRAG parameter of a $\pi/2$ pulse using a Nelder–Mead algorithm,

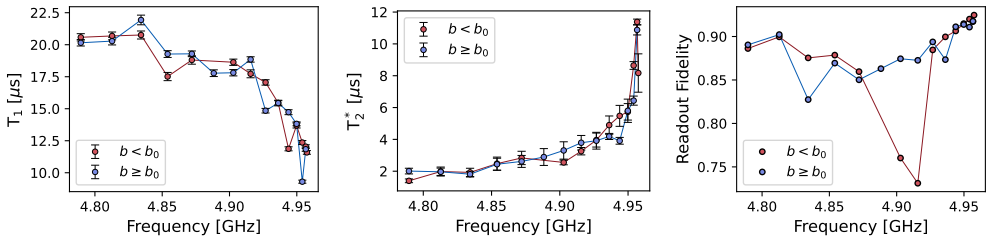


Figure 6.5: T_1, T_2^* and readout fidelity measured as function of qubit's frequency. The blue curve represents the case where detuning is caused by increasing the flux, whereas the red curve corresponds to decreasing flux. At each point, we apply the recalibration procedure outlined in Sect. 6.4.1. Images reproduced from [7].

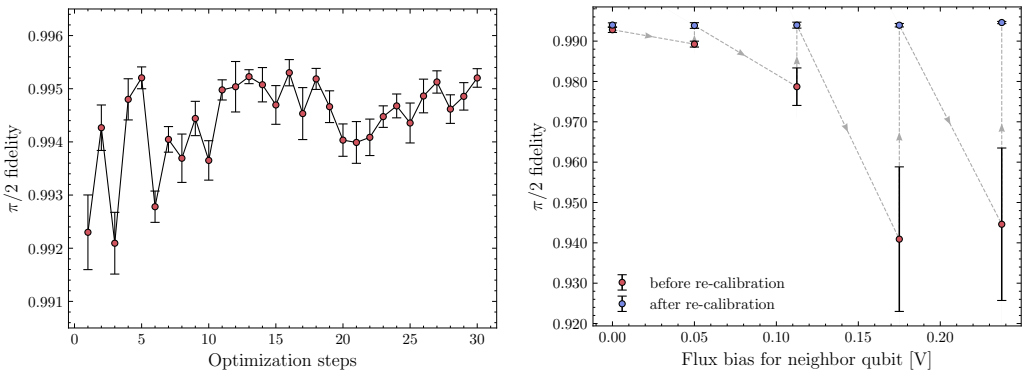


Figure 6.6: (Left) The $\frac{\pi}{2}$-pulse fidelity obtained from randomized benchmarking is used as the cost function in a Nelder–Mead optimization of the pulse amplitude and DRAG parameter. The fidelity after each iteration is shown, demonstrating that the optimization achieves higher fidelity with only a few evaluations. (Right) Recalibration of a qubit's $\frac{\pi}{2}$-pulse following controlled shifts of its flux background. The applied voltage on a neighboring qubit's flux line and the corresponding randomized benchmarking fidelities are shown before (red) and after (blue) recalibration. Gray dashed arrows indicate the evolution of the calibration over time, with flux bias given relative to the qubit's calibrated sweet spot.

with randomized benchmarking fidelity as the cost function. The optimization converged in a few steps, improving gate fidelity (left panel of Figure 6.6).

6.4.3 Re-Calibration after Flux Drifts

Environmental changes, such as flux background drifts, require frequent recalibration. We demonstrated a workflow where Ramsey, Rabi, and single-shot classification routines were combined to re-calibrate a $\pi/2$ pulse after controlled flux shifts. Randomized benchmarking confirmed the recovery of fidelity after recalibration (right panel of Figure 6.6).

6.4.4 Monitoring Qubit Calibration

Finally, `Qibocal` can be integrated into monitoring frameworks such as Grafana [162]. Using Docker containers, we deployed automated routines to measure T_1, T_2^*, and read-

out fidelity at regular intervals, storing results in a PostgreSQL database. Dashboards allow continuous monitoring of qubit performance, with the possibility of triggering automatic recalibration when metrics fall below thresholds.

6.5 Low-Level Computation Can Cooperate with Higher Level

Real-time access to calibration routines enables a new layer of computational flexibility. By embedding calibration steps directly into the execution of experiments, higher-level protocols can be refined with greater precision and stability. This capability not only improves the reliability of complex quantum operations but also paves the way for hybrid computational approaches, where calibration and algorithmic execution are dynamically interleaved. Such integration allows experiments to adapt on the fly, opening possibilities for fine-grained control and more resilient performance in the presence of noise and drift.

In the broader context, these methods extend beyond low-level control and can be leveraged within advanced algorithmic frameworks. For instance, in one of the following chapters we will illustrate how calibration modules can be incorporated into quantum machine learning workflows, introducing the `qiboml` framework as a concrete example of how real-time calibration can support higher-level computational tasks.

Part III

Quantum-Classical Orchestration

CHAPTER 7

`Qiboml`: Hybrid Quantum-Classical Machine Learning

Quantum machine learning (QML) is one of the most interesting application areas for near-term quantum devices [63, 62]. From a solution-oriented standpoint, it is interesting because there are promising QML applications introduced in the last years, such as variational quantum eigensolvers [21] or quantum generative models [163] among others. But, alongside these advancements, QML is a practical playground to explore hybrid solutions and enhance the orchestration techniques we develop to integrate quantum and classical resources. Being this the main focus of this thesis, we will often prioritize the studying of performances and the implementation of software solutions to improve the orchestration of the hybrid quantum-classical workflows.

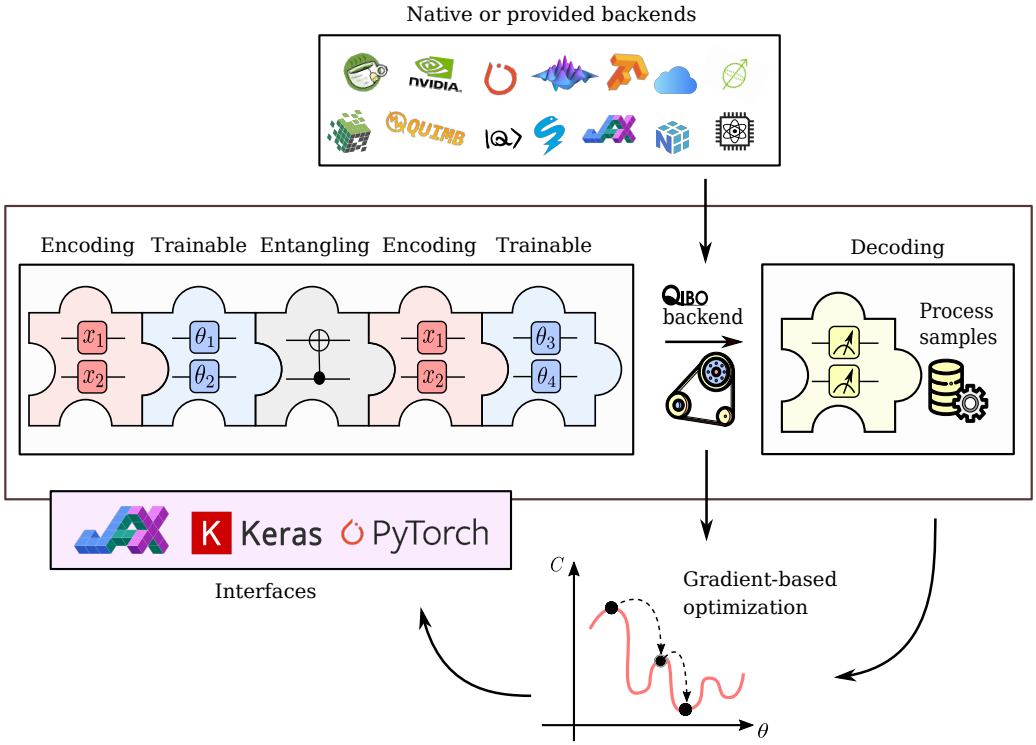


Figure 7.1: Schematic representation of a quantum machine learning pipeline implemented with `Qiboml`. Quantum layers are integrated as native objects within classical ML frameworks, enabling hybrid training workflows.

In particular, in this chapter we introduce `Qiboml`, an open-source software library which integrates `Qibo` with popular classical machine learning frameworks, and we discuss its architecture, features, and potential applications in the field of quantum machine learning. The philosophy behind `Qiboml` is to provide a transparent interface between quantum computing and widely used classical machine learning frameworks such as `TensorFlow`, `PyTorch`, and `Jax`. This allows users to construct hybrid models where quantum layers can be seamlessly combined with classical ones, and trained using the same optimizers, loss functions, and utilities available in the classical frameworks.

`Qiboml` builds on the modular design of `Qibo`, which provides high-level abstractions for quantum circuits, as well as low-level access to control and calibration routines through `Qibolab` and `Qibocal`. This full-stack approach enables the execution of QML models on a wide range of backends: from multi-threaded CPUs, GPUs, and tensor-network simulators, to self-hosted quantum devices and remote cloud-based quantum processors. Importantly, the same high-level code can be executed across all these backends without modification, making it straightforward to switch between simulation and hardware deployment.

An illustration of `Qiboml`'s workflow is shown in Figure 7.1.

> This chapter follows the work presented in [164].
>
> The `qiboml` package is open-source: https://github.com/qiboteam/qiboml.

7.1 Software Design

Putting ourselves in the shoes of a user coming from the classical machine learning field, we organised `Qiboml` such that it provides a set of standard quantum encoders, decoders, and training ansätze, which can be chained to implement `Quantum Layers`. These can be stacked and combined with classical layers seamlessly, enabling hybrid training workflows. In the following sections, the main components of `Qiboml` are described and the possible interfaces are discussed.

In particular, we can identify five main modules in `Qiboml`: *i)* building blocks of a quantum model, such as encoders, decoders, and ansätze; *ii)* interfaces to popular classical machine learning frameworks; *iii)* differentiable backends, implemented in `Qiboml` and made them available for hybrid training; *iv)* custom differentiation engines, which can be selected as alternatives to native automatic differentiation provided by `PyTorch` and `TensorFlow`; and *v)* support components, which include tools for simulating noise, real-time error mitigation and real-time calibration.

7.1.1 Building Blocks of a Quantum Model

`Qiboml`'s quantum layers are constructed to be agnostic to the chosen interface, and can be easily integrated into existing workflows of both `PyTorch` and `TensorFlow`.

Encoders. Encoding blocks map data into quantum circuits. `Qiboml` currently support some of the most commonly used encoding strategies, such as binary encoders and angle encodings. For example, a data-reuploading model [165] with L layers can be easily implemented using L instances of our angle encoding layers. So far, `Qiboml` only

supports pure encoding layers, namely encodings which map data into a quantum circuit which does not depend on any trainable parameters. In the next future, encoders strategies combining data with trainable parameters will be supported.

Qibo circuits. Even though these are not Qiboml's objects per se, any Qibo circuit can be used as a quantum layer, and stacked with encoding layers to compose the full architecture of our model. The way how this can be done is explained in Section 7.1.2. A set of standard ansätze is also provided, such as hardware-efficient ansätze [22] or Hamming-weight preserving ansätze [69].

Decoders. Once data are encoded in the quantum model and the trainable architecture is defined, the last step is to extract relevant information from the quantum state. This is achieved through decoder layers, which map the quantum state back to classical information. We support various decoding strategies; some of them, such as the full state-vector extraction, can be chosen in simulation mode only, while the majority of decoders are compatible with a real quantum hardware setup. Among these we mention `Samples`, `Expectation` and `Probabilities` decoders.

The modularity of Qiboml's design also allows users to define custom encoders and decoders by inheriting from abstract base classes. This makes it possible to tailor QML models to specific applications or hardware constraints. A practical guide to custom encoders and decoders implementation is provided in [166] and [167] documentation pages respectively.

7.1.2 How to Interface with Machine Learning Frameworks?

As previously mentioned, Qiboml is designed to be compatible with popular machine learning frameworks such as `PyTorch` and `TensorFlow`. This is achieved by providing a set of high-level APIs that abstract away the underlying quantum operations, allowing users to leverage familiar constructs from these frameworks. In particular, we provide a `QuantumModel` object, which serves as the main entry point for building and training quantum models within the Qiboml framework.

In the following code snippet, we show how to instantiate the main components of a quantum model and how to construct the overall architecture. This will effectively be equivalent to any `PyTorch` or `TensorFlow` module, and can be stacked into any sequential model of the framework.

```
# Importing Qibo's primitives to build a general circuit
from qibo import Circuit, gates
# Importing Qiboml's building blocks
from qiboml.models.encoding import PhaseEncoding
from qiboml.models.decoding import Expectation
from qiboml.interfaces.pytorch import QuantumModel

nqubits = 4

# Constructing a trainable layer
circ = Circuit(nqubits)
[circ.add(gates.RY(q, theta=0.) for q in range(nqubits))]
[circ.add(gates.RZ(q, theta=0.) for q in range(nqubits))]

# Constructing an encoding layer
enc = PhaseEncoding(nqubits)
```

```
17
18 # Constructing the decoder
19 dec = Expectation(nqubits, backend="any qibo backend")
20
21 # Finally building the model
22 model = QuantumModel(
23   circuit_structure = [enc, circ, enc],
24   decoding=dec,
25 )
```

As shown in the next sections, the `QuantumModel` can be customized not only by choosing different encoders, circuits, and decoders, but also by adding extra instructions such as noise into the simulation, or error mitigation techniques. What is important, so far, is that all these components can be easily integrated into the existing framework, allowing for a high degree of flexibility and customization.

7.1.3 Differentiable Backends

To enable the training of quantum models using gradient-based optimization in simulation mode, `Qiboml` provides differentiable backends that can be used seamlessly with classical machine learning frameworks. These backends are built on top of `Qibo`'s existing simulation capabilities, and are designed to support automatic differentiation. Currently, `Qiboml` si providing three different differentiable backends: `Jax`, `PyTorch`, and `TensorFlow`. Each backend is implemented as a separate module, and can be set following the same `Qibo` interface.

7.1.4 Differentiation of an Hybrid QML Model

Gradient evaluation is a central component of QML training.

In Chapter 2 we discussed differences and similarities between classical and quantum machine learning and, among the others, we highlighted how computing gradients of a quantum model differs from its classical counterpart. In `Qiboml`, we provide a flexible framework to consent the user to choose between standard automatic differentiation techniques, as the ones natively provided by `PyTorch` and `TensorFlow`, or custom differentiation engines, which are more suitable for realistic quantum models.

The reason we provide both options is that, while automatic differentiation is usable in exact simulation mode, it is not compatible with real quantum hardware, where sampling and noise are present.

In practice, we offer a `differentiation` module where various custom differentiation techniques are implemented, and can be selected as alternatives to native automatic differentiation provided by `PyTorch` and `TensorFlow`.

Among the custom differentiation engines we have simulation tools, such as a `Jax`-based engine to enable differentiability for all the backends supported by `Qibo`, including GPU and multi-threaded CPU backends. When sampling and shot-noise are considered, we provide a series of hardware-compatible differentiation techniques. In particular, we offer parameter-shift rules (PSR) [73], and adjoint differentiation techniques [168].

This flexibility ensures that QML models can be trained efficiently across both classical and quantum backends.

From a practical perspective, the choice of the differentiation technique has to be made when instantiating the `QuantumModel` object. In the following code snippet, we provide a simple example of how to set up a quantum model with a specific differentiation technique.

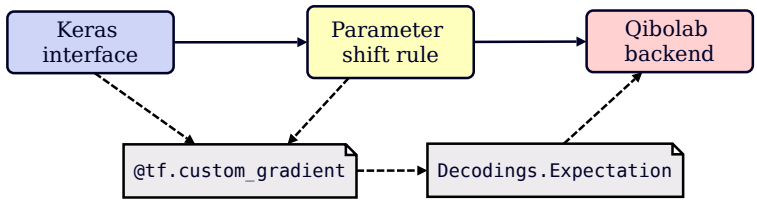


Figure 7.2: Example of custom differentiation in Qiboml. Here, the parameter-shift rule is used to compute gradients on hardware backends, while the high-level interface remains fully compatible with classical ML frameworks such as Keras.

```
# Importing all the necessary modules (see the previous snippet)
# We import here the parameter-shift rule
from qiboml.operations.differentiation import PSR

# Construct the circuit structure
...

# Constructing the decoder
dec = Expectation(nqubits, backend="any qibo backend", nshots=1024)

# Define a model and choosing PSR as differentiation method
model = QuantumModel(
  circuit_structure = chosen_circuit_structure,
  decoding=dec,
  differentiation=PSR(),
)
```

Finally, to give a full overview of the Qiboml interface, we show an example of how to run a VQE (see Section 2.3.1) algorithm using Qiboml's high-level API.

```
# Importing Numpy backend from Qibo
from qibo.backends import NumpyBackend

# Importing tools to build the quantum model
from qiboml.models.ansatze import HardwareEfficient
from qiboml.models.decoding import Expectation
from qiboml.interfaces.pytorch import QuantumModel
# Importing parameter-shift rule differentiation
from qiboml.operations.differentiation import PSR

# Building the VQE circuit
vqe_circ = HardwareEfficient(nqubits=5, nlayers=3)

# Constructing the decoding
dec = Expectation(
    nqubits=5,
    backend=NumpyBackend(),
    nshots=1024,
)

# Constructing the model
model = QuantumModel(circuit_structure=circ, decoder=dec, differentiation=PSR())

# Defining the optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=0.1)
```

```
26
27 # Training
28 for epoch in range(epochs):
29     optimizer.zero_grad()
30     cost = model()
31     cost.backward()
32     optimizer.step()
```

As shown in the code, the `Qiboml` API offers an intuitive interface to write, in a few lines of code, complex quantum machine learning models.

7.2 Support Components

`Qiboml` includes a set of support components to enhance the usability or the definition of quantum machine learning in realistic scenarios. In particular, we mention here *i)* the `CircuitTracer`, an object which facilitates the tracking of the gradients in hybrid workflows; *ii)* noise simulation tools, inherited from `Qibo`, *iii)* real-time error mitigation techniques and *iv)* real-time calibration routines.

Gradient tracing. The differentiation procedure, especially when involving hardware-compatible techniques and complex models which combine data with parameters, is facilitated by a crucial `Qiboml` object: the `CircuitTracer`. A `CircuitTracer` instance is responsible for tracking the flow of data and gradients through the quantum circuit, enabling efficient backpropagation and gradient computation. In practice, it takes all the building blocks of a quantum model and, while constructing the whole quantum circuit, pre-computes the functional form of the Jacobian matrices describing all the forward process.

Noise simulation. To prepare for the challenges posed by noisy quantum devices, `Qiboml` includes a suite of tools for simulating noise, directly inherited from `Qibo`. Specifically, our density matrix simulation mode supports several noise channels as the one presented in Section 3.2.1.

The choice of executing algorithms in density matrix mode inevitably increases the computational cost of the simulation, but consent a flexible modeling of noise effects. This flexibility is crucial for preparing robust quantum machine learning models that can perform well in the presence of hardware imperfections.

To give a practical example of how `Qibo`'s noise simulator can be integrated within a QML pipeline, we provide a code snippet below.

```
1  from qibo.noise import NoiseModel, PauliError
2  from qiboml.models.decoding import Expectation
3
4  # Building the noise model
5  noise_model = NoiseModel()
6  noise_model.add(
7        PauliError([("X", 0.01), ("Y", 0.01), ("Z", 0.01),]),
8        qubits=0,
9    )
10
11 # Informing the decoder
12 # we want noisy simulation
13 dec = Expectation(
14     nqubits=1,
```

```
    density_matrix=True,
    nshots=1024,
    noise_model=noise_model,
)
```

Noise mitigation. To address the challenges posed by noise in quantum circuits, Qiboml incorporates advanced error mitigation techniques. These techniques are designed to enhance the robustness of quantum machine learning models by reducing the impact of noise during training and inference. In particular, we offer a real-time error mitigation procedure, which implements the algorithm that will be discussed in details in Chapter 8.

To give a small introduction, this real-time error mitigation approach involves the learning of the noise map, which is cached by a Mitigator object. The Mitigator is instantiated by the user at the beginning of a training process, and it can be initialized by specifying the mitigation configuration when creating the decoder. An example is shown in the following snippet.

```
from qiboml.models.decoding import Expectation

# Defining the noise model as before
# Setting the QEM configuration
mitigation_config = {
    "threshold": 0.1,
    "min_iterations": 500,
    "method": "CDR",
    "method_kwargs": {"n_training_samples": 50, "nshots": 1024},
}

# Informing the decoder about the noise model
# and real time error mitigation
dec = Expectation(
    nqubits=1,
    density_matrix=True,
    nshots=1024,
    noise_model=noise_model,
    mitigation_config=mitigation_config,
)
```

In the code, we show how to customize the real-time error mitigation configuration. The effectiveness of this approach and the parameters appearing in the mitigation_config dictionary will be discussed in details in Chapter 8, where some experimental results will also be presented.

Real-time calibration. Given the modularity of Qibo, Qiboml can be easily integrated with Qibocal to perform real-time calibration of quantum devices within QML workflows. This integration allows for the dynamic adjustment of calibration parameters based on the performance of the quantum model during training or inference. This feature is implemented adopting a similar approach to the error mitigation procedure, where a Calibrator object is instantiated at the beginning of the training process, and it can be configured by specifying the calibration routines to be executed following the same interface presented in Chapter 6.

At this stage, we are still exploring the best strategies to integrate real-time calibration within QML pipelines. Early stopping criteria based on the real-time check of the

lab conditions, or the execution of light re-calibration protocols during the training are among the most promising approaches.

7.3 It Is Easy to Run on Hardware!

A key advantage of `Qiboml` compared to other QML frameworks, such as `PennyLane` [169], is its native integration with self-hosted quantum devices. In fact, being conceived as a native library of the `Qibo` ecosystem, `Qiboml` is designed to work seamlessly with any of the backends supported by `Qibo`, including `Qibolab`.

This is a big difference with respect to cloud-based access to third-party hardware, providing direct deployment on locally hosted devices. This not only simplifies access but also enables deeper customization and control over the hardware, making it a valuable tool for research and experimentation.

7.4 Benchmarking `Qiboml` with `PennyLane`

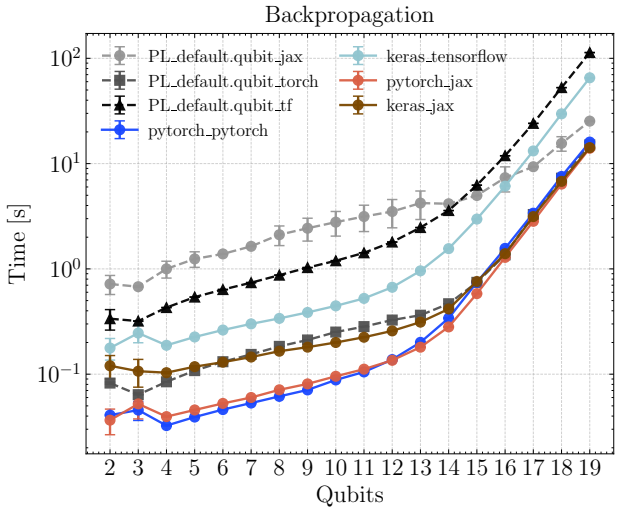


Figure 7.3: Benchmarking of `Qiboml` and `PennyLane` using native automatic differentiation on a single-thread CPU backend. The plot shows the total execution time for training a VQE model with increasing number of qubits. In the legend, `PennyLane`'s backends are labelled with an initial `PL-` prefix, followed by the name of the chosen backend after the lower dash. Our results are labelled as `interface_backend`.

In this section, we will compare the performance of `Qiboml` with `PennyLane`, which is the most widely used framework for quantum machine learning. We perform a series of simple experiments to valuate their performances on various backends. To do so, we choose a simple target Hamiltonian

$$H = -\sum_{i=1}^{n} Z_i, \tag{7.1}$$

and we train a VQE model to find the ground state of H. The ansatz we use is a Hardware Efficient ansatz provided by `Qiboml`. For each system size (n) we execute ten

epochs of training. We repeat each training ten times, collecting performance metrics such as the total execution time. Using the ten collected values, we compute mean and standard deviation of the chosen metric to provide an estimation of the performance with its uncertainty.

In this thesis, we present results for three different differentiation engines: *i)* native automatic differentiation provided by `PyTorch` and `TensorFlow`, *ii)* custom adjoint differentiation [168], and *iii)* custom parameter-shift rule [73].

We first compare `Qiboml` and `PennyLane` using native automatic differentiation on a single-thread CPU backend and up to twenty qubits. The obtained results are shown in Figure 7.3.

After that, we repeat the same experiment using custom adjoint differentiation. `PennyLane` provides a fast adjoint differentiation engine which is implemented in C++, and this is the chosen configuration to benchmark with `Qiboml`. The results are shown in Figure 7.4.

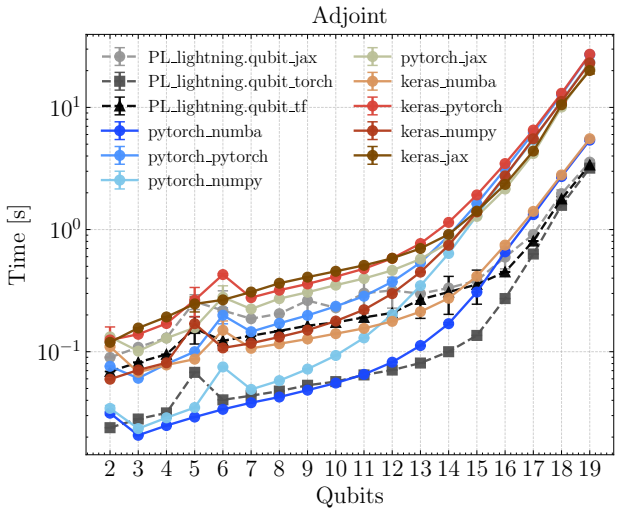


Figure 7.4: Benchmarking of `Qiboml` and `PennyLane` using custom adjoint differentiation on a single-thread CPU backend. The plot shows the total execution time for training a VQE model with increasing number of qubits. In the legend, `PennyLane`'s backends are labelled with an initial `PL-` prefix, followed by the name of the chosen backend after the lower dash. Our results are labelled as `interface_backend`.

Finally, we perform the same experiment using custom parameter-shift rule differentiation, always on a single-thread CPU backend. The results are shown in Figure 7.5.

From the conducted experiments we observe that, made exception for the `PennyLane` lightening qubit backends, which are based on C++ implementations, we provide similar performances across all tested setups. We are currently implementing optimizations to further improve the performance of `Qiboml`.

More benchmarks, including hardware accelerators such as GPUs, and multi-threaded CPU backends, are being tested and will be presented in future works.

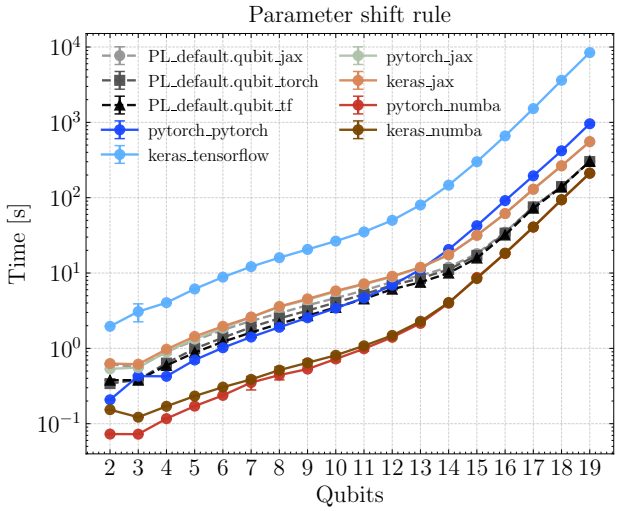


Figure 7.5: Benchmarking of `Qiboml` and `PennyLane` using custom parameter-shift rule differentiation on a single-thread CPU backend. The plot shows the total execution time for training a VQE model with increasing number of qubits. In the legend, `PennyLane`'s backends are labelled with an initial `PL-` prefix, followed by the name of the chosen backend after the lower dash. Our results are labelled as `interface_backend`.

7.5 Orchestration within Quantum Machine Learning Pipelines

One of the most distinctive aspects of QML is its inherently hybrid nature. Training a variational quantum model requires a continuous interplay between quantum and classical resources: quantum processors are used to prepare states and measure observables, while classical optimizers update the parameters and process the results. This orchestration becomes even more critical when considering error mitigation strategies, which often require additional classical post-processing.

The `Qiboml` framework is designed to facilitate this orchestration. By exposing quantum models as native objects within classical ML frameworks, it allows quantum and classical components to be combined into a single computational graph. This means that hybrid models can be trained end-to-end using familiar tools, while the underlying execution is distributed across heterogeneous resources.

A particularly illustrative example is the integration of real-time quantum error mitigation within QML pipelines (see Chapter 8 for details). In this approach, noisy expectation values obtained from the quantum device are corrected on the fly using a mitigation map learned from auxiliary circuits. The map is periodically checked and updated to account for noise drift, ensuring that both predictions and gradients remain reliable throughout training. This procedure requires frequent communication between the quantum processor and the classical node, making it an ideal test case for studying hybrid orchestration.

Looking forward, even tighter integration between classical and quantum resources can be envisioned. For instance, part of the mitigation logic could be moved directly onto FPGA boards that already interface with the quantum hardware. In such a setup, raw measurement samples could be collected, expectation values computed, and mitigation applied entirely on-board, without the need for constant communication with an

external classical node. The two components would only need to synchronize when the noise map itself requires recomputation, which occurs much less frequently. This would significantly reduce latency and improve the efficiency of hybrid QML workflows.

In this sense, QML pipelines implemented with Qiboml serve not only as practical tools for training models on noisy devices, but also as a playground for exploring new strategies in hybrid quantum–classical orchestration. By pushing more logic closer to the hardware, and by leveraging the modularity of the software stack, we can progressively refine the balance between quantum and classical resources, paving the way for more scalable and efficient QML applications.

CHAPTER 8

Real-Time Quantum Error Mitigation

In this chapter, we introduce the first algorithmic experiment of this thesis which aims to explore how our full-stack quantum computing environment can be used to implement real-time solutions, aiming to accelerate the hybrid computing workflow when executing tasks on near-term devices.

In particular, we tackle here the problem of quantum error mitigation within a quantum machine learning workflow and we propose a solution which aims to improve the overall robustness and reliability of the quantum computations, still keeping the computational overhead low.

In the following sections, we will first motivate why quantum machine learning is a good candidate for real-time quantum error mitigation in Section 8.1. Then, we will describe the main components of the algorithm, including the importance Clifford sampling procedure in Section 8.2 and the mitigation map construction in Section 8.2.2. Finally, we will present the *real-time quantum error mitigation* (RTQEM) algorithm in Section 8.3 and we will show some experimental results in Section 8.4.

The RTQEM algorithms has been introduced in [9]. We will follow the main ideas of our original paper, but we will also provide additional details motivated by the recent implementation of the algorithm in `Qiboml`, as described in Chapter 7.

The original `rtqem` package is open-source and available at the repository: https://github.com/qiboteam/rtqem.

8.1 Why Is Quantum Machine Learning a Good Candidate?

Quantum error mitigation routines usually imply a computational overhead and the need for additional resources, making them a challenging task to implement in practice. If we take as an example learning-based error mitigation techniques, as described in Section 3.3.2, we need to compute a certain number of expectation values on a quantum device and, in parallel, use a classical simulator to calculate the corresponding exact values on a classical machine. In addition to this computational effort, we also need to use a classical node to fit the obtained data and extract a mitigation map (see Equation 3.31), which can be finally used to mitigate the results returned by the quantum computer.

This whole process can be very expensive in terms of time and resources, especially when dealing with large-scale quantum machine learning (QML) tasks. Furthermore, communication between classical and quantum computers introduces additional latency in the process. In current superconducting platforms, the timescale of quantum operations is in the range of tens to hundreds of nanoseconds for single- and two-qubit gates, and up to microseconds for measurement. By contrast, the latency introduced by the classical control stack, including signal propagation through cryogenic lines, analog-to-digital conversion, and FPGA-based processing, is already of the same order of magnitude, typically hundreds of nanoseconds to a few microseconds. When the loop extends beyond the local control electronics and involves a classical computing node (*e.g.*, for fitting or optimization), the latency increases by several orders of magnitude, ranging from hundreds of microseconds to milliseconds or even seconds, depending on the complexity of the classical task. This makes real-time feedback extremely challenging, as the classical response is far too slow compared to the coherence times of the qubits.

However, this limitation is less oppressive in the context of QML. Many QML workflows are naturally structured as *batch processes*, where a large number of similar circuits are executed repeatedly to estimate expectation values or gradients. In fact, a, hardware-compatible QML pipeline relies on numerous expectation values calculations, especially because of the need of parameter shift rules (see Section 2.3.3).

In such cases, the quantum device can be used to generate large datasets of noisy measurement outcomes, while the classical node processes these results asynchronously.

Since the noise profile of a quantum device usually doesn't change dramatically, in time scales in the order of the dozens of minutes[1], it is possible to learn a reliable noise map once and reuse it across many iterations of the algorithm. This amortizes the cost of the classical fitting step and reduces the impact of classical-quantum communication latency.

Moreover, QML tasks are often highly parallelizable: multiple circuits can be executed in parallel on the quantum hardware (when supported by the architecture) or batched across different runs, while the classical side can simultaneously process previously collected data. This pipelined workflow ensures that the quantum processor is not idling while waiting for classical feedback, and the classical node can operate on a timescale that is decoupled from the coherence time of the qubits. In other words, while low-latency feedback is essential for quantum error correction, it is not strictly required for QML-based error mitigation, where the learning and fitting steps can tolerate higher latencies as long as the noise model remains stable.

This is why we propose a real-time quantum error mitigation procedure in the context of QML. A schematic representation of the algorithm is shown in Figure 8.1, while a detailed description of the steps involved is provided in the following sections.

8.2 Importance Clifford Sampling

The core idea behind our proposed algorithm is that we do not need to pay the cost of error mitigation every time we evaluate an expectation value on the quantum device. Instead, we learn a noise model from a representative set of measurements and use it to correct subsequent measurements in real time. This remains valid until the learnt noise model becomes outdated, at which point we must recompute it using fresh data.

To enable this strategy, we require circuits that are both classically simulable and structurally similar to the target quantum circuit. These type of circuits serve two pur-

[1] IBM's devices are re-calibrated usually once per day [170].

poses in our framework: *i)* providing a *reference value* that can be used at each iteration as a sanity check of the current mitigation map, and *ii)* supplying a *training set* for retraining the map whenever the reference check indicates that the model is no longer reliable.

For both tasks, we adopt the *importance Clifford sampling* (ICS) procedure introduced in [83]. ICS offers a scalable way to generate circuits that can be efficiently simulated on a classical computer while still capturing the noise characteristics of the original quantum circuit. These ICS-generated circuits thus provide the dual functionality of reference benchmarks and training data, ensuring that our error mitigation remains both efficient and adaptive.

This approach falls into the category of learning-based error mitigation techniques, as described in Section 3.3.2, which are among the best candidates to address trainability problems raised by noise [86]. In fact, our goal here is not to solve all the trainability issues of QML, but to provide a practical solution to mitigate the noise affecting the quantum computations, which is one of the main obstacles to the successful deployment of QML algorithms on real hardware [171].

8.2.1 Motivation for Clifford Sampling

Clifford circuits are particularly attractive because they can be simulated efficiently on a classical computer [36], allowing their error-free expectation values to be computed exactly (see Section 4.3.2 for details). If we construct Clifford circuits that share the same *circuit frame* as the noisy target circuit, then the discrepancy between their noisy and ideal outcomes provides valuable information about the underlying noise processes.

Recalling the notation introduced in Section 3.3.2, the ideal expectation value of an observable O over the state prepared by a given circuit C is

$$\langle O \rangle_{\text{exact}} = \text{Tr}[O\, C(\rho_0)]\,, \tag{8.1}$$

where $\rho_0 = |0\rangle\langle 0|^{\otimes n}$ is the initial state. In the presence of noise, the measured expectation value becomes

$$\langle O \rangle_{\text{noisy}} = \text{Tr}\big[O\, \mathcal{E}\big(C(\rho_0)\big)\big]\,, \tag{8.2}$$

where $\mathcal{E}$ is the noisy channel. The difference between these values, $\langle O \rangle_{\text{noisy}} - \langle O \rangle_{\text{exact}}$, defines the bias introduced by noise.

The goal of ICS is to select Clifford circuits such that their ideal values $\langle O \rangle_{\text{exact}}$ are non-zero and efficiently computable, while their noisy values $\langle O \rangle_{\text{noisy}}$ can be measured on the device. The pairs $(\langle O \rangle_{\text{noisy}}, \langle O \rangle_{\text{exact}})$ are then used to fit the parameters of the mitigation map as described in Section 3.3.2.

Error-Sensitive Clifford Circuits

Not all Clifford circuits are equally informative. If the ideal expectation value $\langle O \rangle_{\text{exact}}$ is zero, the circuit provides no information about the noise. ICS therefore focuses on generating *error-sensitive* Clifford circuits, *i.e.* circuits for which $\langle O \rangle_{\text{exact}} \neq 0$.

For Pauli observables, this condition can be checked explicitly. Given a Clifford circuit U, the effective observable is

$$O_U = U^\dagger O U = \pm P_1 \otimes P_2 \otimes \cdots \otimes P_n, \tag{8.3}$$

with $P_i \in \{I, X, Y, Z\}$. The corresponding ideal expectation value is

$$\langle O \rangle_{\text{exact}} = \pm \prod_{i=1}^{n} \langle 0|P_i|0\rangle. \tag{8.4}$$

If any $P_i \in \{X, Y\}$, then $\langle O \rangle_{\text{exact}} = 0$. Only if all $P_i \in \{I, Z\}$ do we obtain $\langle O \rangle_{\text{exact}} = \pm 1$.

In our implementation, this filtering is performed explicitly: we sample a candidate Clifford circuit from the frame and compute O_U. If X or Y factors appear, we insert *adjustment gates* so that the effective observable reduces to I or Z only. The resulting circuit is guaranteed to be error-sensitive, with non-vanishing expectation value, and is included in the training set.

In particular, the set of error-sensitive Clifford circuits is sampled *uniformly*. This ensures that each generated circuit provides valid training data for learning the mitigation map. The authors of Reference [83] propose a refinement where circuits are drawn according to an importance sampling distribution that favours those with larger bias (and motivates the name of the method itself). Incorporating such a strategy is a promising direction for improving our method, but it is not yet implemented here.

Scaling Properties

A key advantage of ICS is that its resource cost scales only linearly with the number of qubits and gates. This is because Clifford circuits can be simulated efficiently in polynomial time using stabilizer methods [36]. Moreover, the number of error-sensitive circuits needed does not grow significantly with system size [83]. As a result, the residual bias after error mitigation scales as $\mathcal{O}(\sqrt{N})$ where N is the number of gates. This sublinear scaling highlights the effectiveness of ICS for large circuits, compared to the linear $\mathcal{O}(N)$ growth observed without mitigation.

8.2.2 Mitigation Map Construction

The previous sections described how to generate a set of error-sensitive Clifford circuits that can be used in our learning-based error mitigation routine. We now construct a mitigation map ℓ that is representative of the noise affecting a target circuit C_0.

To compute this map, we follow the procedure of Section 3.3.2, where a set $\{C_i\}_{i=1}^{N}$ of training circuits provides pairs of exact and noisy expectation values $\langle O_i \rangle_{\text{exact}}$ and $\langle O_i \rangle_{\text{noisy}}$.

These data are used to train a noise model ℓ mapping noisy values to their exact counterparts. The structure of ℓ is inspired by a global depolarizing channel with depolarizing parameter λ:

$$\langle O \rangle_{\text{noisy}} = (1 - \lambda) \langle O \rangle_{\text{exact}} + \frac{\lambda}{d} \operatorname{Tr}[O], \tag{8.5}$$

where $d = 2^n$ is the Hilbert space dimension and $0 < \lambda < 4^n/(4^n - 1)$.

Restricting to Pauli observables and allowing λ to vary for each circuit, we obtain the phenomenological model

$$\langle O \rangle_{\text{noisy}} = (1 - \lambda_C^i) \langle O_i \rangle_{\text{exact}}, \tag{8.6}$$

where λ_C^i is the depolarizing parameter associated with circuit C_i.

By computing the mean λ_0 and standard deviation σ over the set $\{\lambda_C^i\}$, we define an effective parameter for the target circuit:

$$\lambda_{\text{eff}} = \lambda_0 - \frac{\sigma^2}{1 - \lambda_0}. \tag{8.7}$$

This yields the following mitigation map:

$$\ell(\lambda_{\text{eff}}) = \frac{(1-\lambda_0)}{(1-\lambda_0)^2+\sigma^2}\,\langle O\rangle_{\text{noisy}}. \tag{8.8}$$

This effective noise map empirically improves as the depth of the circuit increases, consistent with the discussion in the previous section.

8.3 The RTQEM Algorithm

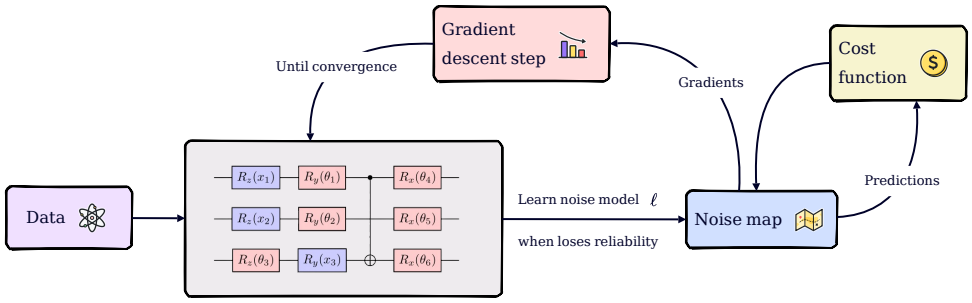


Figure 8.1: Schematic representation of the real-time quantum error mitigation algorithm in the context of quantum machine learning (QML). Image taken from [9].

The procedure described in Section 8.2 allows us, given a reference quantum circuit C_0, to sample error-sensitive circuits C_i that can be used to compute a mitigation map ℓ for the noise affecting C_0. From now on, we will refer to the procedure described in Section 8.2.2 as *map construction*.

Once the noise map is learned, it can be used to mitigate noisy expectation values by applying the learned map to the measured results. This procedure remains valid until the noise map is no longer reliable. In practice, the noise profile of a quantum device changes over time, making it necessary to periodically update the map to ensure accuracy.

To achieve this, we define a strategy referred to as *map check*. At the beginning of the algorithm execution, we sample a reference circuit C_{ref} using the procedure described in Section 8.2. We also cache the exact expectation values $\langle O_{\text{ref}}\rangle_{\text{exact}}$ for the observables of interest. Then, whenever required, we perform a map check by comparing the current mitigated expectation values $\ell(\langle O_{\text{ref}}\rangle_{\text{noisy}})$ with the cached exact values.

In practice, we define a threshold δ such that if the difference

$$\delta = |\ell(\langle O_{\text{ref}}\rangle_{\text{noisy}}) - \langle O_{\text{ref}}\rangle_{\text{exact}}| \tag{8.9}$$

exceeds this threshold, the noise map is considered outdated and a re-evaluation is triggered.

RTQEM workflow. As summarized in Figure 8.1, the RTQEM algorithm can be outlined in the following steps:

1. Sample a reference circuit using ICS strategy.

2. Sample a set of training circuits sampled with the chosen strategy.
3. Perform the map construction to learn a noise map ℓ.
4. Execute the target circuits and mitigate the results using the learned map.
5. Periodically perform a map check to assess the reliability of the noise map.
6. If the map check fails, re-sample training circuits and re-evaluate the noise map.

We stress that the method is agnostic to the specific learning-based error mitigation technique used to compute the mitigation map. The ICS procedure can be used to sample the training circuits, as well as any other strategy. For example, one can decide to use partially Clifford circuits, as it is proposed in the Clifford data regression (CDR) method [172]. To prove the versatility of the RTQEM approach, we will present experimental results using both strategies in Section 8.4.

Within the `Qibo` ecosystem, this procedure is implemented in `Qiboml`'s `Mitigator` object, which provides a user-friendly interface for defining and applying noise mitigation strategies when running a QML algorithm.

It is important to note that every evaluation of the mitigation map ℓ introduces a computational overhead that can affect the overall performance of the quantum algorithm. Therefore, it is crucial to balance the frequency of map checks with the associated computational costs to ensure efficient execution of the QML pipeline. We analyze this trade-off in the following sections.

8.4 Experimental Results

To demonstrate the effectiveness of our proposed real-time quantum error mitigation (RTQEM) strategy, we present a series of experiments conducted on both classical simulators and real quantum hardware.

8.4.1 Classical Simulation

We first evaluate the performance of our method using classical simulations. Two tasks are considered: *i)* one-dimensional regression and *ii)* ground state energy estimation. For both tasks, we compare the results obtained when training the models under four different configurations:

- **Exact simulation:** training on a noiseless simulator without sampling from the final state. Expectation values are exact, providing an upper bound for performance.
- **Noiseless simulation with shots:** training on a noiseless simulator with a finite number of shots, introducing statistical noise.
- **Noisy simulation:** training on a noisy simulator using the local Pauli noise channel introduced in Section 3.2.1.
- **RTQEM:** training on a noisy simulator with the real-time quantum error mitigation procedure applied.

In this first series of tests, we decided to use the Clifford data regression method (CDR) [172], which is slightly different from the phenomenological model described in Section 8.2.2. We make this choice to prove the versatility of the RTQEM approach, which can be implemented with different learning-based error mitigation techniques. After the first simulation tests, we move to a second series of experiments where we use the phenomenological model described in Section 8.2.2 and the ICS procedure to generate the training circuits.

One-Dimensional Regression

The first target function is

$$f(x) = \sin(x)^2 - 0.3\cos(x), \tag{8.10}$$

which we approximate using a single-qubit parametric circuit inspired by the data-reuploading ansatz [165]. The circuit structure is illustrated in Figure 8.2.

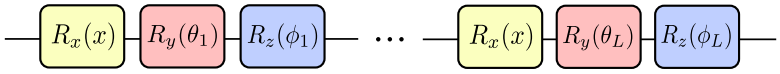


Figure 8.2: Schematic representation of the data-reuploading circuit used for one-dimensional regression. The subscript $1 \leq i \leq L$ indicates the i-th layer of the model.

The training configuration is summarized in Table 8.1. Each training is repeated ten times with different parameter initializations, allowing us to estimate uncertainties. Final results are reported as the median across runs, with error bars given by the median absolute deviation.

Epochs	Runs	Optimizer	Local Pauli Error prob.	# Training data	Shots	Ansatz layers
50	10	Adam($\eta = 0.1$)	0.01	20	500	3

Table 8.1: Training configuration for the one-dimensional regression task.

The results are shown in the left panel of Figure 8.3. The RTQEM approach significantly improves the fit compared to the noisy simulation without mitigation.

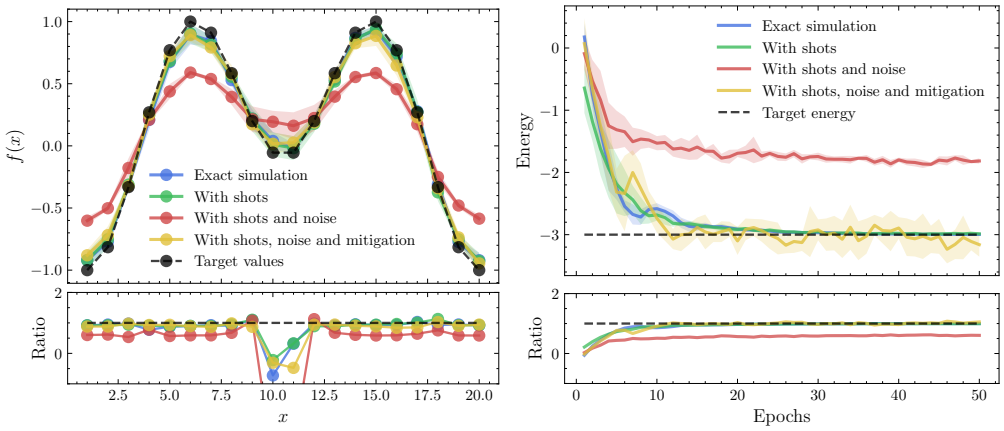


Figure 8.3: Left, results of training a re-uploading architecture to approximate the one-dimensional function of Equation (8.10). Right, results of training a VQE to find the ground state energy of the Hamiltonian in Equation (8.11). In both panels, blue lines correspond to exact simulation, green to noiseless simulation with shots, red to noisy simulation, and yellow to noisy simulation with RTQEM. Predictions are shown in the upper plots, while normalized values are shown in the lower plots.

Ground State Energy Estimation

The second task is a variational quantum eigensolver (VQE) [21] applied to the Hamiltonian

$$H_0 = -\sum_{i=1}^{N} Z_i, \tag{8.11}$$

whose ground state energy is $E_{\text{gs}} = -n$, where n is the number of qubits. We use a hardware-efficient ansatz [22] with alternating layers of R_y and R_z rotations and entangling CNOT layers. The training configuration is summarized in Table 8.2.

Epochs	Runs	Optimizer	Local Pauli Error prob.	Qubits	Shots	Ansatz layers
50	10	Adam($\eta = 0.25$)	0.008	3	500	3

Table 8.2: Training configuration for the VQE simulation.

The results are shown in the right panel of Figure 8.3. Again, RTQEM significantly improves the accuracy of the energy estimation.

The mean squared error (MSE) values for both tasks are reported in Table 8.3. In all cases, RTQEM reduces the error compared to the noisy simulation without mitigation, achieving results close to the noiseless case.

	Exact	Noiseless, shots	Noisy, shots	Noisy, shots, RTQEM
1D fit	$3.5 \cdot 10^{-3}$	$3.4 \cdot 10^{-3}$	$8.0 \cdot 10^{-2}$	$5.6 \cdot 10^{-3}$
Best VQE	$1.2 \cdot 10^{-5}$	$6.4 \cdot 10^{-5}$	1.3	$2.4 \cdot 10^{-7}$
Final VQE	$1.2 \cdot 10^{-5}$	$2.0 \cdot 10^{-4}$	1.4	$2.6 \cdot 10^{-2}$

Table 8.3: Mean squared error (MSE) of the results obtained under different configurations.

In the following we push forward the analysis by evaluating the performance of RTQEM on real quantum hardware in Section 8.4.2, and considering an evolving-noise scenario in Section 8.4.3. In these cases, we focus on one-dimensional regression only, targeting a different function: the up-quark parton distribution function (PDF). Also, to further test the flexibility of the RTQEM approach, we adopt a different error mitigation technique, ICS, presented in Section 8.2.

8.4.2 Deployment on Real Superconducting Hardware

To further validate the RTQEM algorithm, we tested it on real superconducting quantum processors hosted at the Quantum Research Centre (QRC) of the Technology Innovation Institute (TII) [173]. The experiments were carried out using the `Qibo` software stack, with `Qibolab` handling pulse-level control and `Qibocal` providing calibration routines. Two different five-qubit devices were employed: `qw5q`, a QuantWare [174] chip controlled with Qblox [125] electronics, and `iqm5q`, an IQM [175] chip controlled with Zurich Instruments [127] hardware.

Single-qubit training on `qw5q`. We first targeted the one-dimensional u-quark parton distribution function (PDF) using a four-layer single-qubit variational circuit. The training was performed with $N_{\text{data}} = 15$ input points, $N_{\text{shots}} = 500$ per circuit evaluation, and 50 optimization epochs. Two scenarios were compared: unmitigated training and training with RTQEM. The results, shown in Figure 8.4, indicate that the mitigated model was able to exceed the noise-imposed bound visible in the unmitigated case (the black horizontal line). In this case, the RTQEM approach is not providing a significant improvement in terms of MSE, but it is nevertheless able to access a different region of the parameter space, which is unreachable without mitigation. In the next experiment, a longer training is performed, leading to a more pronounced difference between the two scenarios.

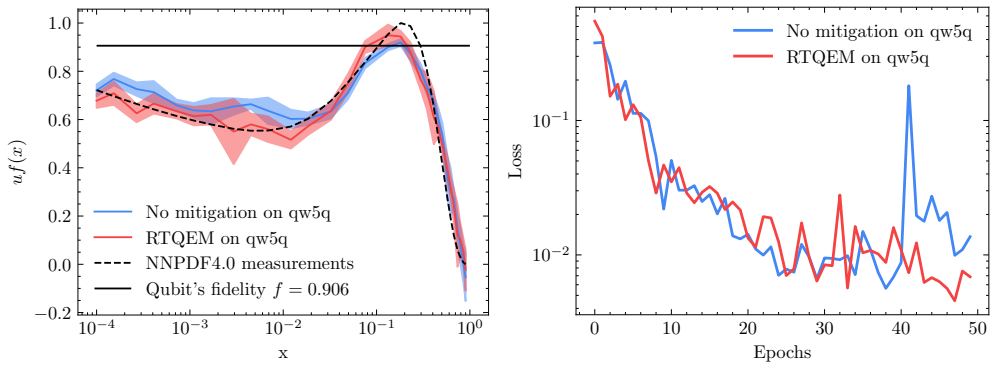


Figure 8.4: Left, estimates of the u-quark PDF obtained by training on the `qw5q` device. The target values (black dashed line) are compared with unmitigated training (blue) and RTQEM training (red). Shaded regions indicate one standard deviation calculated over ten runs. Right, loss function history as a function of optimization epochs. Image taken from [9].

Cross-device validation. To test the portability of the learned parameters, we trained the same model on both `qw5q` and `iqm5q` for one hundred epohcs, starting from identical initial conditions. The parameters obtained from each device were then deployed on `qw5q` for evaluation. Despite the different noise characteristics of the two processors, both sets of parameters produced consistent fits, confirming that RTQEM yields noise-independent solutions. The results are shown in Figure 8.5. For instance, training on `iqm5q` and testing on `qw5q` resulted in an MSE of 3.7×10^{-3}, close to the value obtained when training directly on `qw5q`.

Finally, the parameters obtained from RTQEM training on `qw5q` were deployed on an exact simulator. The resulting predictions closely matched the target function, with an MSE of 1.6×10^{-3}, confirming that the parameters learned under RTQEM are effectively equivalent to those that would be obtained in a noise-free environment.

Overall, these hardware experiments demonstrate that RTQEM not only improves the quality of training on noisy devices but also produces transferable, noise-independent parameters. This suggests that RTQEM can be a key enabler for federated or distributed quantum learning across heterogeneous hardware platforms.

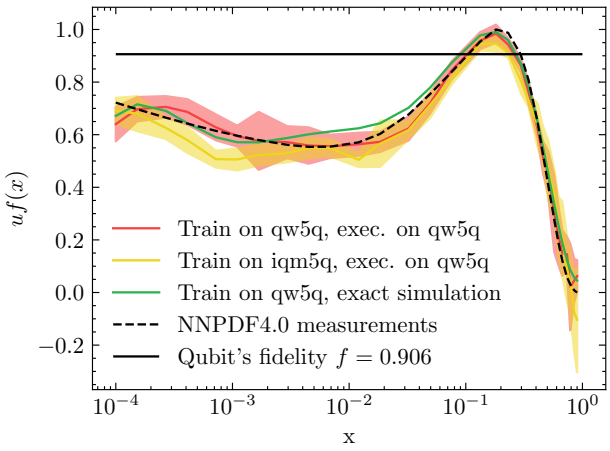


Figure 8.5: Estimates of the u-quark PDF (the training set contains thirty points) obtained using RTQEM training on the better-calibrated qubits of `qw5q` (assignment fidelity $f = 0.906$, red) and `iqm5q` ($f = 0.967$, yellow), each trained for $N_{\text{epochs}} = 100$. The resulting parameter sets are deployed on `qw5q` for evaluation. Predictions from an exact simulation using the best parameters learned on `qw5q` are also shown (green). Target values are indicated by the black dashed line. Shaded regions represent 1σ confidence intervals, obtained from $N_{\text{runs}} = 20$ repetitions. The effective depolarizing parameter is $\lambda_{\text{eff}} = 0.08 \pm 0.02$. Image taken from [9].

8.4.3 How Many Times Is It Convenient to Recompute the Map?

So far, we have considered static noise scenarios or hardware runs where the noise profile remains approximately constant during training. In practice, however, the noise affecting a quantum device can evolve over time due to fluctuations in the environment or calibration drift. It is therefore important to understand how often the mitigation map should be recomputed in order to maintain accuracy without introducing excessive overhead.

To study this, we considered the same problem, namely the fit of the up quark parton distribution function. We repeated our trainings while simulating an evolving noise scenario where the local Pauli parameters $\boldsymbol{q} = (q_x, q_y, q_z)$ follow a random walk. At each training epoch, the parameters are updated as

$$q_j^{(k+1)} = q_j^{(k)} + r\,\delta, \tag{8.12}$$

where $r \in \{-1, +1\}$ and the step length δ is sampled from a normal distribution $\mathcal{N}(0, \sigma_\delta)$. We denote such an evolution as $\text{RW}^N_{\sigma_\delta}$, where N is the number of steps. The readout noise parameter q_M is kept fixed.

In this setting, the RTQEM algorithm monitors the reliability of the mitigation map using the metric defined in Equation 8.9. Whenever the deviation exceeds a threshold ε_ℓ, the map is recomputed. By varying ε_ℓ, we can control how frequently the map is updated.

Figure 8.6 shows the results of four training runs with different thresholds $\varepsilon_\ell = \{0, 0.05, 0.1, 0.2\}$. The evolution of the local Pauli noise parameters starts with $\boldsymbol{q}_0 = (0.005, 0.005, 0.005)$ and follows the random walk $\text{RW}^{100}_{0.002}$. All the random number generators' seeds are fixed, to ensure reproducibility. The corresponding number of map updates during a 100-epoch training was $\{93, 18, 8, 0\}$, respectively. The plot reports the

loss function values evaluated in a noiseless simulation using the parameters obtained during noisy training, allowing us to directly assess how close the optimization remains to the ideal trajectory.

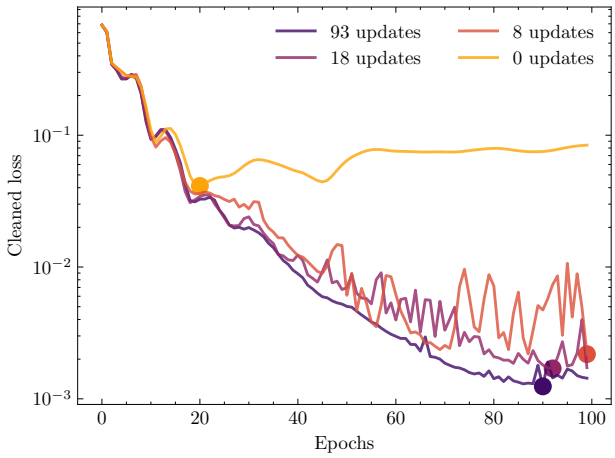


Figure 8.6: Effect of different thresholds ε_ℓ on the frequency of map recomputation in an evolving noise scenario. The curves show the loss function values evaluated in a noiseless simulation using the parameters obtained during noisy training. Lower thresholds lead to more frequent map updates and better tracking of the ideal loss landscape. Image taken from [9].

The results indicate that lowering the threshold improves the training, as the map is updated more often and the optimizer follows a trajectory closer to the ideal one. However, the improvement saturates: even a small number of updates (*e.g.*, 8 times in 100 epochs) already yields a significant reduction in the loss compared to the case with no updates. The difference between updating the map 93 times and 8 times is of order 10^{-3} in the final loss value, showing that frequent recomputation is not always necessary.

In summary, while recomputing the map more often improves accuracy, a moderate update frequency is usually sufficient to capture the evolving noise without incurring excessive overhead. This highlights the importance of choosing an appropriate threshold ε_ℓ to balance performance and efficiency.

8.5 RTQEM as Orchestration Experiment

The real-time quantum error mitigation (RTQEM) framework provides an excellent example of how classical and quantum computing can cooperate in a tightly integrated workflow. Unlike error correction, which requires ultra-low-latency feedback loops, RTQEM leverages the natural batch structure of quantum machine learning tasks to distribute responsibilities between the quantum processor and the classical controller. The quantum device generates large datasets of noisy measurements, while the classical side processes them to build and apply mitigation maps. This continuous interplay makes RTQEM a valuable testbed for exploring hybrid orchestration strategies.

Looking ahead, many intriguing extensions can be envisioned to further boost the efficiency of this cooperation. One particularly promising direction is to move part of the mitigation logic directly onto custom FPGA boards that already interface with the quantum hardware. In such a setup, raw measurement samples could be collected, ex-

pectation values computed, and mitigation maps applied entirely on-board. This would eliminate the need for frequent communication between the quantum processor and an external classical node, thereby reducing latency. The two components would only need to synchronize when the noise map itself requires recomputation, which occurs much less frequently.

This approach would not only accelerate the RTQEM pipeline but also open the door to more advanced hybrid strategies, where classical pre-processing and quantum execution are seamlessly interleaved at the hardware level. In this sense, RTQEM is more than just an error mitigation routine: it is a proof-of-principle for the kind of hybrid quantum–classical orchestration that will be essential in the noisy intermediate-scale quantum (NISQ) era and beyond.

CHAPTER 9

Combining Classical Simulation Techniques

In this chapter, we discuss a novel approach to represent quantum states with classical simulation techniques. In particular, we focus on the recently introduced hybrid stabilizer-matrix product operators (HSMPO) method [176], which combines the strengths of stabilizer formalism and tensor network methods to efficiently simulate many-body quantum dynamics. After introducing the theoretical framework, we present `mpstab`, a state-of-the-art quantum system simulator that implements the HSMPO approach, enabling the study of complex quantum circuits that are otherwise intractable with traditional methods.

> The `mpstab` package is still under development and the code is available upon request.

9.1 Hybrid Stabilizer-MPO

The classical simulation of many-body quantum dynamics is a central challenge for both theoretical investigations and the validation of near-term quantum devices. Among the classical techniques mentioned in Chapter 4, we focus here on tensor networks and stabilizer formalism. Each of these comes with complementary strengths and limitations. On the one hand, MPS methods can approximate generic quantum states and efficiently capture local observables, provided that the entanglement entropy remains bounded by a moderate bond dimension χ. However, under generic unitary evolution entanglement typically grows linearly with time, leading to an exponential increase in χ and to the eventual breakdown of the method [103]. On the other hand, the stabilizer formalism allows polynomial-cost simulation of Clifford circuits [36], but it cannot accommodate non-Clifford gates, which introduce the quantum resource known as non-stabilizerness or "magic" [38].

The recently introduced *Hybrid Stabilizer-MPO* (HSMPO) framework [176] was developed to bridge these two approaches by combining the efficient handling of Clifford operations with the compact representation of non-Clifford rotations as tensor networks (See Figure 9.1 for an illustration). The key insight is that any universal quantum circuit can be decomposed into alternating layers of Clifford operations and local Pauli rotations, such as the T gate or, more generally, $R_\mu(\theta) = e^{-i(\theta/2)\sigma_\mu}$. Formally, a generic unitary evolution can be expressed as

$$U = C_0 R_1 C_1 R_2 C_2 \cdots R_m C_m, \tag{9.1}$$

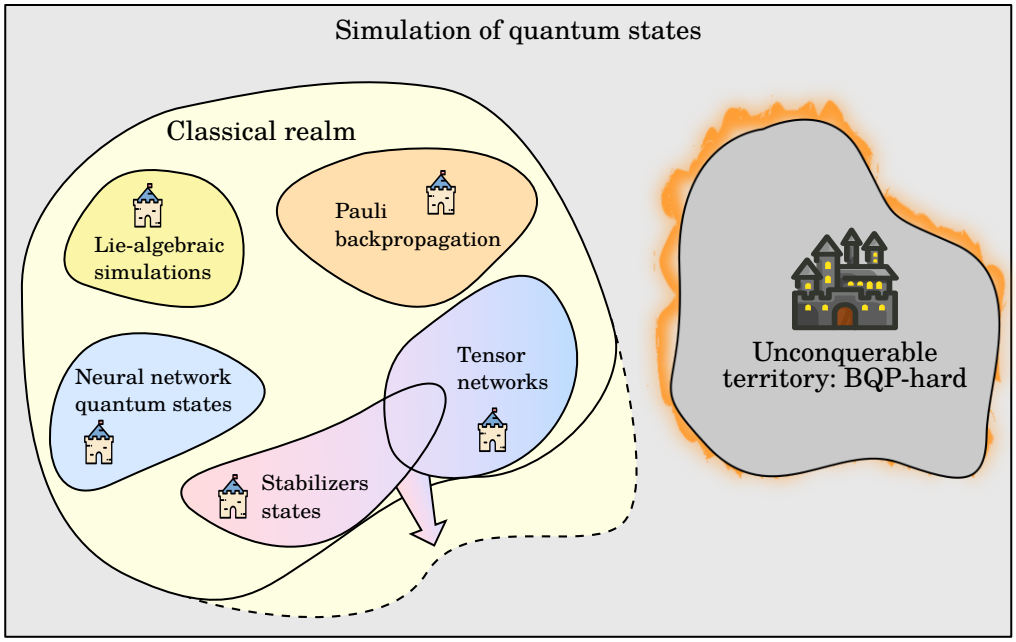


Figure 9.1: Classical simulation techniques can be an optimal choice depending on the structure of the dynamics we aim to represent. In this chapter, we explore the combination of stabilizer formalism and tensor networks expanding the reach of classical simulations with respect to what we showed in Figure 4.5.

where the C_j are Clifford unitaries and the R_j are single-qubit rotations around Pauli axes. An illustrative example of such a circuit is shown in Figure 9.2.

In this sequence of operations, what is challenging to simulate classically is the non-Clifford rotations R_j, which do not preserve the stabilizer structure of the state. In practice, if tracking a single operator through a Clifford circuit results into another single Pauli operator, doing the same through a circuit presenting t local rotations typically produces a linear combination of up to 2^t Pauli strings.

To understand this, recall that Clifford gates preserve the Pauli group under conjugation: for any Pauli operator $P \in \{X, Y, Z\}$ and any Clifford unitary C, we have

$$CPC^\dagger \in \{\pm X, \pm Y, \pm Z\}, \tag{9.2}$$

so the Pauli structure is preserved exactly. By contrast, a local Pauli rotation

$$R_\mu(\theta) = \exp\left(-i\tfrac{\theta}{2}\sigma_\mu\right), \qquad \mu \in \{X, Y, Z\}, \tag{9.3}$$

does not generally map a Pauli operator to a single Pauli operator, but instead to a *linear combination* of two Paulis. Conjugating P by $R_\mu(\theta)$ amounts to rotating the Bloch vector of P about the axis μ by an angle θ. Algebraically, one finds

$$R_Z(\theta)\, X\, R_Z(\theta)^\dagger = \cos\theta\, X + \sin\theta\, Y, \tag{9.4}$$

$$R_Z(\theta)\, Y\, R_Z(\theta)^\dagger = \cos\theta\, Y - \sin\theta\, X, \tag{9.5}$$

$$R_Z(\theta)\, Z\, R_Z(\theta)^\dagger = Z, \tag{9.6}$$

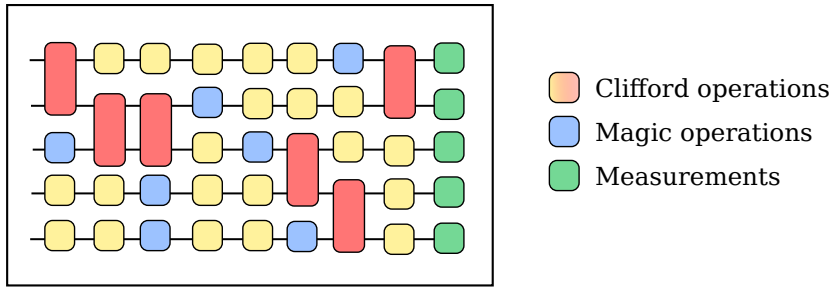


Figure 9.2: A quantum circuit composed of alternating layers of Clifford operations (red and yellow gates) and single-qubit magic operations (blue gates).

and analogous relations hold for rotations around X or Y. Hence, if P commutes with σ_μ it remains invariant, whereas if it anticommutes it expands into a linear combination of two Paulis lying in the plane orthogonal to μ. Thus, a single Pauli operator can expand into *at most two* Pauli operators under such local rotations. Consequently, after t local rotations a single tracked operator can expand into up to 2^t Pauli terms. This is the source of the exponential complexity in simulating Clifford+magic circuits.

Going back to the sequential structure introduced in Equation (9.1), we also note that applying a Clifford operation C to a local Pauli rotation $R_\mu(\theta)$ results in a *non-local* Pauli rotation:

$$CR_\mu(\theta)C^\dagger = \cos\left(\frac{\theta}{2}\right)I - i\sin\left(\frac{\theta}{2}\right)C\sigma_\mu C^\dagger = \cos\left(\frac{\theta}{2}\right)I - i\sin\left(\frac{\theta}{2}\right)\Sigma_\gamma = T_1T_2\cdots T_n, \quad (9.7)$$

where Σ_γ is a Pauli string that may act non-trivially on many qubits and T_j are local rotations. We can exploit this property by adding identities $C_jC_j^\dagger$ between each rotation and Clifford block of the sequence in Equation (9.1), rewriting the entire unitary as

$$U = R'_1R'_2\cdots R'_m\prod_{j=0}^{m} C_j, \quad (9.8)$$

where each R'_j is a rotation about a possibly non-local Pauli string Σ_γ, and the Clifford blocks are collected at the end of the sequence.

This rewriting is advantageous because each Pauli-string rotation admits an exact representation as a Matrix Product Operator (MPO) of bond dimension at most two [176]. Indeed,

$$R' = \cos(\theta/2)\, I - i\sin(\theta/2)\,\Sigma_\gamma, \quad (9.9)$$

is a linear combination of only two tensor-product operators: the identity and a single Pauli string. In MPO form, the auxiliary index carries two channels, corresponding respectively to the identity branch and to the Pauli branch, keeping the representation compact even if Σ_γ acts non-trivially on many qubits. A graphical representation of this MPO structure is shown in Figure 9.3.

Within the HSMPO scheme, Clifford blocks are never applied directly to the state. Instead, they are transferred to the observables, which transform as

$$O \mapsto C^\dagger OC, \quad (9.10)$$

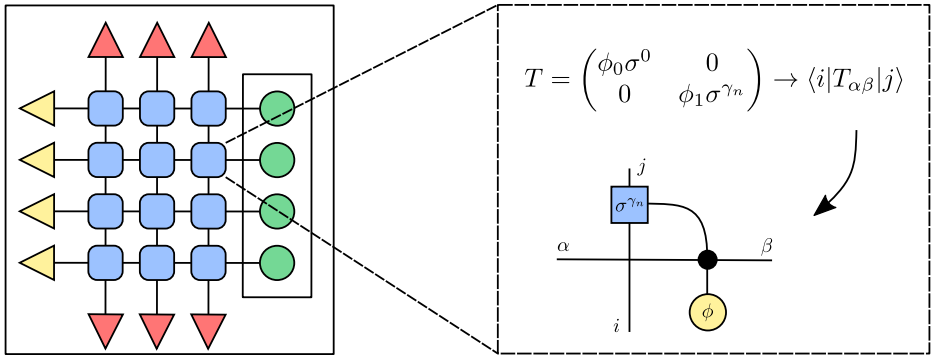


Figure 9.3: A single Pauli-string rotation can be represented exactly as an MPO of bond dimension two, with one channel for the identity branch and one for the Pauli branch. After disentangling the local rotations as shown in Equation (9.7), each local rotation of the original circuit becomes a compact MPO layer. The full circuit appears then as the diagram in the left panel, while a single building block is shown in the right panel.

remaining simple Pauli strings. The expectation value

$$\langle\psi|U^\dagger OU|\psi\rangle \tag{9.11}$$

is therefore rewritten as

$$\langle\psi|(R'_1\cdots R'_m)^\dagger\,(C^\dagger OC)\,(R'_1\cdots R'_m)|\psi\rangle, \tag{9.12}$$

where the state is evolved only under MPO layers of small bond dimension, while the Clifford contribution is absorbed into the observable. This effectively suppresses the entanglement growth in the MPS representation, as the most entangling Clifford dynamics is displaced onto Pauli observables that remain tractable.

Graphically (see left panel of Figure 9.3), this procedure can be interpreted as adding an auxiliary qubit for each magic rotation, such that the global network becomes two-dimensional, with weak entanglement across rows and columns [176]. The auxiliary structure accounts for the "magic" contribution, while the stabilizer backbone ensures efficiency. As a result, HSMPO enables the simulation of Clifford+magic circuits for longer times and larger system sizes than either pure tensor-network or pure stabilizer methods.

This framework has already demonstrated significant improvements in scenarios such as random Clifford+T circuits and Floquet dynamics with sparse non-Clifford rotations [176]. Moreover, it provides a natural foundation for integrating advanced error mitigation strategies. In particular, we are working on leveraging the HSMPO structure to design surrogate circuits for learning-based error mitigation, exploiting its ability to balance entanglement and non-stabilizerness while keeping the simulation classically tractable.

In summary, the hybrid stabilizer-MPO approach exemplifies how the orchestration of complementary classical methods can substantially extend the frontier of classically accessible quantum dynamics. By treating Clifford structure exactly and incorporating non-Clifford rotations through compact MPO layers, HSMPO offers a precise and scalable methodology to simulate many-body quantum circuits, making it a cornerstone for future hybrid simulation and error mitigation frameworks.

9.2 `mpstab`: a Cutting-Edge Quantum System Simulator

The previously described HSMPO framework has been introduced in [176] as theoretical method. To make it accessible to the community, we have developed `mpstab`, a quantum system simulator that implements the HSMPO approach. `mpstab` is an open-source library written in Python, designed for ease of use and integration with the `Qibo` framework. It provides simple interface to define quantum computing ansätze, translate them into HSMPO format, and perform efficient simulations of quantum circuits that include both Clifford and non-Clifford operations. At its current status, the library supports a `NumPy` backend for CPU execution, with plans to extend support to GPU backends in the future. Also, we plan to support differentiable backends, enabling gradient-based optimization of quantum circuits within the HSMPO framework.

9.2.1 Software Design

The `mpstab` package implements the HSMPO workflow by composing four layers: *i)* a Clifford simulator based on tableau updates, *ii)* a lightweight tensor-network engine, *iii)* an `CircuitMPS` constructor that realizes MPO updates on an MPS in canonical form, and *iv)* an `HybridSurrogate` driver that partitions the input circuit, conjugates generators in the Heisenberg picture, and evaluates observables with bond–2 MPO layers [36, 103, 176].

Clifford simulator (tableaus and Pauli strings). Stabilizer propagation is implemented using an "XZ" two–bit encoding of Pauli strings and Aaronson and Gottesman–style tableaus. A Pauli string on n qubits is packed into a $2n$-bit integer; helper routines convert to/from human–readable strings, track phases, and implement conjugation rules.

A `Pauli` object supports phase–correct updates and string manipulation, while `Tableau` classes encode gate–wise conjugations. Each Clifford gate (`H`, `S`, `X`, `Z`, `CNOT`, `SWAP`) is a concrete `Tableau` specifying how X and Z components transform and their application corresponds to fast bit–wise updates.

Given a `Qibo`'s Clifford circuit C, back–propagation in the Heisenberg picture is exposed by

$$P \;\mapsto\; C^{\dagger} P C, \tag{9.13}$$

implemented by iterating the gate queue in reverse, daggerring parameters when present, and updating the `Pauli` via the corresponding `Tableau`. This routine is used both to *a)* conjugate local rotation generators into global Pauli strings and *b)* displace Clifford blocks to measurement observables, as required by HSMPO (see Section 9.1).

Tensor network core (MPO/MPS primitives). The tensor-network layer represents networks as a directed multigraph whose nodes carry dense `NumPy` tensors and whose edges carry axis–pair directions (input/output legs) and metadata.

The base `TensorNetwork` offers: adding tensors, directed edges with axis binding, contractions along named "links", canonicalization helpers (*e.g.*, inserting/splitting via singular value decomposition (SVD) with cut threshold), conjugation, partial traces, and simple drawing utilities.

On top, `MPO` is a thin wrapper that creates a 1D chain of rank–3/4 tensors with explicit physical and link legs; it is used both for gates and observables. Library MPOs include:

1. an exact bond-2 implementation of R'_j of the form of Equation (9.7) for any Pauli string Σ_γ (class `PauliExp`);

2. fixed MPOs for elementary Clifford gates (`H`, `S`, `X`, `CNOT`, `CZ`, `SWAP` and inverses), plus `PauliMPO` for observables.

These components provide the exact bond-2 layers central to HSMPO.

`CircuitMPS`: MPS construction and MPO application. `CircuitMPS` constructs an n-site open-boundary MPS with diagonal link tensors explicitly represented as one-leg nodes, maintaining a canonical form suitable for sequential MPO application. The initializer takes a list of single-qubit state vectors and assembles three-leg site tensors ($|\text{phys}\rangle \otimes$ left $\otimes$ right) with unit links; a one-leg "measurement" dummy is attached to each site to anchor physical legs and simplify contractions. The method `apply(MPO, sites)`

1. unions the MPO graph to the MPS along the specified contiguous range;
2. links physical legs;
3. performs gate application via local contractions;
4. re-canonicalizes by inserting SVD factorizations and truncating to the requested maximum bond dimension (with a numerical SVD cutoff).

The method `expval(MPO)` forms the standard bra-ket sandwich by conjugating the MPS, attaching the MPO on the physical legs, and contracting from the edges to return a scalar $\langle\psi|O|\psi\rangle$.

`HybridSurrogate`: end-to-end HSMPO evaluation. `HybridSurrogate` is the driver that realizes the HSMPO flow for a user ansatz. Its `expectation(observable)` executes:

- **State initialization:** If the user provides an initial state, we first construct an MPS representation of this state, which is embedded in the `CircuitMPS` object. We currently support only initial states prepared as product states of single-qubit kets.
- **Circuit partition:** Obtain from the `Ansatz` a decomposition of the full circuit into a list of parameterized single-qubit rotations ("magic gates") interleaved with Clifford blocks; also obtain the cumulative Clifford-only circuit.
- **Heisenberg conjugation:** For each rotation gate in order, conjugate its local generator (*e.g.*, X, Y, Z) by the prefix Clifford to produce the global Pauli string Σ_{γ_j} using the tableau engine (reverse-walking the prefix queue). Likewise, conjugate the user observable O by the final Clifford C to get a Pauli string $O' = C^\dagger O C$.
- **MPO updates and readout:** Apply each rotation as a bond–2 `PauliExp` MPO to the `CircuitMPS` over the support of Σ_{γ_j} (the class checks that the target sites are contiguous), with on-the-fly re-canonicalization and optional bond truncation. Finally, evaluate $\langle O'\rangle$ by contracting `PauliMPO(`O'`)` with the evolved MPS. This yields the exact HSMPO expectation for the provided bond cutoff (no stochastic sampling).

Surrogates and magic reduction. The `HybridSurrogate` also supports method that can generate lower-magic surrogates by probabilistically replacing non-Clifford single-qubit rotations with nearest-Clifford angles (or random Clifford choices). These surrogates preserve the circuit frame while reducing non-stabilizerness and can be used within learning-based error mitigation schemes as described in Section 3.3.2.

Ansatz layer and transpilation. The `Ansatz` hierarchy exposes: circuit access, optional hardware-aware transpilation, and a partitioner that returns the triple (*magic_gates*, *clifford_only*, *full_circuit*). For concrete ansätze, the partitioner constructs two symmetric Clifford halves around a central local rotation and records the positions and parameters of magic gates inside the full queue. This information drives the Heisenberg conjugations in the driver.

Interfaces and limits. Interfaces assume `qibo` circuits for front-end construction.

`CircuitMPS.apply` requires the target MPO to act on a contiguous, ascending range of sites (the code asserts this). The bond growth is controlled by the user via `max_bond_dimension` and an SVD cutoff. Within these constraints the method is exact for each Pauli–string rotation, matching the HSMPO prescription [176]. A unit test compares `HybridSurrogate` to exact statevector expectations from `qibo`, confirming numerical agreement to tight tolerance.

9.3 `mpstab` as Tool for Quantum Computing Research

The presented framework is helpful to explore the dynamics of quantum circuits and to represent longer evolutions than what is possible with pure MPS or pure stabilizer methods. This makes `mpstab` a valuable tool for quantum computing research, especially in the context of near-term devices where noise and errors are prevalent.

Moreover, the final MPO structure allows adding an extra degree of freedom to control the amount of entanglement in the representation, following exactly the same spirit of tensor-network methods. This provides us with a flexible classical tool which combines the strengths of stabilizer formalism and tensor networks, enabling the study of complex quantum circuits that are otherwise intractable with traditional methods.

An example of application of `mpstab` is in the context of learning-based error mitigation, as described in Section 3.3.2. The HSMPO representation can be used to design surrogate circuits to compose the training set of circuits presented in Section 3.3.2. So far, only exact simulation techniques have been tested to generate the training set, but `mpstab` also allows to explore surrogate circuits with a controlled amount of entanglement and non-stabilizerness, while keeping the simulation classically tractable. This opens up new possibilities for designing effective error mitigation strategies that leverage the strengths of both classical and quantum computing.

Part IV
Full-Stack Applications

CHAPTER 10

Multi-Variable Integration with a Quantum Computer

With this chapter, we start a new part of the thesis, focused on applications of quantum computing to more specific problems. Most of the applications will be in the context of high-energy physics (HEP), and in particular in the context of parton distribution functions (PDFs) determination. HEP is one of the fields where quantum computing is explored more actively, and where there are efforts to understand its potential benefits [177].

This chapter explores the use of quantum circuits to approximate multi-variable integrals, a common challenge in various scientific and engineering contexts. The approach leverages the ability of variational quantum circuits to serve as flexible function approximators, training the circuit's derivative to reconstruct the integral value.

> The discussion will follow the structure of our original publication [11].
>
> The code accompanying this work is available at:
> https://github.com/qiboteam/QiNNtegrate.

10.1 The Multi-Variable Integration Problem

Multi-variable integration is a challenging problem in various industrial and research contexts. At its core, the problem consists in solving integrals of the form

$$I(\boldsymbol{\alpha}) = \int_{\boldsymbol{x_a}}^{\boldsymbol{x_b}} g(\boldsymbol{\alpha}; \boldsymbol{x}) \mathrm{d}^n \boldsymbol{x}, \tag{10.1}$$

where eventual parameters $\boldsymbol{\alpha}$ and integration variables $\boldsymbol{x}$ can be multi dimensional.

Depending on the specific target integrand function and integration region, there may be several reasonable choices about the strategy to be used to tackle Equation (10.1). Considering the classical computing realm, sparse grid [178], quadrature methods [179], quasi Monte Carlo [180] and Monte Carlo methods [181] are valuable options. In particular, (quasi) Monte Carlo techniques are known to handle non-analytically solvable problems well, but they require a number of generated points which scales exponentially with the problem dimensionality. This incurs a cost that increases directly with the complexity of the problem; whether considering points generated from probability distributions or points taken on a grid, the mentioned techniques require the calculation of the integrand value for each considered point.

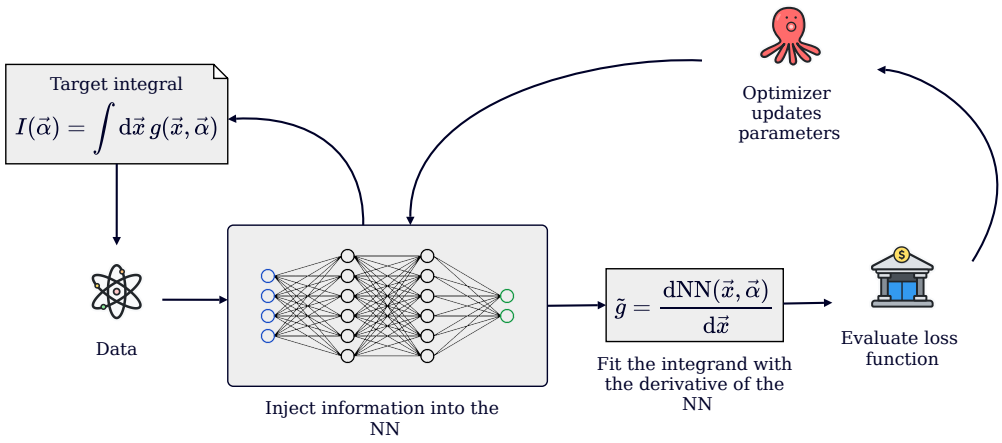


Figure 10.1: Schematic representation of the procedure introduced in [10]: the derivative of a neural network is trained to approximate a target integrand function, so that the neural network itself is then used to reconstruct the integral value.

Recently, machine learning strategies have been considered to handle multi-variable integration problems [182, 183, 10], with the intention of moving computational complexity in a different direction. In fact, unlike the techniques mentioned earlier, using machine learning models the problem becomes training the model to approximate the target as accurately as possible. This involves a training cost, but once a satisfactory model is obtained, the cost of recomputing the same function for different inputs becomes negligible. In particular, in Reference [10], the derivative of a neural network is trained to approximate a target integrand function. Once a satisfactory accuracy is reached, the neural network itself becomes an approximation of the primitive function. A schematic representation of such a procedure is shown in Figure 10.1.

A similar approach can be applied in the context of quantum machine learning, where a quantum circuit can be used in place of a neural network. We keep unchanged the core idea, but at the same time we leverage the fact that the derivative of expectation values with respect to the parameters of a circuit can be obtained by running the same circuit and shifting the target parameter.

Before moving to the details of the method, we mention that other quantum algorithms for numerical integration are being explored, especially in the context of HEP [184, 185, 186, 187]

10.2 Solving Integrals with Quantum Circuits

In this section we provide a detailed description of how quantum circuits can be used to approximate primitive functions of multidimensional integrands, and how this allows us to compute definite integrals efficiently. Since the reader has already been introduced to the basic notions of quantum machine learning and parameter-shift differentiation in Chapter 2 and Section 2.3.3 respectively, here we only recall the notation and focus on its application to the integration problem.

10.2.1 Quantum Circuits as Primitive Functions

Let $g(\boldsymbol{\alpha};\boldsymbol{x})$ denote a real function of n input variables $\boldsymbol{x} = (x_1,\ldots,x_n)$ and external parameters $\boldsymbol{\alpha}$. We are interested in computing the definite integral of Equation (10.1), where the bounds $\boldsymbol{x_a}, \boldsymbol{x_b}$ may differ along each dimension.

The key idea is to construct a parametric quantum circuit $C(\boldsymbol{x} \mid \boldsymbol{\theta})$ such that the expectation value of a chosen observable O on the output state defines a function $G(\boldsymbol{x};\boldsymbol{\alpha})$,

$$G(\boldsymbol{x};\boldsymbol{\alpha}) \;=\; \langle 0|\, C^{\dagger}(\boldsymbol{x} \mid \boldsymbol{\theta})\, O\, C(\boldsymbol{x} \mid \boldsymbol{\theta})\, |0\rangle, \tag{10.2}$$

whose derivative with respect to each integration variable reproduces the target integrand,

$$\frac{\partial}{\partial x_i} G(\boldsymbol{x};\boldsymbol{\alpha}) \simeq g(\boldsymbol{\alpha};\boldsymbol{x}), \qquad i = 1,\ldots,n. \tag{10.3}$$

In this framework, G plays the role of an approximate primitive function for g. Once G is learned, the definite integral reduces to the evaluation of G at the corners of the integration domain, as we detail below.

10.2.2 Reconstruction of Definite Integrals

Once the circuit has been trained such that Equation (10.3) is satisfied, the function $G(\boldsymbol{x};\boldsymbol{\alpha})$ acts as an approximate antiderivative of the target integrand $g(\boldsymbol{\alpha};\boldsymbol{x})$. The definite integral over the n–dimensional hyper-rectangle $[x_{1,a}, x_{1,b}] \times \cdots \times [x_{n,a}, x_{n,b}]$ can then be written using the fundamental theorem of calculus as

$$I(\boldsymbol{\alpha}) = \sum_{\ell_1 \in \{a,b\}} \cdots \sum_{\ell_n \in \{a,b\}} (-1)^{\#\{\ell_i = a\}}\, G(x_{1,\ell_1},\ldots,x_{n,\ell_n};\boldsymbol{\alpha}), \tag{10.4}$$

where $\#\{\ell_i = a\}$ denotes the number of indices ℓ_i set to the lower boundary. In the one–dimensional case this reduces to the usual subtraction at the interval endpoints,

$$I(\alpha) \;=\; G(x_b;\alpha) - G(x_a;\alpha). \tag{10.5}$$

The evaluation of Equation (10.4) requires the value of G at the 2^n corners of the integration region. During training, computing the parameter–shift derivatives involves $2n$ circuit evaluations per training point, while at inference only a single circuit execution per corner is needed. This allows the definite integral to be reconstructed once the model parameters have been optimized.

10.2.3 Circuit Ansatz and Parameterization

The circuit ansatz must be flexible enough to approximate a broad class of primitive functions while remaining shallow enough to be executed on present-day hardware. Following the architecture adopted in Reference [165], we consider an ansatz consisting of alternating layers of single-qubit rotations and entangling gates. Each input variable x_i is re–uploaded into the circuit through rotation gates of the form $R_\mu(\omega_i x_i)$, where $\mu \in \{x, y, z\}$ specifies the rotation axis and ω_i is a trainable frequency. The set of variational parameters $\boldsymbol{\theta}$ controls both the angles of the additional rotations and the weights of entangling gates. This strategy allows the circuit to capture nontrivial dependencies on the inputs by repeated encoding, in line with the data re–uploading paradigm.

The observable O is taken to be a tensor product of Pauli operators. In practice, we adopt $O = \sigma_z$ on a designated readout qubit, which simplifies both simulation and hardware implementation.

10.2.4 Training Procedure

The training proceeds by minimizing the loss function

$$\mathcal{C}(\boldsymbol{\theta}) \;=\; \frac{1}{N_{\text{train}}} \sum_{j=1}^{N_{\text{train}}} \left(g(\boldsymbol{\alpha}; \boldsymbol{x}_j) - \frac{\partial}{\partial x_i} G(\boldsymbol{x}_j; \boldsymbol{\alpha}) \right)^2 , \tag{10.6}$$

where the derivative of G is computed with the parameter–shift rule. The training set $\{\boldsymbol{x}_j\}$ consists of points sampled uniformly from the integration domain, and the cost is averaged over all integration variables. For the optimizer we employed L–BFGS [188], which proved effective in achieving rapid convergence for the relatively small number of variational parameters considered here. Other optimizers, such as gradient descent with momentum or evolutionary strategies, could be used as well.

We emphasize that the training cost scales with the number of training points and the number of variables, since at each point one needs $2n$ circuit evaluations to compute all parameter–shift derivatives. However, this training is performed only once. After optimization, the evaluation of integrals for new parameter values $\boldsymbol{\alpha}$ requires only the inference cost of evaluating G at the corners of the domain.

10.2.5 Summary of the Procedure

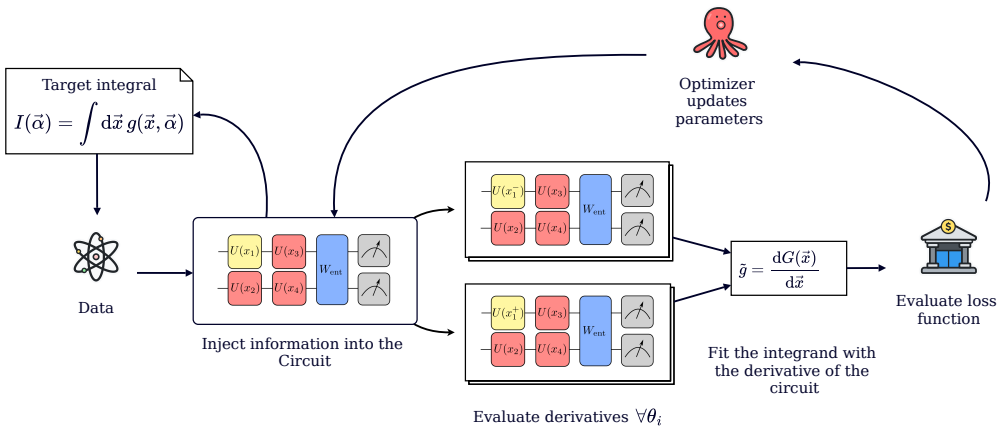


Figure 10.2: Schematic representation of the procedure to compute multidimensional integrals using quantum circuits as primitive functions.

To summarize, the methodology involves the following steps:

1. Select an ansatz $C(\boldsymbol{x} \mid \boldsymbol{\theta})$ with data re–uploading encoding of input variables.
2. Train the parameters $\boldsymbol{\theta}$ by minimizing Equation (10.6), so that parameter–shift derivatives match the target integrand $g(\boldsymbol{\alpha}; \boldsymbol{x})$ on a training set.
3. Once trained, obtain the primitive approximation $G(\boldsymbol{x}; \boldsymbol{\alpha})$ from Equation (10.2).
4. Compute definite integrals using the fundamental theorem of calculus in Equation (10.4), requiring only evaluations of G at the 2^n corners of the integration box.

An illustration of the overall procedure is provided in Figure 10.2. This establishes the quantum–circuit analogue of the neural–network integration scheme of Reference [10].

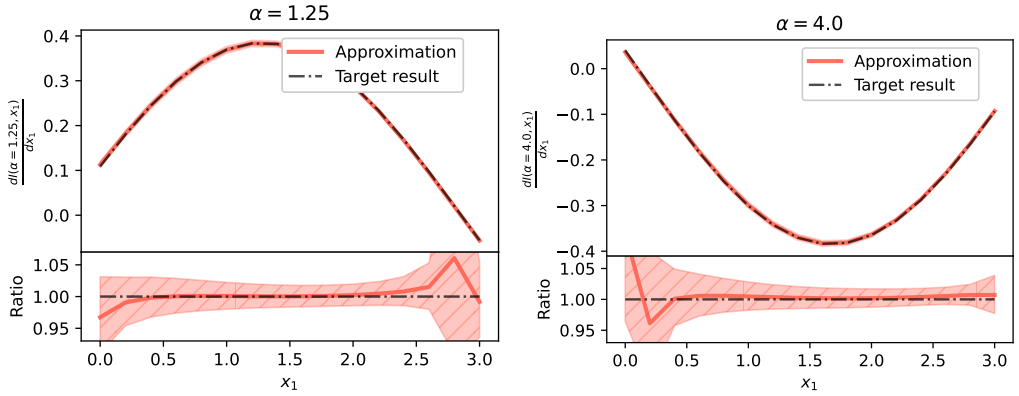


Figure 10.3: Differential distribution of the integral of Equation (10.7) with respect to x_1, shown for different values of α_0 and with $\boldsymbol{\alpha} = \{1,\ 2,\ \frac{1}{2}\}$. The ranges selected for x_1 are the same as those used for the integration over all other variables. Results were obtained through exact state-vector simulation. Training employed 100 sample points in x and 10 values of α_0 uniformly chosen from $(0, 5)$, optimized with L-BFGS for 200–300 iterations. Error bands are obtained by retraining the circuit with different random seeds. Figure taken from [11].

10.3 Validation Experiments

In order to showcase the possibilities of the methodology presented in this paper we are going to use a VQC for two different target functions. We will first show the flexibility of the method to obtain total or partial integrals and differential distributions, and then we will apply to a practical case in which the approach can introduce an utility. Both examples are implemented in the public code which accompanies this paper [189].

10.3.1 Toy Model

For our first example we utilize the reuploading ansatz introduced in [165] to approximate a d-dimensional trigonometric function:

$$g(\boldsymbol{x}) = \cos(\boldsymbol{\alpha} \cdot \boldsymbol{x} + \alpha_0), \tag{10.7}$$

with $\boldsymbol{x}$ and $\boldsymbol{\alpha}$ d-dimensional vectors. The integral of Equation (10.7), while trivial to perform analytically, will serve to demonstrate how training one single circuit to obtain the primitive,

$$I(\boldsymbol{\alpha}; \boldsymbol{x}) = \int g(\boldsymbol{\alpha}; \boldsymbol{x}) \mathrm{d}\boldsymbol{x}, , \tag{10.8}$$

can provide us the flexibility to obtain other derived quantities.

For instance, we might be interested in the differential distributions $\frac{\mathrm{d}I(\boldsymbol{\alpha};\boldsymbol{x})}{\mathrm{d}x_i}$ for a given i and for different values of one of the parameters $\boldsymbol{\alpha}$. In general, this would require performing the numerical integration once per choice of i, per bin in the distribution, and per choice of α. By having a surrogate for $I(\boldsymbol{\alpha}; \boldsymbol{x})$ we can obtain each distribution as seen in Figure 10.3, where we collect results obtained by training the model with exact state vector simulation of the quantum circuits.

In our experiment we have considered $d = 4$. In Figure 10.3 we have plotted the differential distribution

$$\frac{\mathrm{d}I(\boldsymbol{\alpha}; x_1)}{\mathrm{d}x_1}, \tag{10.9}$$

for two different values of α_0 within the training range $(0, 5)$. All other parameters have remained fixed in order to minimize the computing cost in this toy-model example. The training range for the integrated variables (x_1, x_2, x_3) has been $(0, 3.5)$. In the plots we choose to integrate x_2 and x_3 from 0 to 3 for every value of x_1, but any choice of integration limits within the integration range would be possible.

A shortcoming of this approach is the lack of an intrinsic uncertainty associated to the numerical integration. Following the suggestion in Reference [10], one can use an ensemble of replicas of the model trained on the same data (with different seeds) and interpret the resulting spread as an error estimate. We follow the same strategy here by training an ensemble of circuits with randomized training points, which provides the error bands shown in Figure 10.3.

Other numerical methods provide shortcuts to obtain similar derived quantities. For instance, MC integration methods allow binning quantities that depend on the integration variables (provided that the desired distributions are known a priori). However, a change in the parameter α_0 requires a new integration. Once we have a circuit that approximates Equation (10.8), derived quantities inside the training range are accessible without further runs.

There is no free lunch: the computational cost is transferred to training (and to the evaluation of the surrogate during training), but the outcome is a flexible representation that can be reused. As with Monte Carlo, accuracy can be improved by increasing the computational budget (more training points, longer optimization). The model also allows generating an essentially unlimited amount of synthetic evaluations at inference time.

10.3.2 The u-quark PDF

We now consider a realistic use case: the integration of an empirically available function, the unnormalized u-quark distribution $u(x, Q)$ from the NNPDF4.0 interpolation grids. The target is the momentum fraction

$$I_u(Q) = \int_{10^{-4}}^{0.7} x\, u(x, Q)\, \mathrm{d}x, \tag{10.10}$$

which plays a role in normalization and sum–rule constraints.

We use the qPDF-inspired two-dimensional ansatz described in Section 10.2.3. Training is performed on values of $x \in [10^{-4}, 0.7]$ and on a set of Q points chosen between 1.65 GeV and 40 GeV. For fixed Q we train the derivative of the circuit with respect to x to match $x\, u(x, Q)$. Figure 10.4 shows examples of the fit for two representative Q values, obtained in exact state vector simulation.

Once the derivative fit is satisfactory, we compute the integral of Equation (10.10) by evaluating the learned primitive $G(x, Q)$ at the endpoints in x for each Q. To demonstrate generalizability, we trained the circuit across a dense set of Q values and reconstructed $I_u(Q)$ for $N_q = 20$ points. The integrals in Figure 10.5 are computed with shot-noise simulations: for each Q we perform $N_{\mathrm{runs}} = 100$ predictions and each prediction averages $N_{\mathrm{shots}} = 10^6$ executions. The orange line and the shaded belt correspond to the mean and one standard deviation over the N_{runs} predictions. The average relative percentage error (lower panel) is calculated over the N_{runs} realizations.

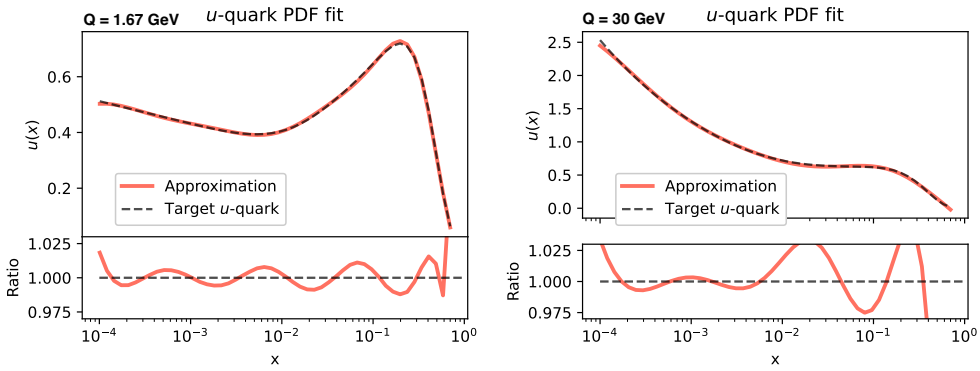


Figure 10.4: Comparison between the fit to the derivative of the circuit (red line) and the target values extracted from the interpolation grids for the central NNPDF4.0 replica of the u-quark, shown at fixed scales $Q = 1.67$ GeV (the fitting scale of NNPDF4.0) and $Q = 30$ GeV. The training set consists of approximately 100 points for each of the five fixed values of Q, chosen between the initial $Q = 1.67$ GeV and $Q = 40$ GeV. Results are obtained via exact state-vector simulation. In Figure 10.5, we employ the same strategy, extending the training over a wider range of Q, to plot the integrand as a function of the energy. Figure taken from [11].

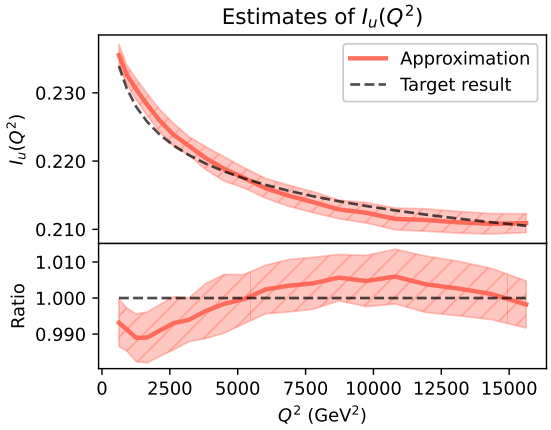


Figure 10.5: Top: integral of $x\,u(x, Q)$ evaluated at $N_q = 20$ different values of Q. Results are obtained from circuit simulations including shot noise. For each Q, we perform $N_{\text{runs}} = 100$ independent predictions, where each prediction is obtained by executing the circuit $N_{\text{shots}} = 10^6$ times. The orange line and its confidence band are computed as the mean and one standard deviation over the N_{runs} prediction sets. Bottom: average relative percentage error evaluated from the same set of predictions. Training was carried out on approximately 100 values of Q, uniformly spaced in Q^2, and required about 20 h on a 32-core machine. Figure from [11].

In this shot–noise setting the statistical uncertainty (dominated by finite-shot sampling) is typically at the percent level for the chosen shot budget. We observe a small systematic bias of order a few per mille with the chosen training configuration; this can be reduced by increasing the training data, optimizing hyperparameters, or modifying the ansatz.

10.3.3 Normalized by Construction

One direct application of the primitive-based strategy is to avoid repeated numerical integration inside a fitting loop. For example, valence–sum rules require computing

$$V(x) = \frac{3V(x)}{\displaystyle\int_{x_a}^{x_b} \mathrm{d}x\, V(x)}, \tag{10.11}$$

where the denominator is typically evaluated numerically at every fitting step. With our method, the denominator $I_V = \int_{x_a}^{x_b} V(x)\mathrm{d}x$ is obtained as

$$I_V = G(x_b) - G(x_a), \tag{10.12}$$

so the normalized distribution can be written as

$$V(x) = \frac{V(x)}{I_V}. \tag{10.13}$$

For the ansatz used here, estimating the derivative at a training point requires multiple circuit executions (shifts), but this replaces the need for $\mathcal{O}(10^3)$ integrand evaluations per training step in a typical PDF fit. In the concrete example of Reference [157] this can translate into a net reduction of integrand calls by a factor of order a few, depending on implementation details.

10.3.4 Integrating on a Real Qubit

Finally, we executed a proof-of-principle experiment on a real superconducting qubit hosted at the QRC (TII) [173]. We consider the simple one-dimensional target

$$g(x) = \frac{1}{2}\sin(2x), \qquad x \in [0, 1], \tag{10.14}$$

and evaluate $N_{\mathrm{data}} = 20$ points uniformly distributed in the interval. Each expectation value is obtained on hardware with $N_{\mathrm{shots}} = 5000$ samples, and each point is predicted $N_{\mathrm{runs}} = 5$ times to estimate reproducibility. Figure 10.6 shows the averaged estimates (orange) together with the reference (dashed black). We report the integral estimate obtained from $N_{\mathrm{int}} = 10$ repeated full evaluations as $I_{\mathrm{target}} = 0.326 \pm 0.011$, to be compared with the exact value $I_{\mathrm{target}} = 0.354$. The uncertainty is the sample standard deviation over the N_{int} estimates.

This simple hardware test indicates that systematic errors can be partially mitigated by the endpoint subtraction inherent in the integral reconstruction and that the approach is implementable on current NISQ devices.

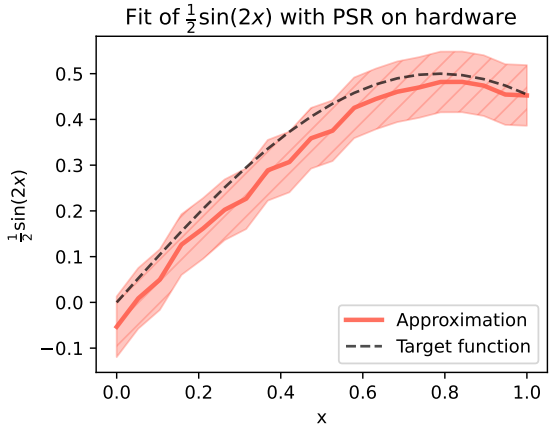


Figure 10.6: Estimates of the integrand $g(x) = \frac{1}{2}\sin(2x)$ in the interval $x \in [0, 1]$, obtained by running the proposed algorithm on a real superconducting qubit. The target function values ($N_{\text{data}} = 20$, black dashed line) are compared with the estimates (orange line), computed as the average of $N_{\text{runs}} = 5$ independent prediction sets. The confidence interval corresponds to 2σ over the experiments. Results are reported without applying any error-mitigation technique. Image taken from [11].

10.4 Can Quantum Computers Help in the Integration Problem?

In this work we have presented a methodology to approximate multidimensional integrals by exploiting the properties of variational quantum circuits. The key element is the possibility of training a circuit derivative to reproduce the integrand, while using the same architecture as a surrogate for the primitive. This allows the definite integral to be reconstructed through boundary evaluations of the circuit. We have illustrated the method both in toy examples and in a realistic case study involving the u-quark parton distribution, as well as in a proof-of-principle hardware demonstration. Across these scenarios, the method shows that once the training is completed, derived quantities such as marginal distributions or parameter–dependent integrals can be obtained at almost no additional computational cost.

The results indicate that, despite the limitations of current NISQ hardware, this approach provides a flexible representation of integrals that can replace costly numerical evaluations in certain contexts. The trade-off between upfront training cost and downstream flexibility resembles that of classical machine-learning surrogates, but the use of quantum circuits may open additional opportunities thanks to properties such as efficient derivative evaluation via the parameter shift rule.

Looking forward, an important perspective is the integration of this methodology within hybrid computational pipelines. For instance, in modern global analyses of parton distribution functions such as NNPDF [190], the numerical integration of large classes of observables is a central and computationally demanding task. Embedding a primitive-based quantum surrogate in such Monte Carlo workflows would allow orchestration between classical sampling strategies and quantum-assisted integral evaluation, with the potential to reduce the number of repeated integrand calls while maintaining precision. More generally, similar hybrid strategies could be developed in other fields where integrals are evaluated repeatedly in parameter-dependent settings, such as lattice QCD,

Bayesian inference, or statistical mechanics.

While further research is needed to assess scalability and robustness under hardware noise, the present work demonstrates a first step towards such integration. We believe that the orchestration of classical and quantum tools, each handling complementary aspects of the calculation, is a natural direction for exploiting quantum circuits in practical applications of numerical integration.

CHAPTER 11

Density Estimation with Adiabatic Quantum Computing

In this thesis, we have often discussed the importance of orchestrating quantum and classical resources within hybrid protocols. In this chapter, we add an extra component to this hybridization: the use of a different quantum computing paradigm, namely adiabatic quantum computing (AQC). We already introduced AQC in Chapter 1 as an alternative to the more common digital quantum computing. Here, we will tackle a specific problem, namely the determination of probability density functions (PDFs), showing how AQC can be leveraged to help solve it within an hybrid protocol. As illustrated in the diagram in Figure 11.1, the protocol involves a classical training of an adiabatic evolution model, which we want to reproduce a cumulative distribution function (CDF) computed from a sampled dataset. The trained evolution is then translated into a quantum circuit, whose output is differentiated to obtain an approximation of the PDF.

We stress that this application is a proof-of-concept of how AQC can be integrated within a hybrid quantum-classical workflow. The addressed dimensionality is limited to one, and the quantum circuits we derive are relatively shallow, making the whole protocol classically simulable. However, our intention is to demonstrate the feasibility of the approach, and to provide a starting point for future investigations on more complex scenarios, paving the way for more intriguing hybrid quantum-classical protocols.

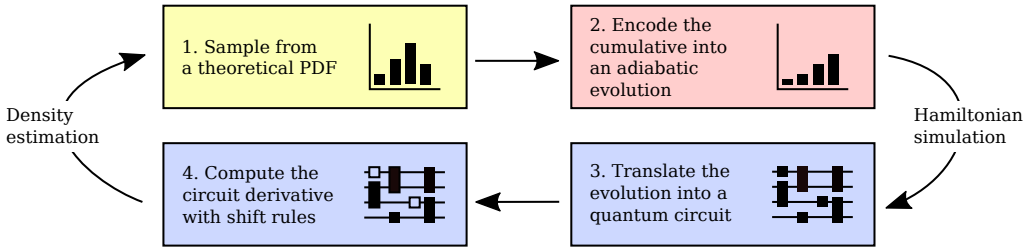


Figure 11.1: Workflow of the proposed hybrid quantum-classical protocol to determine PDFs. A sampled dataset is used to compute a cumulative distribution function (CDF), which is then fed into an adiabatic evolution model. The evolution is trained to reproduce the CDF, and then translated into a quantum circuit. The derivative of the circuit output is finally used to reconstruct an approximation of the PDF.

In the following, we first introduce the practical problem of determining PDFs and the hybrid quantum-classical protocol we propose to solve it. Then, we showcase a series of experiments we performed to validate our approach.

The chapter is based on our published work [12].

The code accompanying this work is available at: https://github.com/qiboteam/adiabatic-fit.

11.1 PDF Estimation with Hybrid Quantum-Classical Models

In this section we describe the hybrid quantum-classical algorithm that we propose for the determination of probability density functions. The method is composed of two main steps. First, we approximate the empirical cumulative distribution function (CDF) of a sample by means of a regression model based on adiabatic quantum evolution. Second, we exploit a circuit representation of the adiabatic Hamiltonian to compute the derivative of the approximated CDF, which provides an estimate of the underlying probability density function (PDF).

In the following, we will refer to this whole protocol as *quantum adiabatic machine learning* (QAML).

11.1.1 Adiabatic Quantum Evolution for Monotonic Regression

Formally, we construct a parametric Hamiltonian of the form

$$H(t) = \big[1 - s(t;\theta)\big]H_0 + s(t;\theta)H_1, \tag{11.1}$$

where H_0 and H_1 are two reference Hamiltonians and $s(t;\theta)$ is a scheduling function depending on a set of parameters θ. The intention is then to use the properties of the time-evolving system to approximate a target function $F(t)$, with $t \in [0, T]$.

This framework is the same as the one we introduced in Appendix C and it is particularly useful in this context, given the nature of the problem we are addressing. As described in Appendix C, the adiabatic theorem states that, if the system is initialized in the ground state of H_0 and the evolution is slow enough, the state of the system at time t will be the ground state of $H(t)$.

The idea here is that we can use the evolving ground state of $H(t)$ to approximate a target function $F(t)$, which in our case is the empirical CDF of a sampled dataset. In particular, we want to find an optimal set of parameters $\boldsymbol{\theta}$ such that the expectation value of a chosen observable O over the evolved state $|\psi(t)\rangle$ approximates the target function,

$$\langle\psi(t;\boldsymbol{\theta})|O|\psi(t;\boldsymbol{\theta})\rangle \simeq F(t), \tag{11.2}$$

where we denote the time-evolved state as $|\psi(t;\boldsymbol{\theta})\rangle$ to emphasize its dependence on the parameters $\boldsymbol{\theta}$ through the scheduling function.

We will denote the left-hand side as $\langle O\rangle_t$ for brevity. The regression problem is therefore reduced to the task of finding an optimal set of parameters $\boldsymbol{\theta}^*$ such that $\langle O\rangle_t$ matches the empirical CDF of the sample.

The CDF is particularly well-suited to this framework because it satisfies two properties that are naturally embodied in the adiabatic evolution: strict monotonicity with respect to t, and boundary conditions fixed at $F(0) = 0$ and $F(T) = 1$. These requirements can be enforced by a suitable choice of H_0, H_1, and O, and by imposing constraints on the scheduling function. In our implementation, we take

$$O = \sigma_z, \quad H_0 = \sigma_x, \quad H_1 = -\sigma_z, \tag{11.3}$$

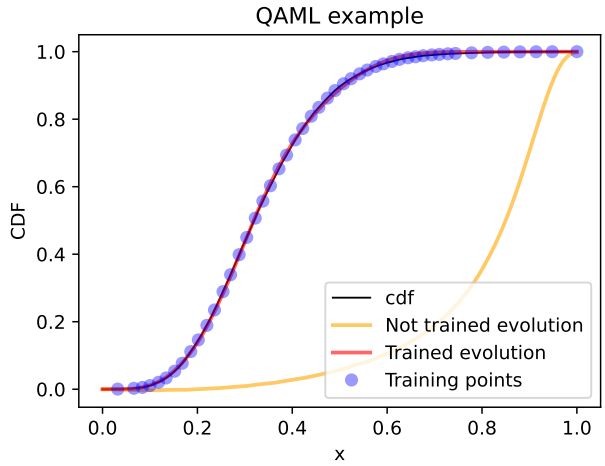


Figure 11.2: Schematic representation of the adiabatic regression model. The system is initialized in the ground state of H_0 and evolved according to the Hamiltonian in Equation (11.1). The expectation value of the observable O over the evolved state is used to approximate the target function $F(t)$. The Hamiltonians framework is chosen such that the energy of our tracking observable is zero in the initial state and one in the final state. An initial untrained schedule is respecting the boundary conditions, but does not approximate the target function (yellow line). After training (red line), the schedule is optimized to reproduce the target function (blue points). Image taken from [12].

which ensures that $\langle O \rangle_0 = 0$ and $\langle O \rangle_T = 1$. As scheduling function we use a polynomial ansatz of degree p:

$$s(t;\theta) = \frac{1}{\eta} \sum_{i=1}^{p} \theta_i t^i, \qquad \eta = \sum_{i=1}^{p} \theta_i, \tag{11.4}$$

whose monotonicity is guaranteed by enforcing $\theta_i > 0$ for all i. This choice is flexible enough to approximate a wide class of empirical CDFs while satisfying the required constraints.

The training of the model follows a supervised strategy. From a dataset $\{x_j\}$ sampled from the target distribution, we construct the empirical CDF $\{F_j\}$. Each pair (x_j, F_j) is mapped to a normalized evolution time $\tau_j = t_j/T \in [0,1]$ and an expectation value $\langle O \rangle_{\tau_j}$. After rescaling the input variable to the unit interval, we define the loss function as

$$C(\theta) = \frac{1}{N_{\text{train}}} \sum_{j=1}^{N_{\text{train}}} \Big(F_j - \langle O \rangle_{\tau_j} \Big)^2, \tag{11.5}$$

which is minimized with respect to θ using a classical optimizer. In our implementation we employed the *covariance matrix adaptation evolution strategy* (CMA-ES) [191], although the method is agnostic with respect to the choice of optimizer. A simple example of how the adiabatic regression model can be trained to approximate a target CDF function is shown in Figure 11.2.

Generalizing the Evolution to Any Time

We now move from the discretized adiabatic evolution to a representation valid for any $t \in [0, T]$. Let $\tau \equiv t/T \in [0, 1]$ and consider the evolution produced by the sequence $\{H(\tau_j)\}$ associated to the discretized schedule $\{\tau_j = j\, d\tau\}$ with step $d\tau$. Each instantaneous Hamiltonian generates a local propagator

$$U(\tau_j) \equiv U_j \;=\; e^{-i\, d\tau\, H(\tau_j)}\,, \tag{11.6}$$

and the state at τ_n is obtained as

$$|\psi(\tau_n)\rangle \;=\; \left(\prod_{j=1}^{n} U_j\right) |\psi(\tau_0)\rangle \;:=\; C(\tau_n)\,|\psi(\tau_0)\rangle\,. \tag{11.7}$$

Write the instantaneous adiabatic Hamiltonian at step j as

$$H_j \;=\; H(\tau_j) \;=\; P_j\, D_j\, P_j^{-1}\,, \qquad D_j \;=\; \mathrm{diag}(\lambda_j,\, -\lambda_j)\,, \tag{11.8}$$

with $\lambda_j = \sqrt{2s_j^2 - 2s_j + 1}$ and $s_j = s(\tau_j; \theta)$. Then

$$C(\tau_n) \;=\; \prod_{j=1}^{n} \left(P_j\, e^{-i\, D_j\, d\tau}\, P_j^{-1}\right). \tag{11.9}$$

For a smoothly varying scheduling function $s(\tau; \theta)$ and sufficiently small $d\tau$ (adiabatic regime), adjacent diagonalizers satisfy $P_j^{-1} P_{j-1} \to I$ with an error proportional to $d\tau$. Under this approximation, the product simplifies to

$$C(\tau_n) \;=\; P_n \exp\!\left(-\,i \sum_{j=0}^{n} D_j\, d\tau\right) P_0^{-1}\,. \tag{11.10}$$

Taking the limit $d\tau \to 0$ (and $n \to \infty$ with $\tau_n = n\, d\tau$ fixed), the sum becomes an integral and we obtain a closed expression for the propagator at arbitrary time t:

$$C_t \;=\; P_t\, \exp\!\left(-\,i \int_0^{t/T} D_\tau\, d\tau\right)\, P_0^{-1}\,, \tag{11.11}$$

where P_t and P_0 denote, respectively, the diagonalization matrices associated with the last and the first evolution operators needed to evolve the ground state of H_0 to time t. Defining

$$I \;\equiv\; \int_0^{\tau} \lambda(t')\, dt'\,, \qquad s \equiv s(\tau), \quad \lambda \equiv \lambda(\tau), \quad \tau = t/T\,, \tag{11.12}$$

the matrix elements in the first row of C_t can be written explicitly as

$$c_{0j} \;=\; \frac{1-s}{s\,\sqrt{\lambda(\lambda - s)}} \left[\cos I \left(1 + (-1)^j\, \frac{\lambda - s}{1-s}\right) + i\, \sin I \left(1 - (-1)^j\, \frac{\lambda - s}{1-s}\right)\right], \qquad j \in \{0, 1\}\,. \tag{11.13}$$

This construction does not depend on the specific functional form of the scheduling $s(\tau; \theta)$; when the integral I admits a closed form, it can be used directly.

Finally, while Equation (11.11) is accurate in the adiabatic regime, one can construct arbitrarily accurate representations of U_t by employing the Magnus expansion of the time-ordered exponential and combining it with high-order Trotter–Suzuki decompositions, thereby controlling the approximation error introduced by finite-step discretization.

Circuit Representation

Any unitary $U \in \mathrm{SU}(2)$ can be written as a product of three rotations thanks to the Euler decomposition. We choose

$$U \;\equiv\; R_z(\phi)\, R_x(\theta)\, R_z(\psi)\,, \tag{11.14}$$

and determine the angles (ϕ, θ, ψ) from the matrix elements of C_t. Writing $C_t = \left(\begin{smallmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{smallmatrix}\right)$, one has

$$\phi \;=\; \frac{\pi}{2} - \arg(c_{01}) - \arg(c_{00})\,, \qquad \theta \;=\; -\,2\,\arccos\big(|c_{00}|\big)\,, \qquad \psi \;=\; \arg(c_{01}) - \frac{\pi}{2} - \arg(c_{00})\,, \tag{11.15}$$

with c_{00} and c_{01} obtained from Equation (11.13) (and the remaining entries fixed by unitarity). This yields a shallow circuit representation of C_t whose parameters depend on the evolution time through $s(\tau)$, $\lambda(\tau)$ and $I(\tau)$, and which is amenable to analytic differentiation in what follows.

We stress that this circuit representation is independent on the choice of the scheduling function, and, for some choices of s, the integral I can be solved analytically.

From the CDF to the PDF

Given the circuit representation of C_t in Equation (11.15), we recover the target function as

$$F(t) \;=\; \langle\psi(0)|\, C_t^\dagger\, O\, C_t\, |\psi(0)\rangle\,. \tag{11.16}$$

When F is the empirical CDF associated with the sample, its one–dimensional probability density is obtained by differentiation with respect to t:

$$\rho(t) \;=\; \frac{dF(t)}{dt} \;=\; \frac{d}{dt}\,\langle\psi(0)|\, C_t^\dagger\, O\, C_t\, |\psi(0)\rangle\,. \tag{11.17}$$

In practice, we evaluate the derivative on quantum hardware using parameter–shift rules (PSR) for rotation–based circuits [72, 73, 74], which express the derivative of the expectation in Equation (11.17) as a linear combination of the same circuit evaluated at shifted angles. This avoids finite–difference instabilities and is robust to shot noise; more general PSR formulas can be employed in case a parameter appears multiple times [192]. The time dependence enters through the angles (ϕ, θ, ψ) computed from the matrix elements $\{c_{0j}\}$ in Equation (11.13).

11.2 Validation Experiments

We now validate the procedure on controlled examples with known target densities and on simulated high–energy physics (HEP) observables. In all tests, the input variable is affinely rescaled to $[0, 1]$ so that $\tau = t/T \in [0, 1]$; the original units are recovered by the inverse transformation. The adiabatic evolution runs from $\tau = 0$ to $\tau = 1$ with step $d\tau = 0.002$, and the scheduling function follows the polynomial ansatz in Equation (11.4). Models are trained until the CDF loss in Equation (11.5) reaches a fixed threshold J_{thresh}.

For quantitative assessment we use the mean squared error (MSE),

$$\mathrm{MSE} \;=\; \frac{1}{N}\sum_{j=1}^{N} \big(y_{j,\mathrm{meas}} - y_{j,\mathrm{pred}}\big)^2, \tag{11.18}$$

computed both for CDF and PDF targets. For the PDF under shot noise we also report the Kullback–Leibler divergence,

$$\mathrm{KL} \;=\; \sum_{j=1}^{N} y_{j,\mathrm{pred}} \;\log\!\left(\frac{y_{j,\mathrm{pred}}}{y_{j,\mathrm{meas}}}\right). \tag{11.19}$$

Unless otherwise stated, for shot–noise simulations each prediction point is obtained as the mean of $N_{\mathrm{runs}} = 20$ independent estimates, each from $N_{\mathrm{shots}} = 2 \cdot 10^5$ circuit evaluations; the associated band corresponds to the sample standard deviation across the N_{runs}.

11.2.1 Sampling Known Distributions

We first draw samples from two families with known densities in order to stress–test both the CDF fit and the PDF reconstruction via differentiation:

- a Gamma distribution,

$$\rho(x;\alpha,\beta) \;=\; \frac{\beta^{\alpha}}{\Gamma(\alpha)}\, x^{\alpha-1}\, e^{-\beta x}, \qquad \alpha = 10,\ \beta = 0.5, \tag{11.20}$$

  with $N_{\mathrm{sample}} = 5 \times 10^4$;

- a mixture of two Gaussians,

$$\rho(x;\boldsymbol{\mu},\boldsymbol{\sigma}) \;=\; 0.6\,\mathcal{N}(x;\mu_1,\sigma_1) \;+\; 0.4\,\mathcal{N}(x;\mu_2,\sigma_2), \qquad \boldsymbol{\mu} = (-10, 5),\ \boldsymbol{\sigma} = (5, 5), \tag{11.21}$$

  with $N_{\mathrm{sample}} = 2 \times 10^5$.

For the Gamma case we use a scheduling polynomial with $p = 25$ parameters; for the Gaussian mixture $p = 30$. Training targets the CDF loss threshold $J_{\mathrm{thresh}} = 10^{-5}$ (see Table 11.1 for the achieved best J_f).

We report both exact statevector simulations and shot–noise simulations. In the shot–noise setting, we consider $N_{\mathrm{shots}} = 2 \cdot 10^5$ as the main configuration and, for illustration, also $2 \cdot 10^4$ to highlight the degradation in precision. For each t we average over $N_{\mathrm{runs}} = 20$ repetitions to obtain the mean estimator and the 1σ band.

Figure 11.3 summarizes the findings. In the top row, the trained CDF tracks the empirical target with MSE at the 10^{-5}–10^{-6} level in both exact and shot–noise simulations. In the bottom row, the PDF reconstructed via parameter–shift maintains few–percent agreement across most of the support in exact simulation; with shot noise, $N_{\mathrm{shots}} = 2{\cdot}10^5$ is sufficient to reach $\mathcal{O}(4\text{–}10\%)$ precision away from the tails, while $2 \cdot 10^4$ shots yield worse than 10% almost everywhere, as visible from the broader yellow bands. Table 11.1 collects the numerical metrics, including the KL divergence for the PDF under shot noise. For these two synthetic cases, PDF MSEs remain in the 10^{-3}–10^{-2} range.

11.2.2 Density Estimation for Simulated HEP Observables

As a more realistic test, we consider the estimation of probability densities for observables in high–energy physics (HEP). Such tasks are relevant in experimental analyses, where the underlying distributions are often unknown and must be inferred from finite samples. We thus consider one–dimensional observables from $pp \to t\bar{t}$ at $\sqrt{s} = 13$ TeV: the rapidity y and the transformed Mandelstam variables $-\log(-t)$ and $-\log s$. In our

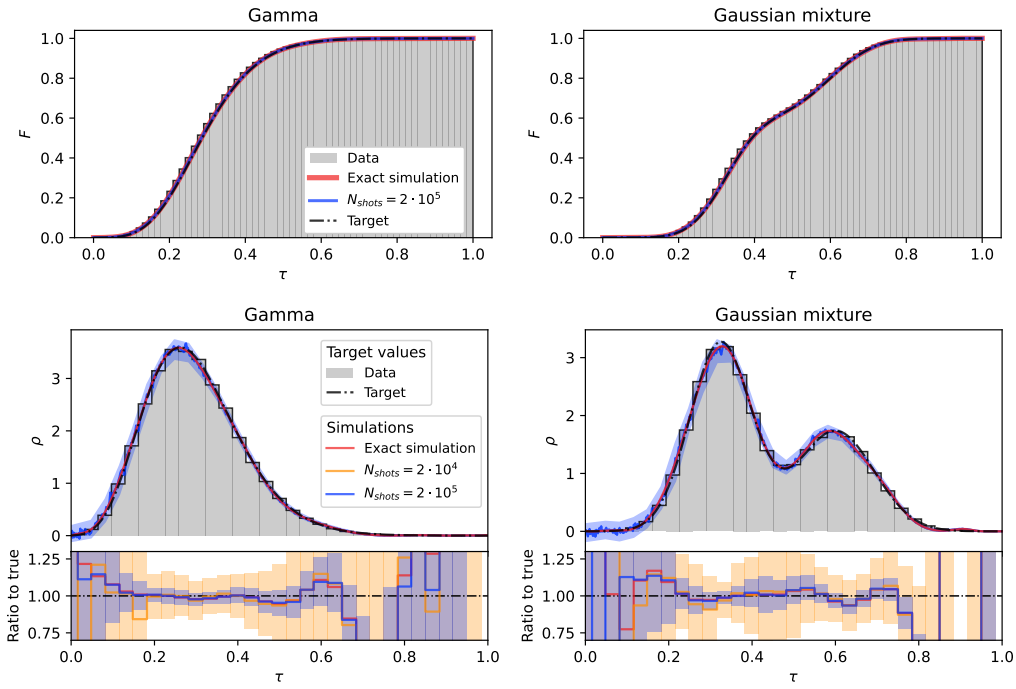


Figure 11.3: Top: CDF fits via QAML for the Gamma distribution (left) and the Gaussian mixture (right). Targets (dashed black) versus QAML exact (red) and shot–noise (blue); data histograms in grey. Bottom: PDFs from differentiating the trained circuit, with the same color code, and ratios to targets. Shot–noise curves shown for $N_{\text{shots}} = 2 \cdot 10^4$ (yellow) and $2 \cdot 10^5$ (blue); in the PDF panels, only $N_{\text{shots}} = 2 \cdot 10^5$ is drawn. Uncertainty bands denote 1σ from $N_{\text{runs}} = 20$ repetitions. Images taken from [12].

tests, the samples are produced by a Style–based quantum GAN (Style–qGAN) [163] trained on 10^5 leading–order events, providing realistic shapes for validation.

We adopt the same training and inference setup as in Section 11.2.1, using $p = 20$ for $-\log(-t)$ and $-\log s$, and $p = 8$ for y (see Table 11.1). Metrics for HEP observables are computed on the bin centers of a histogram with $N_{\text{bins}} = 34$.

Figure 11.4 shows CDF fits (top) and PDFs from the differentiated circuit (bottom). The trained CDFs reproduce the empirical targets with MSE $\sim 3 \cdot 10^{-4}$ under both exact and shot–noise simulations. For the PDFs, exact simulations achieve sub–percent to few–percent agreement around the peaks; with $N_{\text{shots}} = 2 \cdot 10^5$ the precision is $\sim$ 4–10% near the maxima and degrades towards the kinematic tails where densities are small—consistent with classical estimators under sparse statistics. Table 11.1 reports the full set of CDF/PDF MSEs and KL values.

11.2.3 Benchmark Against Kernel Density Estimation

To contextualize our results, we compare the QAML predictions (exact simulation) with the one-dimensional kernel density estimator (KDE) provided by `scikit-learn` [193], using top-hat and exponential kernels.

The kernel bandwidth is selected by a tree-of-Parzen-estimators (TPE) search [194]

Fit function	N_{sample}	p	J_f	$\text{MSE}_{\text{CDF}}^{\text{exact}}$	$\text{MSE}_{\text{CDF}}^{\text{shots}}$	$\text{MSE}_{\text{PDF}}^{\text{exact}}$	$\text{MSE}_{\text{PDF}}^{\text{shots}}$	$\text{KL}_{\text{PDF}}^{\text{shots}}$
Gamma	$5 \cdot 10^4$	25	$2.9 \cdot 10^{-6}$	$1 \cdot 10^{-5}$	$1 \cdot 10^{-5}$	$9 \cdot 10^{-4}$	$2 \cdot 10^{-3}$	$4 \cdot 10^{-3}$
Gaussian mix	$2 \cdot 10^5$	30	$4.4 \cdot 10^{-6}$	$9 \cdot 10^{-6}$	$9 \cdot 10^{-6}$	$2 \cdot 10^{-3}$	$4 \cdot 10^{-3}$	$6 \cdot 10^{-3}$
t	$5 \cdot 10^4$	20	$2.1 \cdot 10^{-6}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$1 \cdot 10^{-3}$	$3 \cdot 10^{-3}$	$5 \cdot 10^{-3}$
s	$5 \cdot 10^4$	20	$7.9 \cdot 10^{-6}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$1 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	$4 \cdot 10^{-3}$
y	$5 \cdot 10^4$	8	$3.7 \cdot 10^{-6}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$9 \cdot 10^{-4}$	$2 \cdot 10^{-3}$	$2 \cdot 10^{-3}$

Table 11.1: Summary of results. Left to right: sample size, number of schedule parameters, best CDF loss J_f, MSE for CDF (exact and shot–noise), MSE for PDF (exact and shot–noise), and KL divergence for the PDF under shot noise. In shot–noise simulations, each MSE entry is computed from the mean of $N_{\text{runs}} = 20$ estimates obtained with $N_{\text{shots}} = 2 \cdot 10^5$. For Gamma and Gaussian mixture we use 500 points uniformly spaced in $[0, 1]$; for HEP observables, metrics are computed on histogram bin centers with $N_{\text{bins}} = 34$.

over a random grid of 10^3 candidates sampled uniformly from $[10^{-5}, 1]$.

Figure 11.5 reports the comparison for the Gamma, Gaussian mixture, and the $-\log(-t)$ observable from the HEP sample. QAML is comparable to KDE baselines in terms of accuracy; for $-\log(-t)$, the differentiated circuit yields a smoother profile under the chosen binning. In terms of computational cost, QAML concentrates effort in the offline optimization of the adiabatic schedule; once trained, PDF evaluation via PSR is lightweight. KDE presents the opposite profile: limited hyperparameter search cost, but a prediction cost that scales with the bandwidth (and with the number of kernels effectively contributing around each query point).

11.2.4 Deployment on Quantum Hardware

We now deploy the trained models on a superconducting quantum processor to assess the performance of the method on real hardware. We use a 5–qubit device hosted at the Quantum Research Centre (QRC) of the Technology Innovation Institute (TII) [173]. Control and execution are performed with `qibolab`, while device characterization and calibration are obtained using `qibocal`. No error–mitigation techniques are applied.

Qubit ID	Assignment fidelity	MSE
0	0.926	$1.3 \cdot 10^{-2}$
1	0.886	$1.4 \cdot 10^{-2}$
2	0.953	$3.4 \cdot 10^{-3}$
3	0.952	$2.7 \cdot 10^{-3}$
4	0.707	$1.3 \cdot 10^{-1}$

Table 11.2: Prediction performance per qubit on hardware. For each qubit we report the assignment fidelity at execution time and the MSE with respect to the target CDF.

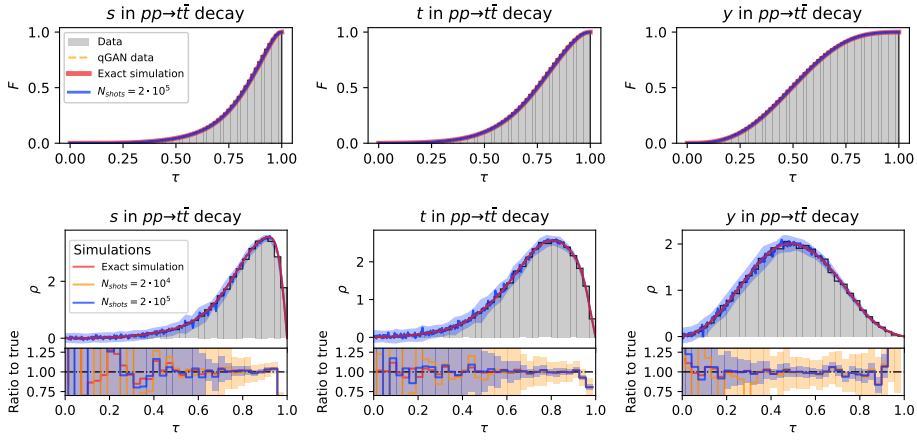


Figure 11.4: Top: CDF fits for the HEP observables $-\log s$ (left), $-\log(-t)$ (center), and y (right), comparing exact (red) and shot–noise (blue) simulations; histograms in grey. Bottom: PDFs from the differentiated circuit, with ratios to the targets. Shot–noise curves for $N_{\text{shots}} = 2 \cdot 10^4$ (yellow) and $2 \cdot 10^5$ (blue); only $N_{\text{shots}} = 2 \cdot 10^5$ is shown in the PDF panels. Bands: 1σ from $N_{\text{runs}} = 20$. Images taken from [12].

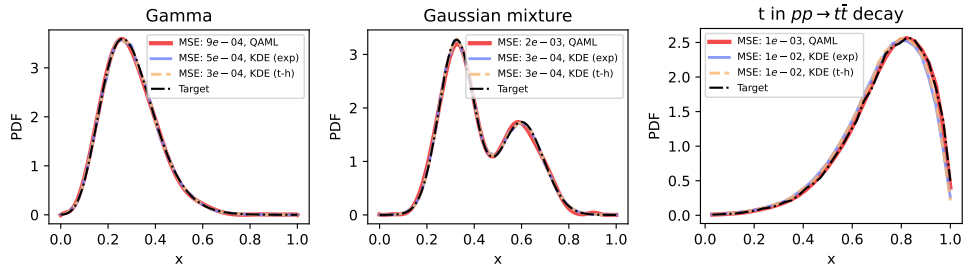


Figure 11.5: Comparison of QAML (red) with kernel density estimation using top–hat (orange) and exponential (blue) kernels; targets in black. Images taken from [12].

Experimental Protocol

We consider the Gamma case of Section 11.2.1 and fix the model parameters to those obtained after training with shot–noise simulation (see Table 11.1). We evaluate the CDF at $N_{\text{data}} = 25$ points uniformly spaced in $[0, 1]$. For each query point, we perform $N_{\text{runs}} = 10$ repetitions on hardware, each consisting of $N_{\text{shots}} = 1000$ circuit evaluations. The final prediction and its uncertainty are obtained as the mean and standard deviation across runs. We compute the MSE with respect to the target CDF for each qubit.

Figure 11.6 shows the CDF predictions obtained using each qubit of the device. Table 11.2 reports the corresponding MSE values together with the qubit assignment fidelities at execution time. We observe a clear dependence of the prediction accuracy on the assignment fidelity: qubits with higher fidelity achieve smaller MSE. The worst performance is associated with the lowest–fidelity qubit.

The hardware results corroborate the simulational study: with unmitigated noise and modest shot budgets, the method retains acceptable accuracy for qubits with good readout quality. The observed dispersion across qubits suggests that improvements in

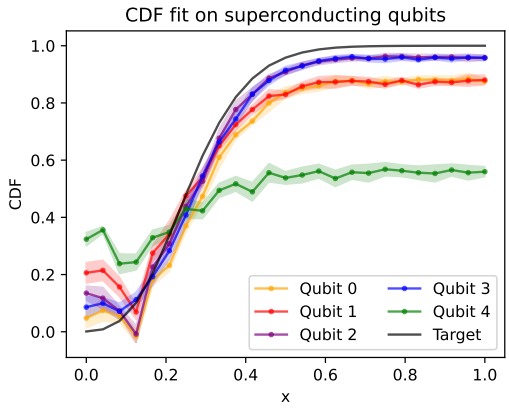


Figure 11.6: $N_{\text{runs}} = 10$ predictions for $N_{\text{data}} = 25$ points in the target range $[0, 1]$ using each qubit of a 5-qubit superconducting device hosted at QRC. The solid lines denote the mean prediction across runs; shaded bands indicate one standard deviation. Image taken from [12].

calibration and error–mitigation, as well as modest increases in shot counts, can further reduce the hardware–simulation gap. Since the circuit depth is low by construction, the dominant limitations here stem from state preparation and readout rather than coherent gate errors.

11.3 A Further Step Towards Quantum-Classical Orchestration

We presented a hybrid quantum-classical protocol for probability density estimation based on adiabatic quantum evolution and circuit differentiation. The method is particularly interesting if we consider how different computational resources are orchestrated to solve the problem at hand. *i)* The adiabatic evolution is used to learn a monotonic function (the CDF) that satisfies fixed boundary conditions. This is a task that is naturally suited to the adiabatic framework, which accommodates monotonicity and allows for flexible scheduling functions. *ii)* The training of the adiabatic model is performed classically, using a classical optimizer to minimize the CDF loss. *iii)* Once the adiabatic model is trained, we project the evolution onto a circuit representation that can be efficiently evaluated on any implementation of a digital quantum computer. At this point, we can exploit well-established parameter-shift rules to compute the derivative of the approximated CDF, which provides an estimate of the underlying PDF.

Before converging to this final protocol, we explored alternative strategies to compute the CDF approximation. One option was to use a more standard quantum machine learning approach (see Chapter 2) based on parameterized quantum circuits (PQC) to learn the CDF directly. Interestingly, we found that such an approach was particularly inappropriate for this task, as the PQC tends to produce oscillatory behaviors in its approximations. It is expected, since we usually encode data into rotational gates, and this is leading to approximation regimes which resonates with Fourier series [195].

If, on the one hand, this is a desirable property for learning tasks, since it allows to define universal approximators, on the other hand, it is not ideal for monotonic functions, which are typically smooth and do not exhibit oscillatory patterns. The adiabatic framework, instead, is naturally suited to learn monotonic functions, as the evolution of

the ground state of a Hamiltonian is inherently smooth and continuous. This makes it a more appropriate choice for approximating CDFs, which are by definition monotonic and smooth.

This simple exercise highlights how very different computational resources can be orchestrated to solve a specific problem instead of being seen as competing paradigms.

To do so, it is more and more important to develop tools and frameworks that allow to seamlessly integrate different quantum and classical resources, especially within an hybrid HPC environment.

CHAPTER 12

Double-Bracket Quantum Algorithms

Ground state preparation is a long-standing problem in quantum computing. Estimating or approximating the ground state of a Hamiltonian is relevant for condensed matter physics, chemistry, and high-energy applications. Quantum phase estimation (QPE) represents the main algorithmic approach in the fault-tolerant regime, providing access to eigenvalues with exponential precision [196, 1]. However, the circuit depth required for QPE makes it unsuitable for near-term devices. Variational methods, most notably the variational quantum eigensolver (VQE) [21], have been proposed as alternatives in the noisy intermediate-scale quantum (NISQ) era (see Section 2.3.1 for an introduction to VQEs). They rely on parametrized circuits optimized with classical routines, and have been widely investigated in quantum chemistry and many-body physics. Yet, VQE faces fundamental challenges such as barren plateaus and the absence of rigorous convergence guarantees [67] as discussed in Section 2.3.2.

Recently, a new class of algorithms has been proposed with the aim of combining rigorous convergence properties with implementability on near-term devices. These are known as double-bracket quantum algorithms (DBQAs) [13] and were originally introduced as diagonalization routines as illustrated in Figure 12.1.

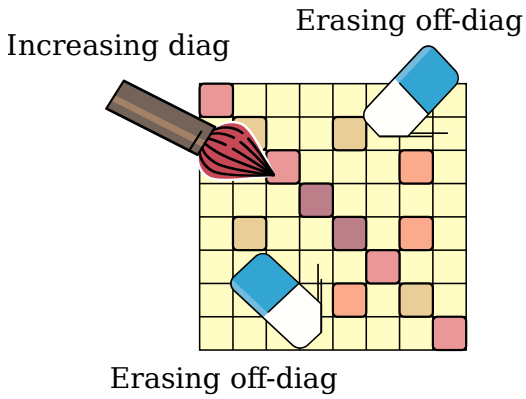


Figure 12.1: Illustration of the effect of double-bracket quantum algorithms on a target Hamiltonian: if some hypotheses are satisfied, the algorithm acts diagonalizing the Hamiltonian [13].

In a second work, we focused on their application to ground state preparation [14], and this will be the focus of the present chapter. In particular, in what follows *i)* we provide an introduction to the DBQA framework, we *ii)* we discuss the computational cost of compiling the protocol into quantum circuits, *iii)* we describe how to interface

DBQAs with approximate routines to construct hybrid algorithms, and *iv)* we present numerical results for a one-dimensional Heisenberg model.

The chapter is based on our published works [14] and [197].

The code accompanying this work is available at: https://github.com/qiboteam/boostvqe.

12.1 Introduction to Double-Bracket Quantum Algorithms

DBQAs are based on the idea of *double-bracket iterations* (DBIs), a recursive scheme originally developed in the context of flow equations for quantum many-body systems [198, 199, 200]. The recursive update of an operator A_k is defined as

$$A_{k+1} = e^{s_k W_k} A_k e^{-s_k W_k}, \tag{12.1}$$

where $W_k = [D_k, A_k]$ is the commutator of A_k with a diagonal operator D_k, and $s_k \in \mathbb{R}$ is a step size. If both A_k and D_k are Hermitian, then the generator $e^{-s_k W_k}$ is unitary and can be implemented as a quantum circuit.

Under appropriate conditions, such as sufficiently small s_k and non-degenerate D_k, the sequence A_k converges exponentially to a diagonal fixed point. This implies that the state obtained by applying the sequence of unitaries to a reference input tends to an eigenstate of the original operator A_0. Since we have convergence guarantees only in the limit of infinite steps, the problem of finding an optimal finite-step approximation arises naturally. In this work, we will see how to address this problem in the context of ground state preparation by interfacing DBQAs with an initial approximate state. This will allow us to construct hybrid protocols relying on a few DBQA steps to refine the initial guess. In this context, DBQAs are suitable candidates for near-term quantum devices. They do not require auxiliary qubits as in QPE, and unlike VQE they are can be used without relying on heuristic optimization. Instead, their convergence is analytically controlled, while leaving space for classical optimization to further enhance performance. In the next sections we show how DBQAs can be interfaced with approximate routines to construct hybrid protocols for ground state preparation.

12.2 Interfacing DBQAs with an Initial Approximation

In practical applications, DBQAs can be advantageously interfaced with an approximate ground state preparation routine. This approach relies on a two-stage protocol: a first stage where a short-depth circuit produces an initial approximation of the ground state, followed by one or more DBQA steps that systematically refine the state [14]. Variational methods such as VQE provide a natural candidate for the warm-start stage, but other strategies, including Hartree-Fock circuits [201], imaginary-time evolution [202], or qubitization-based methods [203], may also be employed [67]. The key observation is that DBQAs do not require a highly accurate initialization, but only a state closer to the ground state than to other excited levels. Once this condition is met, the exponential convergence of DBQAs ensures that the iterated state is driven towards the ground state.

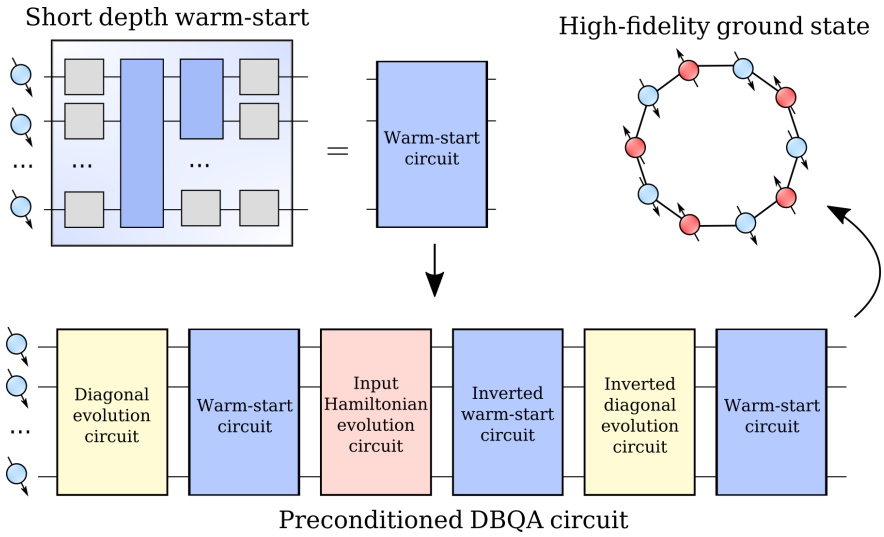


Figure 12.2: Illustration of our hybrid ground state preparation protocol. A first approximation is prepared with a short-depth circuit Q, and then refined with k DBQA steps. In this specific case we have $k = 1$. Figure taken from [14].

Formally, the warm-start mechanism is introduced by setting

$$A_0 = Q^\dagger H_0 Q, \tag{12.2}$$

where Q is the unitary preparing the approximate ground state. The DBQA steps then act recursively on A_0, leading to states

$$|\psi_k\rangle = R_k \cdots R_0 Q|0\rangle, \tag{12.3}$$

with energy expectation values

$$E^{(k)} = \langle\psi_k|H_0|\psi_k\rangle = \langle 0|A_{k+1}|0\rangle, \tag{12.4}$$

which monotonically decrease with k until convergence [13]. This procedure, to which we refer as VQExDBQA when VQE provides the initialization, combines heuristic flexibility at the first stage and analytic convergence guarantees at the second stage.

12.2.1 Compiling DBQAs into Circuits

The central object of DBQAs is the unitary $R_k = e^{-s_k W_k}$ with $W_k = [D_k, A_k]$. To execute such transformations on a quantum device, one must compile them into circuits. A practical approach is the *group-commutator iteration* (GCI) scheme [204], which approximates R_k using a sequence of exponentials of A_k and D_k:

$$V_k = e^{ir_k\varphi D_k} e^{ir_k A_k} e^{-ir_k(\varphi+1)D_k} e^{-ir_k(1-\varphi)A_k} e^{ir_k D_k}, \tag{12.5}$$

where $\varphi = \frac{1}{2}(\sqrt{5}-1)$ and $r_k = \sqrt{s_k}$. This construction satisfies

$$V_k^\dagger A_k V_k \approx R_k^\dagger A_k R_k, \tag{12.6}$$

with an approximation error of order $O(s_k^2)$. The unitary after k steps is then

$$U_k = V_{k-1} \cdots V_1 V_0, \tag{12.7}$$

such that $A_k = U_k^\dagger A_0 U_k$.

Each exponential in V_k corresponds to a Hamiltonian evolution under either A_k or D_k, which can be implemented with standard Hamiltonian simulation techniques, *e.g.* via Trotter–Suzuki decompositions. The recursive structure implies that compiling U_{k+1} requires access to U_k, leading to a depth growth that is exponential in the number of steps. For this reason, in practice one restricts to a small number of DBQA steps (one to three) after the warm-start.

12.2.2 Explicit Cost of Multiple DBQA Steps

The recursive unfolding of GCI circuits can be made explicit. For each step k, the exponential $e^{ir_k A_k}$ is compiled by substituting

$$e^{ir_k A_k} = U_k^\dagger Q^\dagger e^{ir_k H_0} Q U_k, \tag{12.8}$$

which requires a Hamiltonian evolution under H_0. Similarly, evolutions under D_k are compiled from parametrized classical Ising-like operators, requiring at most two layers of CZ gates [14].

This recursive structure explains the scaling of circuit depth. Numerical estimates for the XXZ Heisenberg model (see results section below) show that a warm-start circuit of depth ~ 12 CZ gates per qubit followed by one DBQA step yields a depth of ~ 75, while two steps reach ~ 390, and three steps nearly 2000 CZ gates per qubit [14].

Beyond a few steps, the exponential growth in resources outweighs the gains, making one or two DBQA refinements the practical compromise for near-term devices.

More details about our numerical tests are provided in the following section.

In summary, interfacing DBQAs with approximate routines provides a hybrid ground state preparation protocol. The warm start delivers a sufficiently good initialization, while one or two DBQA steps refine it with analytic convergence. The recursive compilation via GCI ensures implementability, at the cost of an exponential growth in depth with the number of steps, which sets the practical limit of applicability on NISQ hardware.

12.3 Training Procedure and Numerical Simulations

To test the performance of the hybrid protocol, we consider the one-dimensional XXZ Heisenberg model with periodic boundary conditions, defined by the Hamiltonian

$$H_0 = \sum_{i=1}^{L} \left(X_i X_{i+1} + Y_i Y_{i+1} + \Delta Z_i Z_{i+1} \right), \tag{12.9}$$

where X_i, Y_i, Z_i are the Pauli operators acting on qubit i, and Δ is the gap.

In the following, we interface DBQAs with an *Hamming-weight preserving* VQE ansatz to prepare the warm-start state [205, 69]. This choice leverages the conservation of total spin in the XXZ model, allowing us to restrict the ansatz to the relevant symmetry sector. The VQE training is followed by an hyperoptimization of the DBQA parameters, as illustrated in Figure 12.3.

In our work we also tested other ansätze, including hardware-efficient circuits and some different target Hamiltonians, such as a more complicated Heisenberg model with next-to-nearest neighbor interactions. Details about these additional tests can be found in the original publication [14] and will be not be reported here since they lead to similar conclusions to the ones discussed in this sections.

12.3.1 Warm-Start Training with Hamming-Weight Preserving Ansatz

For the warm-start stage we employ a VQE routine with a symmetry-preserving ansatz. In particular, when targeting the XXZ Heisenberg model, we exploit the conservation of total spin. For an even number of qubits L, the ground state lies in the half-filling subspace $S = L/2$, which motivates restricting the ansatz to

$$\dim \mathcal{H}_{\text{sub}} = \binom{L}{L/2}, \tag{12.10}$$

instead of the full Hilbert space of size 2^L. To enforce this restriction, we use a *Hamming-weight preserving* ansatz built from reconfigurable beam-splitter (RBS) gates of the form

$$\text{RBS}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{12.11}$$

A gate of the form of Equation (12.11) acts non-trivially only on the single-excitation subspace spanned by $\{|01\rangle, |10\rangle\}$, while leaving the states $|00\rangle$ and $|11\rangle$ unchanged. Since the number of ones in the bitstring is indeed the Hamming weight, these gates preserve the Hamming weight of computational basis states. In physical terms, once a state is prepared in the half-filling subspace, the ansatz cannot drive it out of that sector. This will ensure that the final approximation will present the expected magnetization. This ansatz is both expressive within the relevant subspace and efficient to train, since it avoids parameter updates outside the symmetry sector of interest [69].

The parameters of the ansatz are optimized with standard classical optimizers, minimizing the variational energy

$$E(\theta) = \langle 0|U_\theta^\dagger H_0 U_\theta|0\rangle, \tag{12.12}$$

until convergence to a sufficiently accurate warm-start state. At this stage, training is limited to relatively shallow circuits (12–20 CZ gates per qubit) to remain within the feasible regime of near-term devices.

12.3.2 DBQA Parameter Optimization

Once the warm-start state is prepared, we initialize the DBQA by setting $A_0 = Q^\dagger H_0 Q$, with $Q = U_{\theta^*}$. For each DBQA step, the diagonal operators D_k are parametrized as classical Ising Hamiltonians. Their coefficients are optimized to minimize the energy expectation value

$$E^{(k)} = \langle 0|A_{k+1}|0\rangle, \tag{12.13}$$

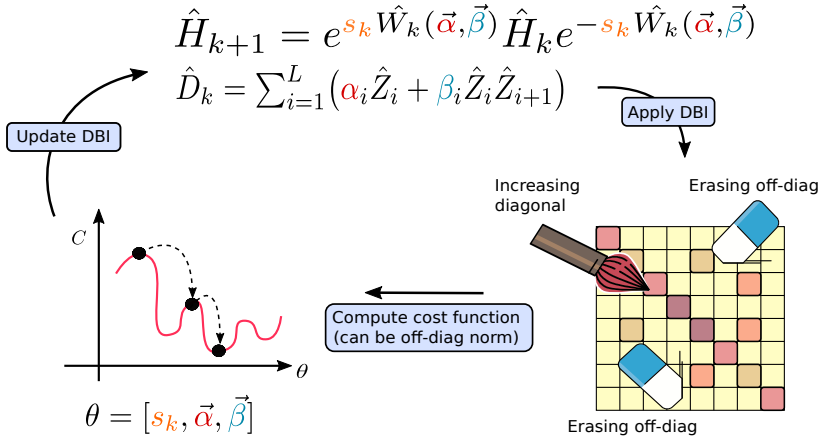


Figure 12.3: Training scheme for the DQBA part of the hybrid protocol. The chosen double-bracket rotation is parametric through its diagonal operator D_k. The parameters of D_k are optimized to minimize the energy expectation value $E^{(k)}$ using a classical optimizer.

using derivative-free methods such as Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [191]. The optimization is carried out sequentially: once step k is optimized, its unitary is fixed and the procedure continues with step $k+1$. An illustration of the chosen parametrization of the diagonal operators and of the optimization procedure is shown in Figure 12.3.

12.3.3 Computational Cost of Training

Since we discuss here a solution for near-term devices, we consider the total number of CZ gates and the circuit depth (defined as number of CZ layers per qubit in a circuit) as the main cost metrics. The computational cost of the protocol has two contributions:

(i) the cost of training the warm-start VQE until a given epoch e;
(ii) the cumulative cost of DBQA's parameters optimization.

The first contribution to the cost is given by the total number of CZ gates executed during the VQE training up to epoch e, denoted $n_{\text{CZ}}^{\text{VQE}}(e)$. In this estimate we include the overhead due to the parameter-shift rule, discussed in Section 2.3.3. We assume that training is performed on hardware, so that the gradient of the energy with respect to the circuit parameters must be estimated via shift-rules. This implies a linear scaling of the cost with the number of parameters. In particular, each parameter requires two local rotations, needed to decompose an RBS gate into native operations on superconducting devices (see Figure 6 of [206]).

The second contribution comes from the repeated evaluations of the energy function during the classical optimization of DBQA. Denoting by n_{fval} the number of function evaluations required for optimizing a single DBQA step, and by $n_{\text{CZ}}^{\text{DBQA}}$ the number of two-qubit gates in the compiled circuit for that step, the total cost after k DBQA steps can be expressed as

$$N_{\text{CZ}}^{\text{tot}}(e,k) = N_{\text{CZ}}^{\text{VQE}}(e) + \sum_{j=1}^{k} n_{\text{fval}}^{(j)} \cdot n_{\text{CZ}}^{(j)}, \tag{12.14}$$

where the first term accounts for the VQE training and the second term for the DBQA iterations.

For the XXZ Heisenberg model with $L = 10$ qubits, numerical simulations show that:

- a warm-start with 3 ansatz layers (~ 12 CZ gates per qubit) refined with one DBQA step yields a circuit depth of ~ 75 CZ gates per qubit;
- two steps increase the depth to ~ 390 CZ gates per qubit;
- three steps approach ~ 2000 CZ gates per qubit.

This exponential scaling in depth limits the number of DBQA refinements that can be realistically applied on near-term devices. In practice, one or two steps are sufficient to obtain order-of-magnitude improvements in energy accuracy, while remaining within feasible computational budgets.

12.4 Validation Experiments

We validate the hybrid protocol on the one-dimensional XXZ Heisenberg model with periodic boundary conditions presented in Equation (12.9) focusing on $L = 10$ and $\Delta = 0.5$. The warm start is prepared with the Hamming-weight preserving ansatz composed of RBS gates as described in Equation (12.11). DBQA steps are compiled with the higher-order group-commutator iteration and optimized by tuning diagonal operators D_k parameterized as classical Ising models. We use short-depth Trotter–Suzuki for e^{-itH_0} and compile e^{-itD_k} with at most two CZ layers. Statistics are reported over fifty random initializations; medians and median absolute deviations are used as point estimates and uncertainties.

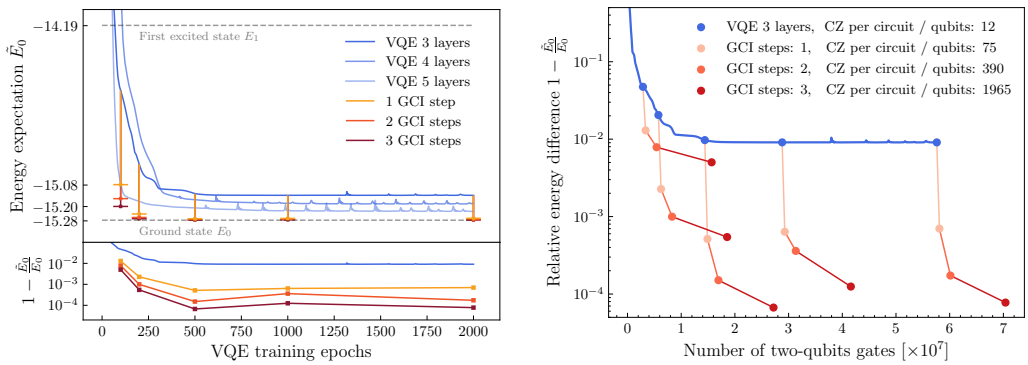


Figure 12.4: Effect of adding DBQA steps on top of a Hamming-weight preserving warm start for XXZ with $L = 10$, $\Delta = 0.5$. Left: VQE training trajectories (3–5 layers) and the improvement obtained by one or more DBQA steps applied at selected epochs. Bottom panel: relative energy error $\Delta E = (\tilde{E}_0 - E_0)/E_0$. Right: cumulative CZ-gate cost versus achieved accuracy, separating VQE training and DBQA optimization. Figure taken from [14].

Table 12.1 summarizes the energy ratios and resource estimates. A single compiled DBQA step (depth 50–100 CZ gates per qubit, here ~ 75) improves the energy error by about one order of magnitude; two steps (depth ~ 390) yield further gains; three steps approach ~ 2000 CZ gates per qubit and are typically unnecessary for the target accuracies. Fidelity lower bounds inferred from energies (Table 12.2) confirm the same trend.

Layers	Warm-start		1 GCI step		2 GCI steps		1 DBI step	Long VQE training	
	$1-\tilde{E}_0/E_0$		$1-\tilde{E}_0/E_0$		$1-\tilde{E}_0/E_0$		$1-\tilde{E}_0/E_0$	$1-\tilde{E}_0/E_0$	
3	0.012 ± 0.004		0.0011 ± 0.0007		0.0005 ± 0.0004		0.0009 ± 0.0006	0.010 ± 0.004	
4	0.008 ± 0.004		0.0006 ± 0.0005		0.0002 ± 0.0002		0.0005 ± 0.0004	0.004 ± 0.002	
5	0.005 ± 0.003		0.0003 ± 0.0002		0.0001 ± 0.0001		0.0002 ± 0.0002	0.003 ± 0.002	
	Depth	Cumulative cost	Depth	Cumulative cost	Depth	Cumulative cost	-	Depth	Cumulative cost
3	12	1.44×10^7	75	1.49×10^7	390	1.69×10^7	-	12	5.76×10^7
4	16	2.56×10^7	95	2.62×10^7	490	2.88×10^7	-	16	10.24×10^7
5	20	4.0×10^7	115	4.07×10^7	590	4.38×10^7	-	20	16.0×10^7

Table 12.1: XXZ with $L = 10$, $\Delta = 0.5$: median energy ratio $\Delta E = 1 - \tilde{E}_0/E_0$ (with MAD over 50 runs) and resources. Top: warm-start VQE, and VQExDBQA with 1–2 compiled GCI steps (and 1 DBI step via dense evolution). Bottom: compiled CZ depth per qubit and cumulative CZ counts accrued during VQE training and DBQA optimization.

Layers	Warm-start	1 GCI step	2 GCI steps	3 GCI steps
3	0.83 ± 0.06	0.95 ± 0.01	0.993 ± 0.006	0.997 ± 0.003
4	0.89 ± 0.05	0.992 ± 0.007	0.997 ± 0.003	0.998 ± 0.001
5	0.93 ± 0.04	0.996 ± 0.003	0.998 ± 0.002	0.9992 ± 0.0008

Table 12.2: Fidelity lower bounds corresponding to Table 12.1, computed from energy estimates using the gap between the ground and first excited states.

12.4.1 Results on Quantum Hardware

We perform a proof-of-concept experiment on an IBM superconducting device for the 10-qubit XXZ model with open boundary conditions. The warm start uses two layers of nearest-neighbour Hamming-weight preserving gates trained in simulation. The diagonal operator includes single-qubit R_Z and next-neighbour R_{ZZ} terms; the evolution is implemented by a single second-order Trotter step. Error mitigation follows a depolarizing-noise renormalization strategy, complemented by Pauli-twirling and readout calibration.

	Warm-start	1 GCI step
CZ depth	4	22
statevector	-14.01	-14.27
raw	-13.14 ± 0.01	-10.60 ± 0.01
mitigated	-13.95 ± 0.01	-14.16 ± 0.04

Table 12.3: Energy on IBM `ibm_fez` for XXZ with $L = 10$ (open boundary). Each point averages 50 twirls $\times$ 1000 shots. Target energy is -14.36.

Overall, one compiled DBQA step improves the VQE warm start by roughly an order of magnitude in relative energy error at depths in the range 50–100 CZ gates per qubit, with further gains from a second step at a few hundred CZ gates per qubit. The IBM test indicates that modest error mitigation is important for realizing these gains on current hardware.

12.5 Orchestration and Outlook

The protocol described in this chapter should be seen as part of a larger workflow, where classical and quantum components are combined in a coordinated way. Rather than being a stand-alone algorithm, DBQA is a module that can be placed in different positions of the pipeline. We can identify three main parts: *i)* the warm start, which produces a first approximation of the ground state and defines the dressed Hamiltonian $A_0 = Q^\dagger H_0 Q$; *ii)* the DBQA core, which applies a small number of double-bracket steps compiled via GCI; emphiii) the compilation and execution layer, which realizes short evolutions under H_0 and diagonal D_k on the chosen hardware. This separation makes the method flexible and easy to interface with other tools.

Warm starts beyond VQE. In our numerical tests the warm start was prepared with VQE, but the protocol does not depend on this choice. Any unitary Q that provides a reasonable approximation can be used. For example, *i)* tensor-network states such as MPS or TTN, prepared classically, can be mapped to circuits and used as Q; *ii)* problem-specific unitaries such as Hartree–Fock or Givens rotations are natural in quantum chemistry; *iii)* short runs of imaginary-time evolution or spectral filtering can also serve as initialization; *iv)* when available, stabilizer or MPO descriptions can be compiled into shallow circuits that preserve the relevant symmetry sector. In all these cases, the DBQA steps that follow refine the initial guess in a controlled way.

Diagonal operators and compilation. The diagonal operators D_k are the adjustable part of each DBQA step. In our tests they were parameterized as classical Ising Hamiltonians, but more structured choices are possible. One can impose symmetries such as translation, reflection, or fixed magnetization, or derive diagonals from approximate classical densities. Compilation of these operators relies on efficient synthesis of diagonal unitaries. Besides quantum compiling strategies, classical logic synthesis methods such as EXORCISM for Boolean functions can also be used to optimize the cost of implementing D_k on hardware.

Scheduling and resource budget. Because the cost of DBQA grows quickly with the number of steps, scheduling is important. A reasonable strategy is: *i)* train the warm start only up to the first plateau in performance; *ii)* add one DBQA step, which already reduces the energy error by about one order of magnitude; *iii)* consider a second step only if the gain per additional two-qubit gate is still favorable. In practice, one or two steps give the best compromise. Stopping rules can be defined in terms of relative energy improvements, fidelity bounds, or a global gate budget.

Interfaces with other protocols. The modular structure of DBQA allows for different ways of integration. On the one hand, it can be placed *after* a classical or variational method: DBQA then refines the approximate state provided by VQE, tensor networks, or stabilizer-MPO methods. On the other hand, it can also be used as a *starting point*. For example, a VQExDBQA state could serve as an input for more precise routines such as quantum phase estimation in the fault-tolerant regime, thus reducing the number of repetitions or the required circuit depth. Similarly, DBQA-refined states can be used to feed classical solvers, for instance to update tensor-network approximations with more accurate correlation data.

In conclusion, DBQA should not be seen as a closed routine, but as a component that can be orchestrated with other classical and quantum methods. Its strength is to provide a convergence step that is both analytically controlled and implementable with near-term resources. This makes it a useful building block in hybrid workflows that combine classical pre-processing, shallow quantum approximations, and more accurate post-processing routines.

Part V

Quantum Systems as Sensors

CHAPTER 13

Towards a Global Search for New Physics with Isotope Shifts

Atomic precision spectroscopy is a precise tool to test the Standard Model (SM) and to look for new effects. Laser cooling, optical frequency combs, optical lattices, and quantum logic spectroscopy let optical clocks reach relative accuracy near 10^{-18}. With this precision, clock comparisons and other differential measurements can probe models of light dark matter and new force mediators.

A central application is *isotope shift (IS) spectroscopy*. Traditionally, IS data constrain nuclear charge radii. More recently, they have been used to test new bosons that couple to electrons and neutrons. Such mediators appear in $B-L$ models and in scalar or vector portals. Isotope shifts are sensitive to mediator masses from the eV to the MeV range. This fills the gap between Casimir-force, beam-dump, and collider searches.

The main analysis tool is the *King plot*. At leading order, isotope shifts in two transitions are related by a straight line. This relation removes the unknown nuclear charge radius difference, which is hard to compute. Deviations from linearity, known as King nonlinearities, can be due to higher-order SM effects or to new physics. Recent *Hz- and sub-Hz-level precision* measurements show such deviations, which makes the analysis richer and also more delicate.

Because many elements and charge states now have precise isotope-shift data, a global approach is needed. New physics should couple in the same way to electrons and neutrons across elements. This motivates a framework that combines data from several elements to obtain a single constraint.

In this chapter we first review the King plot formalism and the algebraic methods used to set bounds on new physics. We then present a fit-based framework, `kifit`, that enables a global analysis of isotope shifts. Finally, we compare algebraic and fit approaches, report current limits, and summarise future prospects.

The discussion follows the work of the Kifit collaboration [15]. My own contribution was the development of the `kifit` framework.

The `kifit` package is open-source and available at https://github.com/QTI-TH/kifit.

13.1 Searching for New Physics with King Plots

An isotope shift is the difference between the transition-i frequency measured in isotope A and in isotope A':

$$\nu_i^{AA'} \equiv \nu_i^A - \nu_i^{A'}. \tag{13.1}$$

At leading order, isotope shifts are described by two terms, each written as a product of an electronic coefficient and a nuclear factor [207, 208, 209, 210]:

$$\nu_i^{AA'} \approx K_i\, \mu^{AA'} + F_i\, \delta\langle r^2\rangle^{AA'}. \tag{13.2}$$

The first contribution is the *mass shift* (MS). It is proportional to the difference of inverse nuclear masses,

$$\mu^{AA'} \equiv \frac{1}{m^A} - \frac{1}{m^{A'}}, \tag{13.3}$$

and describes the nuclear-recoil correction to the electron kinetic energy [211]. The second contribution is the *field shift* (FS), which depends on the change of the nuclear charge radius:

$$\delta\langle r^2\rangle^{AA'} \equiv \langle r^2\rangle^A - \langle r^2\rangle^{A'}. \tag{13.4}$$

Since we mostly deal with isotope pairs, we denote them by a single index $a = AA'$.

13.1.1 Linear King Relation

The factorisation in Equation (13.2) allows us to combine isotope shift measurements of two transitions and eliminate the nuclear charge radius, which is not precisely known. For two transitions labelled 1 and 2, one obtains a linear relation [208, 212]:

$$\tilde{\nu}_2^a = K_{21} + F_{21}\, \tilde{\nu}_1^a, \tag{13.5}$$

where the mass-normalised isotope shifts are defined as

$$\tilde{\nu}_i^a \equiv \frac{\nu_i^a}{\mu^a}, \tag{13.6}$$

and the effective electronic coefficients are

$$F_{21} \equiv \frac{F_2}{F_1}, \qquad K_{21} \equiv K_2 - F_{21} K_1. \tag{13.7}$$

Why King plots are powerful.

- They eliminate the unknown charge-radius differences $\delta\langle r^2\rangle^{AA'}$, which are difficult to measure and even harder to compute reliably.
- They determine the relevant electronic coefficients directly from data, specifically $F_{21} \equiv F_2/F_1$ and $K_{21} \equiv K_2 - F_{21}K_1$—avoiding the need for challenging *ab initio* atomic calculations.

Equation (13.5) is the standard *King relation*. It predicts that isotope shifts of two transitions fall on a straight line, with slope F_{21} and intercept K_{21}. This is illustrated in the left panel of Figure 13.1.

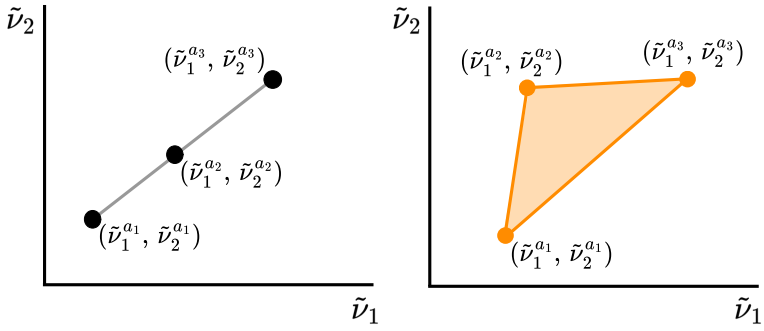


Figure 13.1: Left: Two-dimensional King plot. The isotope shifts follow the linear relation of Equation (13.5). Right: In the presence of extra contributions, the data deviate from the line and span a finite volume. Figure from [15].

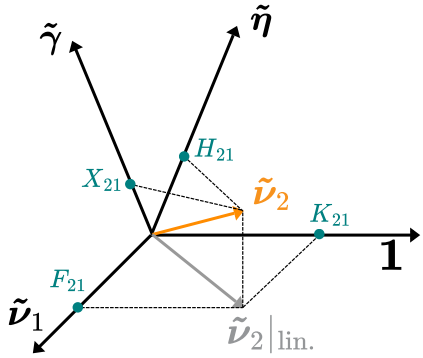


Figure 13.2: Illustration of the King plane, spanned by $\tilde{\boldsymbol{\nu}}_1$ and $\mathbf{1}$. If Equation (13.9) holds, $\tilde{\boldsymbol{\nu}}_2$ lies in this plane. Nonlinearities shift $\tilde{\boldsymbol{\nu}}_2$ out of plane, caused either by new physics or by higher-order Standard Model effects. Figure from [15].

13.1.2 Vector Formulation and the King Plane

If several isotope pairs are available, it is useful to arrange the shifts into vectors in isotope-pair space:

$$\tilde{\boldsymbol{\nu}}_i = \left(\tilde{\nu}_i^1, \ldots, \tilde{\nu}_i^n\right), \qquad \mathbf{1} = (1, \ldots, 1). \tag{13.8}$$

The King relation then takes the form

$$\tilde{\boldsymbol{\nu}}_2\big|_{\text{lin.}} = K_{21}\,\mathbf{1} + F_{21}\,\tilde{\boldsymbol{\nu}}_1. \tag{13.9}$$

Geometrically, $\tilde{\boldsymbol{\nu}}_2$ lies in the plane spanned by $\tilde{\boldsymbol{\nu}}_1$ and $\mathbf{1}$, called the *King plane*. This is shown in Figure 13.2. Deviations from this plane correspond to *King nonlinearities*. Such nonlinearities can come from higher order Standard Model contributions or from new physics effects. An illustration is shown in the right panel of Figure 13.1.

Two equivalent viewpoints. Data can be organized in two ways. *In transition space* (Fig. 13.1, left), each isotope pair a gives a point $(\tilde{\nu}_1^a, \tilde{\nu}_2^a)$ expected to lie on the *King line*;

King nonlinearities appear as *deviations from this line*, forming a small triangular area. *In isotope-pair space* (Fig. 13.2), the data for a given transition form a vector; nonlinearities push $\tilde{\boldsymbol{\nu}}_2$ *out of the King plane* spanned by $\tilde{\boldsymbol{\nu}}_1$ and $\boldsymbol{\mu}$. Both views test deviations from Eq. (13.9).

13.1.3 Algebraic Methods for Linear King Plots

King nonlinearities are any deviations from the linear relation in Equation (13.9). If these are smaller than the experimental uncertainties, the King plot is considered linear. Nonlinearities may be caused by higher-order SM effects or by new physics.

A new boson ϕ with mass m_ϕ that couples to electrons and neutrons induces a Yukawa potential [213, 214][1]:

$$V_{\mathrm{NP}}(r, m_\phi) = -\alpha_{\mathrm{NP}}(A - Z)\frac{e^{-m_\phi r}}{r}\,, \tag{13.10}$$

with $\alpha_{\mathrm{NP}} = (-1)^s \frac{y_e y_n}{4\pi}$. In this expression, s is the spin of the mediator, y_e and y_n are its couplings to electrons and neutrons, Z is the proton number and A the nucleon number.

The predicted shift becomes

$$\nu_i^a|_{\mathrm{pred.}} = K_i \mu^a + F_i \delta\langle r^2\rangle^a + \frac{\alpha_{\mathrm{NP}}}{\alpha_{\mathrm{EM}}} X_i \gamma^a\,, \tag{13.11}$$

where $\gamma^a = A - A'$ is the neutron-number difference, and X_i are electronic coefficients describing the overlap of the Yukawa potential with the electronic wave functions:

$$X_i(m_\phi) = \int \frac{e^{-m_\phi r}}{r}\left[|\Psi_i^{\mathrm{fin}}(r)|^2 - |\Psi_i^{\mathrm{init}}(r)|^2\right] \mathrm{d}r\,. \tag{13.12}$$

They are largest for transitions involving S-states. For their numerical calculation we use `AMBiT` [215].

Combining Equation (13.11) for two transitions gives the modified King relation:

$$\tilde{\boldsymbol{\nu}}_2|_{\mathrm{pred.}} = K_{21}\mathbf{1} + F_{21}\tilde{\boldsymbol{\nu}}_1 + \frac{\alpha_{\mathrm{NP}}}{\alpha_{\mathrm{EM}}} X_{21}\tilde{\boldsymbol{\gamma}}, \tag{13.13}$$

where $X_{21} = X_2 - F_{21}X_1$ and $h^a \equiv \gamma^a/\mu^a$ with $\gamma^a = A - A'$ the neutron-number difference and μ^a the mass factor (see above); we collect these into the vector $\tilde{\boldsymbol{\gamma}}$. Geometrically, this adds an out-of-plane component to $\tilde{\boldsymbol{\nu}}_2$ (see Figure 13.2).

With three isotope pairs ($n = 3$), the new physics coupling can be solved directly [216]:

$$\frac{\alpha_{\mathrm{NP}}}{\alpha_{\mathrm{EM}}} = \frac{\det(\tilde{\boldsymbol{\nu}}_1, \tilde{\boldsymbol{\nu}}_2, \mathbf{1})}{\varepsilon_{ij}\det(X_i\tilde{\boldsymbol{\gamma}}, \tilde{\boldsymbol{\nu}}_j, \mathbf{1})} = \frac{\mathrm{V_{dat}}}{\mathrm{V_{pred}}}\,, \tag{13.14}$$

where ε_{ij} is the two-dimensional Levi-Civita symbol and $\det(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ is the determinand of the square matrix whose columns are the vectors $\boldsymbol{a}, \boldsymbol{b}$ and $\boldsymbol{c}$.

This *Minimal King Plot formula* allows setting bounds on α_{NP}, but cannot by itself distinguish new physics from higher-order SM effects.

The sensitivity of the King plot method to new physics depends on the mediator mass m_ϕ through the coefficients X_i:

[1] The potential is expressed in relativistic units ($c = \hbar = 1$).

- In the *massless limit*, $V_{\rm NP} \propto 1/r$, the X_i become independent of m_ϕ, and King plots reach their maximum sensitivity.
- In the *intermediate region*, X_i vary with m_ϕ, and cancellations may reduce sensitivity for certain masses.
- In the *large-m_ϕ limit*, the potential becomes contact-like. The new contribution aligns with the field shift, and the sensitivity to new physics is lost.

Additional Algebraic Methods

Besides the Minimal King Plot formula, two further approaches are often used: the *No-Mass King Plot* and the *Projection Method*. These methods can be advantageous when nuclear mass uncertainties are large, or when more than three isotope pairs are available.

No-Mass King Plot. The *No-Mass King Plot (NMKP)* eliminates not only the charge radius variance $\delta\langle r^2\rangle$ but also the nuclear masses from Equation (13.11). This requires isotope shift measurements for a third transition $\boldsymbol{\nu}_3$ and gives [217]:

$$\frac{\alpha_{\rm NP}}{\alpha_{\rm EM}} = \frac{2\det(\boldsymbol{\nu}_1, \boldsymbol{\nu}_2, \boldsymbol{\nu}_3)}{\varepsilon_{ijk}\det(X_i\boldsymbol{\gamma}, \boldsymbol{\nu}_j, \boldsymbol{\nu}_k)}\,. \tag{13.15}$$

This formula reduces the impact of nuclear mass uncertainties in the extracted bounds. It is particularly useful if these uncertainties dominate over those of the isotope shifts.

Projection Method. The *Projection Method* [218] applies to isotope shift data for two transitions measured across n isotope pairs. It yields a bound on $\alpha_{\rm NP}$ of the form

$$\left|\frac{\alpha_{\rm NP}}{\alpha_{\rm EM}}\right| = \frac{\mathrm{V}(\mathbf{1}, \tilde{\boldsymbol{\nu}}_1, \tilde{\boldsymbol{\nu}}_2)}{|X_{21}|\,\mathrm{V}(\mathbf{1}, \tilde{\boldsymbol{\nu}}_1, \tilde{\boldsymbol{\gamma}})}, \tag{13.16}$$

where $\tilde{\boldsymbol{\nu}}_i$ are the mass-normalized isotope-shift vectors, $\tilde{\boldsymbol{\gamma}}$ has components $h^a = \gamma^a/\mu^a$, and V denotes the oriented volume spanned by three vectors. For $n = 3$ isotope pairs, this reduces to an expression similar to the Minimal King Plot formula (Equation (13.14)), though it is insensitive to the sign of $\alpha_{\rm NP}$.

13.1.4 Algebraic Methods for Nonlinear King Plots

So far we considered isotope shifts that obey the linear relation of Equation (13.2). Increasing experimental precision has now revealed clear deviations, for example in samarium (Sm) [219, 208, 220], ytterbium (Yb) [221, 222, 223, 224, 225, 226], and calcium (Ca) [227]. In some cases, deviations first observed were later resolved by improved measurements, as in cadmium (Cd) [228, 229].

To describe such cases, one can extend Equation (13.11) by adding next-to-leading Standard Model contributions of the form $H_i\eta^a$:

$$\nu_i^a = K_i\mu^a + F_i\delta\langle r^2\rangle^a + H_i\eta^a + \ldots + \frac{\alpha_{\rm NP}}{\alpha_{\rm EM}}X_i\gamma^a\,, \tag{13.17}$$

where the ellipsis indicates further higher-order effects. In the presence of nonlinearities, new methods are needed. The strategy is to identify the main SM sources of nonlinearity and remove them, either by data-driven approaches or with theoretical input. This preserves the usefulness of King plots as probes of new physics.

The Nonlinearity Decomposition

If at least four isotope pairs are measured, and if a clear hierarchy of nonlinearities is present with nuclear-structure predictions available, the *Nonlinearity Decomposition Plot* [221, 222] helps identify the dominant source of nonlinearity.

The idea is to project the data onto the n basis vectors $(\mathbf{1}, \tilde{\boldsymbol{\nu}}_1, \boldsymbol{\Lambda}^1, \ldots, \boldsymbol{\Lambda}^{n-2})$. The pair $(\mathbf{1}, \tilde{\boldsymbol{\nu}}_1)$ spans the plane of King linearity. The remaining $n-2$ vectors $\{\boldsymbol{\Lambda}^\ell\}_{\ell=1}^{n-2}$ span the space of nonlinearities and can be chosen orthogonal[2]. We expand

$$\tilde{\boldsymbol{\nu}}_i = K_{i1}\mathbf{1} + F_{i1}\tilde{\boldsymbol{\nu}}_1 + \sum_{\ell=1}^{n-2} \lambda_{i1}^{\ell}\boldsymbol{\Lambda}^{\ell}\,. \tag{13.18}$$

For the minimal case of 4 isotope pairs, as in Refs. [221, 222, 226, 227], one can choose the nonlinearity basis vectors as [222]

$$\begin{aligned} \boldsymbol{\Lambda}^+ &\sim \left(\tilde{\nu}_1^3 - \tilde{\nu}_1^2,\, \tilde{\nu}_1^1 - \tilde{\nu}_1^4,\, \tilde{\nu}_1^4 - \tilde{\nu}_1^1,\, \tilde{\nu}_1^2 - \tilde{\nu}_1^3\right), \\ \boldsymbol{\Lambda}^- &\sim \left(\tilde{\nu}_1^4 - \tilde{\nu}_1^2,\, \tilde{\nu}_1^1 - \tilde{\nu}_1^3,\, \tilde{\nu}_1^2 - \tilde{\nu}_1^4,\, \tilde{\nu}_1^3 - \tilde{\nu}_1^1\right), \end{aligned} \tag{13.19}$$

so that

$$\tilde{\boldsymbol{\nu}}_i = K_{i1}\mathbf{1} + F_{i1}\tilde{\boldsymbol{\nu}}_1 + \lambda_{i1}^{+}\boldsymbol{\Lambda}^{+} + \lambda_{i1}^{-}\boldsymbol{\Lambda}^{-}\,. \tag{13.20}$$

The coordinates $(\lambda_{i1}^+, \lambda_{i1}^-)$ are then plotted in the *Nonlinearity Decomposition Plot*. For $i = 2, 3$, each point represents the nonlinearity of the King plot built from $(i, 1)$. If the uncertainty ellipse includes the origin, the corresponding King plot is consistent with linearity. Otherwise, extra terms are required to describe the data.

The method is most informative when a single nonlinearity dominates and factorises into electronic and nuclear parts, *e.g.* $H_i\eta^a$ in Equation (13.17). In that case, all points lie close to a single line in the $(\lambda_{i1}^+, \lambda_{i1}^-)$ plane. The slope is transition-independent:

$$\frac{\lambda^-}{\lambda^+} \equiv \frac{\lambda_{i1}^-}{\lambda_{i1}^+} = \frac{(\tilde{\boldsymbol{\eta}}\cdot\boldsymbol{\Lambda}^-)(\boldsymbol{\Lambda}^+\cdot\boldsymbol{\Lambda}^+) - (\tilde{\boldsymbol{\eta}}\cdot\boldsymbol{\Lambda}^+)(\boldsymbol{\Lambda}^-\cdot\boldsymbol{\Lambda}^+)}{(\tilde{\boldsymbol{\eta}}\cdot\boldsymbol{\Lambda}^+)(\boldsymbol{\Lambda}^-\cdot\boldsymbol{\Lambda}^-) - (\tilde{\boldsymbol{\eta}}\cdot\boldsymbol{\Lambda}^-)(\boldsymbol{\Lambda}^-\cdot\boldsymbol{\Lambda}^+)}\,, \tag{13.21}$$

so comparing slopes with nuclear-structure predictions $\tilde{\boldsymbol{\eta}}$ can reveal the origin of the nonlinearity.

The purple line through the origin in Figure 13.3 shows the prediction for new physics that is proportional to neutron number (see Equation (13.11)). If the uncertainty ellipses overlap with this line, the data are compatible with that hypothesis. Other SM higher-order effects define different lines set by their nuclear structure. If the data do not align with a single line through the origin, more than one source of nonlinearity is likely present, and electronic inputs for each source are required.

Generalisation to n isotope pairs With n isotope pairs, several nonlinearities can be identified in a data-driven way. Define the coordinate vector for transitions $(i, 1)$ as $\boldsymbol{\lambda}_{i1} = (\lambda_{i1}^1, \ldots, \lambda_{i1}^{n-2})^\top$, and the $(n \times (n-2))$ matrix $\boldsymbol{\Omega} = (\boldsymbol{\Lambda}^1, \ldots, \boldsymbol{\Lambda}^{n-2})$ that spans the nonlinearity space. Then $\sum_{\ell=1}^{n-2} \lambda_{i1}^{\ell}\boldsymbol{\Lambda}^{\ell} = \boldsymbol{\Omega}\boldsymbol{\lambda}_{i1}$. Using the Moore-Penrose pseudoinverse gives

$$\frac{\lambda_{i1}^{\ell'}}{\lambda_{i1}^{\ell}} = \frac{\left[(\boldsymbol{\Omega}^\top\boldsymbol{\Omega})^{-1}\boldsymbol{\Omega}^\top\tilde{\boldsymbol{\eta}}\right]^{(\ell')}}{\left[(\boldsymbol{\Omega}^\top\boldsymbol{\Omega})^{-1}\boldsymbol{\Omega}^\top\tilde{\boldsymbol{\eta}}\right]^{(\ell)}}\,, \quad \ell' \neq \ell\,, \tag{13.22}$$

[2]The discussion generalises to non-orthogonal $\boldsymbol{\Lambda}^\ell$. In that case, one must include the projections of higher-order terms, such as $H_i\eta^a$, onto $\mathbf{1}$ and $\tilde{\boldsymbol{\nu}}_1$.

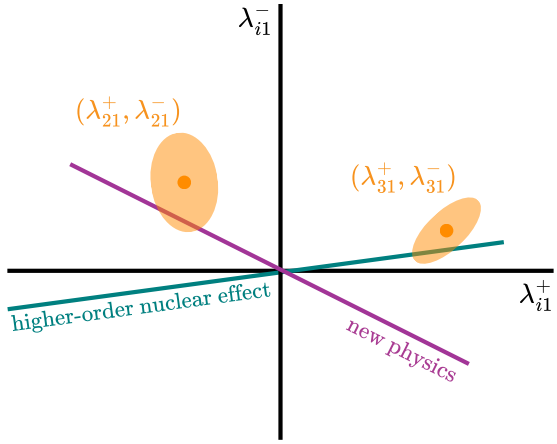


Figure 13.3: Schematic Nonlinearity Decomposition Plot for four isotope pairs. Orange points with uncertainty ellipses show the nonlinearities in the King plots for $(1, 2)$ and $(1, 3)$. The purple line indicates the slope expected from a new-physics term proportional to neutron number. The teal line shows a representative higher-order SM contribution. Figure from [15].

so up to $n - 3$ distinct nonlinearities can be resolved (one for four isotope pairs).

While the slope is transition-independent, the coordinates themselves depend on i and set the magnitude. One finds

$$\|\boldsymbol{\lambda}_{i1}\| = \sqrt{\sum_{\ell} (\lambda^{\ell}_{i1})^2} = |H_{i1}| \sqrt{\tilde{\boldsymbol{\eta}}^{\top} \boldsymbol{\Omega} (\boldsymbol{\Omega}^{\top} \boldsymbol{\Omega})^{-2} \boldsymbol{\Omega}^{\top} \tilde{\boldsymbol{\eta}}}\,, \tag{13.23}$$

or, for four isotope pairs,

$$\begin{aligned} \lambda^+_{i1} &= H_i\, \tilde{\boldsymbol{\eta}} \cdot \frac{\boldsymbol{\Lambda}^+ (\boldsymbol{\Lambda}^- \cdot \boldsymbol{\Lambda}^-) - \boldsymbol{\Lambda}^- (\boldsymbol{\Lambda}^- \cdot \boldsymbol{\Lambda}^+)}{(\boldsymbol{\Lambda}^+ \cdot \boldsymbol{\Lambda}^+)(\boldsymbol{\Lambda}^- \cdot \boldsymbol{\Lambda}^-) - (\boldsymbol{\Lambda}^- \cdot \boldsymbol{\Lambda}^+)^2}\,, \\ \lambda^-_{i1} &= H_i\, \tilde{\boldsymbol{\eta}} \cdot \frac{\boldsymbol{\Lambda}^- (\boldsymbol{\Lambda}^+ \cdot \boldsymbol{\Lambda}^+) - \boldsymbol{\Lambda}^+ (\boldsymbol{\Lambda}^- \cdot \boldsymbol{\Lambda}^+)}{(\boldsymbol{\Lambda}^+ \cdot \boldsymbol{\Lambda}^+)(\boldsymbol{\Lambda}^- \cdot \boldsymbol{\Lambda}^-) - (\boldsymbol{\Lambda}^- \cdot \boldsymbol{\Lambda}^+)^2}\,, \end{aligned} \tag{13.24}$$

which allows extraction of the electronic coefficients H_{i1} from the data.

Generalised King Plots

In recent Yb [221, 222, 223, 224, 225, 226] and Ca [227] analyses, the leading nonlinearity is not compatible with a neutron-coupled new-physics term. In such cases, *Generalised King Plots* (GKP) [217] are used to set bounds on α_{NP}. These extend Equations (13.14) and (13.15) to higher dimensions, using extra transitions to eliminate higher-order nuclear terms in the isotope shift equations.

Assume the isotope shift take the form of Equation (13.17). With two transitions $i = 1, 2$, the charge-radius variance $\delta\langle r^2 \rangle$ can be removed, giving

$$\tilde{\nu}^a_2 = K_{21} + F_{21} \tilde{\nu}^a_1 + H_{21} \tilde{\eta}^a + \ldots + \frac{\alpha_{\mathrm{NP}}}{\alpha_{\mathrm{EM}}} X_{21} \tilde{\gamma}^a\,, \tag{13.25}$$

with $H_{21} \equiv H_2 - F_{21} H_1$. If a third transition $i = 3$ is available, it can remove $\tilde{\eta}^a$:

$$\tilde{\nu}^a_3 = K_{321} + F_{321} \tilde{\nu}^a_1 + H_{321} \tilde{\nu}^a_2 + \ldots + \frac{\alpha_{\mathrm{NP}}}{\alpha_{\mathrm{EM}}} X_{321} \tilde{\gamma}^a\,, \tag{13.26}$$

	KP Equations (13.14)	NMKP (13.15)	GKP (13.27)	NMGKP (13.28)	`kifit` Section 13.2
Isotope pairs	3	3	$3 \leq n$	$3 \leq n$	$3 \leq n$
Transitions	2	3	$n-1$	n	$2 \leq m$
Spurions	-	-	$n-3$	$n-3$	-

Table 13.1: Number of transitions and isotope pairs required by the algebraic methods of Section 13.1 and by the fit framework `kifit` (Section 13.2). For GKP and NMGKP we list the number of eliminated higher-order nuclear parameters (spurions). KP and NMKP are the minimal cases of GKP and NMGKP, respectively.

where $H_{321} \equiv H_{31}/H_{21}$ and $P_{321} = P_{31} - H_{321}P_{21}$ for $P \in \{K, F, X\}$. Bounds can be set if the ellipsis is negligible. Direct extraction of $\alpha_{\rm NP}$ from Equation (13.26) is limited by the theoretical uncertainty on X_{321} (a difference of differences of X_i).

A key advantage of GKP is that it uses the X_i directly. If m transitions are measured across $n = m + 1$ isotope pairs, solving the $m \times n$ system yields [217]

$$\frac{\alpha_{\rm NP}}{\alpha_{\rm EM}} = \frac{(n-2)!\ \det(\tilde{\boldsymbol{\nu}}_1, \ldots, \tilde{\boldsymbol{\nu}}_{n-1}, \mathbf{1})}{\varepsilon_{i_1 \ldots i_{n-1}} \det\left(X_{i_1}\tilde{\boldsymbol{\gamma}}, \tilde{\boldsymbol{\nu}}_{i_2}, \ldots, \tilde{\boldsymbol{\nu}}_{i_{n-1}}, \mathbf{1}\right)} \,. \tag{13.27}$$

If higher-order SM terms are present and nuclear-mass uncertainties dominate, one can combine GKP with the No-Mass KP to obtain a *No-Mass Generalised King Plot* (NMGKP):

$$\frac{\alpha_{\rm NP}}{\alpha_{\rm EM}} = \frac{(n-1)!\ \det(\boldsymbol{\nu}_1, \boldsymbol{\nu}_2, \ldots, \boldsymbol{\nu}_n)}{\varepsilon_{i_1, \ldots, i_n} \det(X_{i_1}\boldsymbol{\gamma}, \boldsymbol{\nu}_{i_2}, \ldots, \boldsymbol{\nu}_{i_n})} \,. \tag{13.28}$$

Subtracting SM Nonlinearities

In lighter systems (*e.g.* Ca), the mass shift can dominate the field shift. The next-to-leading SM term may be the second-order mass shift [230]: $\nu_i^a|_{\rm MS(2)} = K_i^{(2)}\mu^{a(2)}$, with $\mu^{a(2)} \equiv 1/(m^A)^2 - 1/(m^{A'})^2$ and an electronic coefficient $K_i^{(2)}$. Since isotope masses are precisely known, one can keep this term explicit:

$$\tilde{\nu}_2^a = K_{21} + F_{21}\tilde{\nu}_1^a + \frac{\alpha_{\rm NP}}{\alpha_{\rm EM}} X_{21}\tilde{\gamma}^a + K_{21}^{(2)}\tilde{\mu}^{a(2)}. \tag{13.29}$$

With four isotope pairs, Equation (13.29) can be solved for $\alpha_{\rm NP}$:

$$\frac{\alpha_{\rm NP}}{\alpha_{\rm EM}} = \frac{\det\left(\tilde{\boldsymbol{\nu}}_1, \tilde{\boldsymbol{\nu}}_2, \mathbf{1}, \tilde{\boldsymbol{\mu}}^{(\mathbf{2})}\right)}{\varepsilon_{ij} \det\left(X_i\tilde{\boldsymbol{\gamma}}, \tilde{\boldsymbol{\nu}}_j, \mathbf{1}, \tilde{\boldsymbol{\mu}}^{(\mathbf{2})}\right)} \,, \tag{13.30}$$

where all bold symbols are 4-vectors in isotope-pair space. We refer to this as the *Nuclear Input King Plot* (NIKP).

Alternatively, if $K_i^{(2)}$ can be computed with sufficient accuracy (*e.g.* a $\sim 10\%$ uncertainty [227, 230]), one can subtract the second-order mass shift from the data:

$$\left(\tilde{\nu}_2^a - K_{21}^{(2)}\tilde{\mu}^{a(2)}\right) = K_{21} + F_{21}\tilde{\nu}_1^a + \alpha_{\rm NP}X_{21}\tilde{\gamma}^a \,. \tag{13.31}$$

Then Equation (13.14) applies with $\tilde{\nu}_2^a \to (\tilde{\nu}_2^a)' = \tilde{\nu}_2^a - K_{21}^{(2)} \tilde{\mu}^{a(2)}$. Because $K_i^{(2)}$ is common to all isotope pairs, it induces correlations among $\{(\tilde{\nu}_2^a)'\}$, which reduces the impact of its uncertainty on $\sigma[\alpha_{\mathrm{NP}}]$ [227].

This subtraction strategy can bridge the gap between light systems (H, D, He) [231, 232, 233, 234], where theory-experiment comparisons are very clean, and heavy systems (*e.g.* Yb), where nuclear-shape effects (*e.g.* deformation) are sizable yet hard to predict from first principles [226].

Combining Data Sets

The algebraic methods in Section 13.1 (Table 13.1) are simple but rigid, as they apply only to data sets with fixed numbers of isotope pairs and transitions. When data have mixed dimensions, one must use subsets and the optimal way to combine bounds is unclear.

To overcome this, we now introduce a flexible approach based on a fit to isotope-shift data. This allows us to use all available information, even across different elements.

13.2 The King Plot Fit

The King plot fit generalises the idea of a linear King plot to arbitrary numbers of isotope pairs and transitions. It follows the method of Reference [235], but is extended to handle current precision and heterogeneous uncertainties. The proposed method is structured into four main phases (Figure 13.5): build, search, experiment, and consolidation. The algorithm is implemented in the Python package `kifit` [236].

13.2.1 Geometric Construction (Build Phase)

The fit is based on a geometric representation of the isotope shifts in *transition space*. Fixing one transition $i = 1$ as reference, the isotope shifts for an isotope pair a satisfy

$$\tilde{\boldsymbol{\nu}}^a|_{\mathrm{lin.}} = \boldsymbol{\mathcal{K}} + \tilde{\nu}_1^a \boldsymbol{\mathcal{F}} \,, \tag{13.32}$$

where $\boldsymbol{\mathcal{K}}$ and $\boldsymbol{\mathcal{F}}$ are vectors of electronic coefficients. This expresses the fact that the leading-order (linear-model) predictions $\tilde{\boldsymbol{\nu}}^a|_{\mathrm{lin.}}$ lie on a straight line in transition space, called the *King line*. These are not the measured isotope-shift vectors, but their LO expectations given the factorisation in Eq. (13.32).

The orientation of the King line is determined by the electronic field shift coefficients F_{i1}, while the intercept is set by K_{i1}. In practice, only the orientation matters for the fit. The unit vector along the King line is

$$\hat{\boldsymbol{e}}_{\boldsymbol{\mathcal{F}}} = \frac{\boldsymbol{\mathcal{F}}}{\|\boldsymbol{\mathcal{F}}\|} \,. \tag{13.33}$$

Including new physics. A new boson coupling to neutrons and electrons modifies the isotope shifts. For isotope pair a, the induced shift has the form

$$\boldsymbol{\delta}^a = \frac{\alpha_{\mathrm{NP}}}{\alpha_{\mathrm{EM}}} \tilde{\gamma}^a \, \boldsymbol{\mathcal{X}} \,, \tag{13.34}$$

where $\boldsymbol{\mathcal{X}}$ is a vector of electronic coefficients and $\tilde{\gamma}^a$ is the neutron-number difference of the isotope pair. The predicted isotope shifts become

$$\tilde{\boldsymbol{\nu}}^a|_{\mathrm{pred.}} = \tilde{\boldsymbol{\nu}}^a|_{\mathrm{lin.}} + \boldsymbol{\delta}^a \,, \tag{13.35}$$

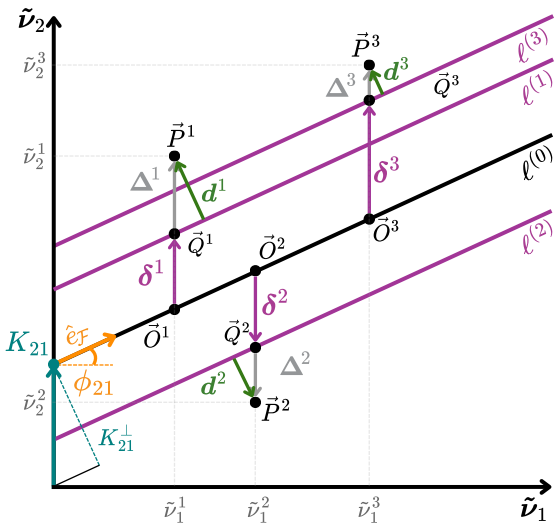


Figure 13.4: Example with two transitions and three isotope pairs. The points $\vec{O}^a$ follow the King line $\ell^{(0)}$ if only SM physics is present. New-physics shifts $\vec{\delta}^a$ displace the points to $\vec{Q}^a$, which can be compared with the measured data $\vec{P}^a$. Figure from [15].

so that each isotope pair defines a line parallel to the King line but displaced by an amount proportional to the coupling α_{NP}.

This construction provides the basis of the fit: one first determines the best-fit King line from the data, and then tests whether the residuals are compatible with a new-physics shift of the form Equation (13.34).

A more detailed description of the build phase is given in Reference [15]. Here we focus on the key aspects of the fit framework.

Construction of the Log-Likelihood

The King plot fit compares the measured isotope-shift vectors $\tilde{\boldsymbol{\nu}}^a$ (data points $\vec{P}^a$ in Figure 13.4) with the theoretical predictions $\tilde{\boldsymbol{\nu}}^a|_{\mathrm{pred.}}$ (points $\vec{Q}^a$). The key quantity is the shortest distance between each data point and the parallel line $\ell^{(a)}$ associated with isotope pair a. This distance is orthogonal to the King line direction $\hat{\boldsymbol{e}}_{\mathcal{F}}$ and quantifies the residual after subtracting the best-fit linear relation.

Collecting these distances into vectors $\boldsymbol{d}^a$, the fit assumes they follow approximately Gaussian statistics. The negative log-likelihood can then be written as

$$-\log \mathcal{L} \propto \tfrac{1}{2}\Big[\log\det \boldsymbol{\Sigma}_{\boldsymbol{d}} + \mathbf{d}^{\top}\boldsymbol{\Sigma}_{\boldsymbol{d}}^{-1}\mathbf{d}\Big], \tag{13.36}$$

where $\mathbf{d} = (\|\boldsymbol{d}^1\|, \ldots, \|\boldsymbol{d}^n\|)^{\top}$ is the vector of norms of the residuals and $\boldsymbol{\Sigma}_{\boldsymbol{d}}$ is their covariance matrix.

In practice, $\boldsymbol{\Sigma}_{\boldsymbol{d}}$ can be estimated in two ways. A rough estimate comes from linear error propagation, using the experimental uncertainties on isotope-shift frequencies and nuclear masses. A more robust option, used in the `kifit` code, is Monte Carlo sam-

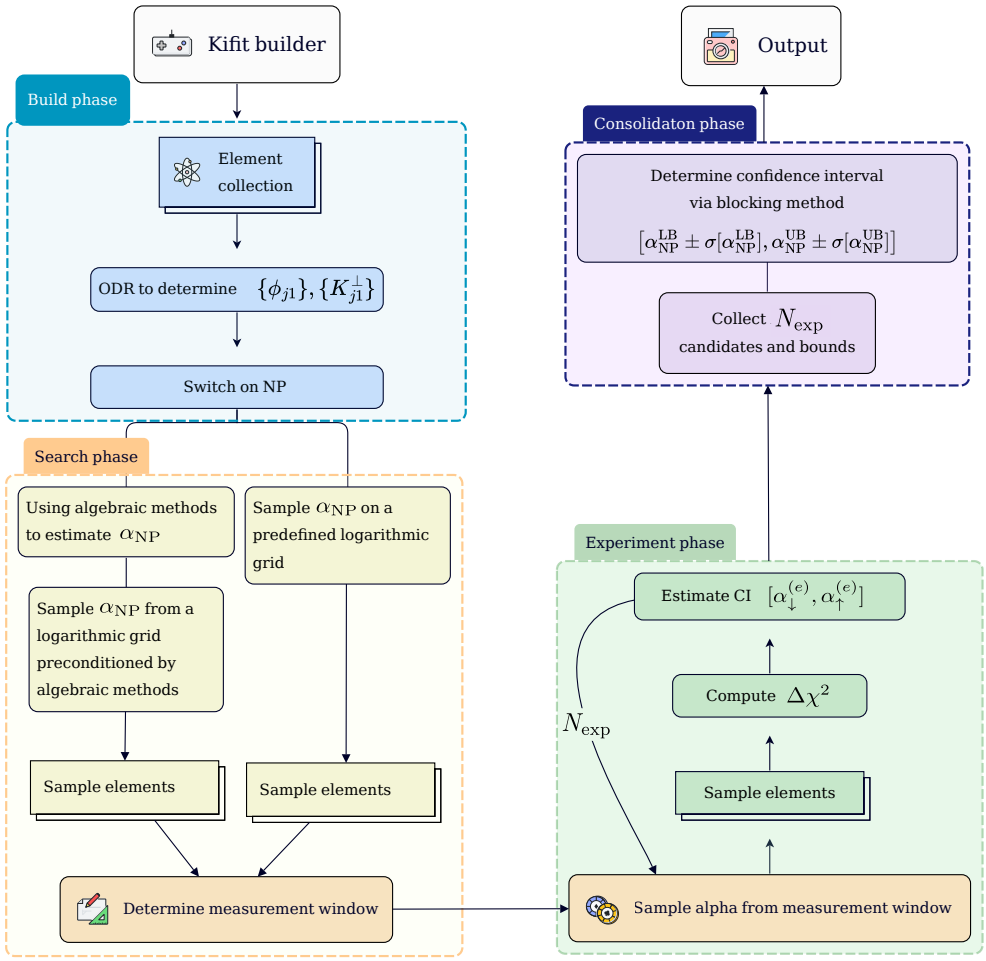


Figure 13.5: Flow of the `kifit` algorithm: build, search, experiment, and consolidation phases. See Section 13.2. Figure from [15].

pling: the measured inputs and the fitted King-line parameters are sampled from their Gaussian distributions, and the resulting spread of residuals is used to construct $\boldsymbol{\Sigma_d}$.

For numerical stability, the covariance matrix is regularised and factorised (*e.g.* via a Cholesky decomposition), so that the log-likelihood can be evaluated efficiently. This procedure guarantees that confidence intervals for α_{NP} are stable and insensitive to rescalings of the residuals.

Finally, if data from multiple chemical elements are analysed, the total likelihood is simply the sum of the contributions from each element:

$$-\log \mathcal{L}(\alpha_{\mathrm{NP}}) = -\sum_{E} \log \mathcal{L}^{(E)}(\alpha_{\mathrm{NP}})\,, \tag{13.37}$$

under the assumption that different data sets are uncorrelated.

For a fixed α_{NP} value and a set of input and fit parameter samples, `kifit` evaluates

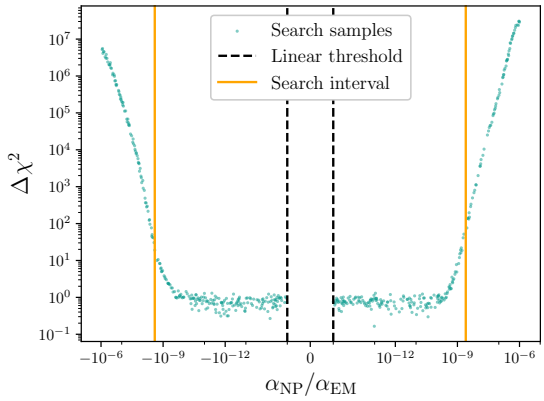


Figure 13.6: *Search phase* example with $N_\alpha = 500$ and $N_{\mathrm{exp}} = 10$. Blue points show $(\alpha_{\mathrm{NP}}, \Delta\chi^2(\alpha_{\mathrm{NP}}))$. The vertical orange lines mark the search window limits. Black dashed lines indicate the region of linear scaling. Figure from [15].

this sum and computes the $\Delta\chi^2$ values:

$$\Delta\chi^2(\alpha_{\mathrm{NP}}) \equiv 2(x - Q[x, p]), \tag{13.38}$$

where $x = -\log \mathcal{L}(\alpha_{\mathrm{NP}})$ and $Q[x, q]$ is the p-th percentile of the negative log-likelihood values of the full set of α_{NP} samples.

The likelihood of Equation (13.37) forms the basis for the subsequent search, experiment, and consolidation phases of the algorithm, illustrated in Figure 13.5. The procedure identifies the region of α_{NP} values compatible with the data, and determines confidence intervals as shown in Figures 13.6 and 13.7.

13.2.2 Determining the Search Window (Search Phase)

The goal of the *search phase* is to find a reasonable range of α_{NP} values that will be used for Monte Carlo sampling in the fit. Two options are implemented:

- `detlogrid` (faster). A logarithmic grid is built around the smallest and largest algebraic estimates of α_{NP} obtained from all valid subsets of the data (using Equations (13.14), (13.15), (13.27), (13.28)). The user parameter `logrid_frac` enlarges this range by a chosen number of decades.

- `globalogrid` (more agnostic). A logarithmic grid is scanned between $\alpha_{\mathrm{NP}} = 10^{-15}$ and an upper edge that grows with m_ϕ.

For each grid point of α_{NP}, the likelihood is evaluated by sampling the experimental inputs and the fitted King-line parameters. The user controls how many α_{NP} points are tested, and how many samples are drawn per point. The *search window* $[\alpha_\downarrow, \alpha_\uparrow]$ is then defined by the smallest and largest α_{NP} whose $\Delta\chi^2$ lies below a liberal threshold, chosen to retain all potentially interesting regions (see Figure 13.6). In practice, both strategies give similar windows, as shown in Figure 13.10.

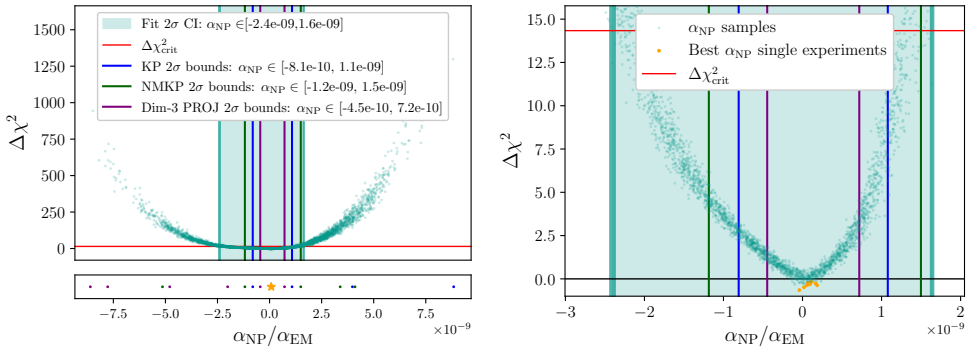


Figure 13.7: *Experiment phase* example with $N_\alpha = 500$ and $N_{\text{exp}} = 10$. Teal points show $(\alpha_{\text{NP}}, \Delta\chi^2(\alpha_{\text{NP}}))$. The red line marks the critical $\Delta\chi^2$. The teal band shows the 2σ confidence interval with uncertainty. Orange stars and bars are best-fit values and their 1σ errors. Figures from [15].

13.2.3 Estimating the Confidence Interval (Experiment Phase)

Given $[\alpha_\downarrow, \alpha_\uparrow]$, `kifit` performs N_{exp} independent *experiments* to determine the confidence interval for α_{NP}:

1. Draw N_α values of α_{NP} from a normal distribution centred at the provisional best value α_* (the one with the lowest negative log-likelihood in the search window) and with a width that covers $[\alpha_\downarrow, \alpha_\uparrow]$.

2. For each sampled α_{NP}, resample the input data and the King-line parameters from their Gaussian uncertainties.

3. Compute $\Delta\chi^2$ for all samples; this yields N_α points in the $(\alpha_{\text{NP}}, \Delta\chi^2)$ plane per experiment.

4. Define the confidence interval of experiment e as $[\alpha^e_\downarrow, \alpha^e_\uparrow]$, the minimal range of sampled α_{NP} with $\Delta\chi^2$ below the critical value for $2m + 1$ degrees of freedom at the chosen $N\sigma$ (default $N = 2$).

5. Record the best-fit point α^e_* of the experiment, *i.e.* the α_{NP} with the lowest negative log-likelihood.

The hyperparameters N_{exp}, N_α, the number of data and parameter samples per α_{NP}, and the chosen N (in $N\sigma$) are user-settable. Figures 13.7 show typical outputs.

13.2.4 Producing Results (Consolidation Phase)

Because the bounds come from Monte Carlo sampling, we need a robust way to quote both the interval and its uncertainty. We use a blocking-style procedure inspired by the *blocking average method* [237]:

- Collect the N_{exp} intervals $[\alpha^e_\downarrow, \alpha^e_\uparrow]$ from the experiments.
- Split them into B blocks of size N_b.

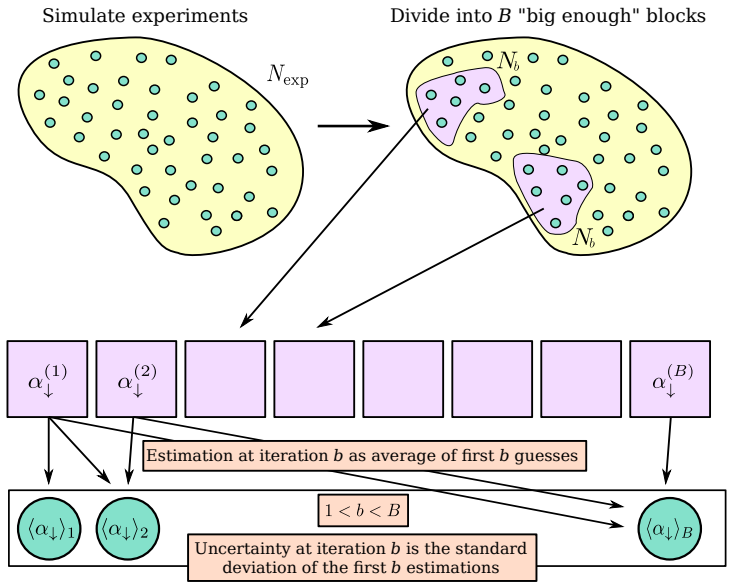


Figure 13.8: Illustration of the blocking method applied to the estimation of the lower bound (for the upper bound, replace $\alpha_\downarrow$ with $\alpha_\uparrow$). The $N_{\rm exp}$ simulated experiments are grouped into blocks labeled $b = 1, \ldots, B$. For each block, the minimum lower bound on $\alpha_{\rm NP}$, denoted $\alpha_\downarrow^{(b)}$, is determined. The set of estimates $\{\alpha_\downarrow^{(b)}\}_{b=1}^{B}$ is then employed to calculate an iterative average, with the final estimate of the lower bound given by $\langle\alpha_\downarrow\rangle_B$. Figure adapted from [15].

- For each block b, define a conservative estimator: use the minimum of the lower bounds and the maximum of the upper bounds in the block, *e.g.*

$$\alpha_\downarrow^{(b)} \equiv \min_{e\in b} \alpha_\downarrow^{e}\,. \tag{13.39}$$

An illustration is shown in Figure 13.8.
From the B block estimators, compute the mean and its sample variance:

$$\langle\alpha_\downarrow\rangle_B = \frac{1}{B}\sum_{b=1}^{B}\alpha_\downarrow^{(b)}\,, \tag{13.40}$$

$$\sigma[\alpha_\downarrow]_B = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}\left(\alpha_\downarrow^{(b)} - \langle\alpha_\downarrow\rangle_B\right)^2}\,. \tag{13.41}$$

Repeat for the upper bound. If the block size is large enough to reduce correlations, both the means and their uncertainties stabilise with B (see Figure 13.9).
We report the final conservative $N\sigma$ interval as

$$[\alpha_{\rm NP}^{\rm LB}, \alpha_{\rm NP}^{\rm UB}] = \left[\langle\alpha_\downarrow\rangle_B - N\,\sigma[\alpha_\downarrow]_B,\ \langle\alpha_\uparrow\rangle_B + N\,\sigma[\alpha_\uparrow]_B\right]. \tag{13.42}$$

We also quote a single *best* $\alpha_{\rm NP}$ value as the median of $\{\alpha_*^e\}$ over experiments, with uncertainty given by their standard deviation (see Figure 13.7).

Repeating the whole procedure for different m_ϕ (thus different $X_i(m_\phi)$) produces exclusion curves (*e.g.* Figure 13.10). At large m_ϕ, the sampling becomes harder and intervals widen; this can be mitigated with more samples, variance reduction, or adaptive sampling.

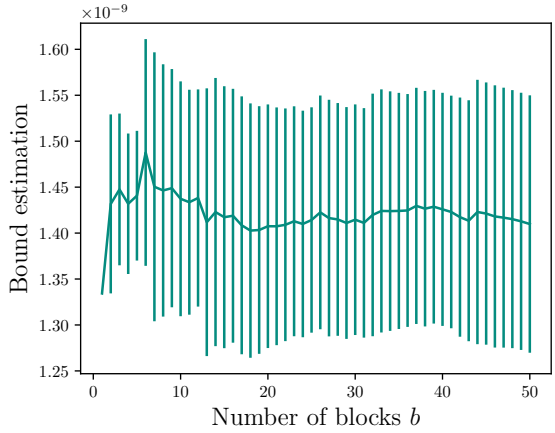


Figure 13.9: Blocking method example for the upper bound after the consolidation phase. Results shown for $N_{\text{exp}} = 500$ split into $B = 50$ blocks. Figure from [15].

13.2.5 Comparison to a Previous King Plot Fit

The geometric construction in `kifit` (Section 13.2.1) is inspired by the fit of Reference [235], but there are key differences in the choice of fit parameters and in the likelihood. Here we summarise the main points.

The fit in Reference [235] assumed similar, uncorrelated uncertainties across transitions and isotope pairs. In our notation, the χ^2 proposed in Reference [235] was

$$\chi^2_a = \sum_{i=1}^{m} \left(\frac{\hat{d}^a_i}{\sigma[\tilde{\nu}^a_i]} \right)^2, \quad \hat{d}^a_i = \tilde{\nu}^a_i - \left(K_{i1} + \frac{\alpha_{\text{NP}}}{\alpha_{\text{EM}}} \tilde{\gamma}^a X_{i1} \right). \tag{13.43}$$

To illustrate the difference with our likelihood, consider the 2D case of Figure 13.4 with angle ϕ_{21}. One finds

$$\hat{d}^a_1 = \|\boldsymbol{d}^a\| \sin\phi_{21}, \quad \hat{d}^a_2 = \|\boldsymbol{d}^a\| \cos\phi_{21}, \tag{13.44}$$

so

$$\chi^2_a = \|\boldsymbol{d}^a\|^2 \left[\left(\frac{\sin\phi_{21}}{\sigma[\tilde{\nu}^a_1]} \right)^2 + \left(\frac{\cos\phi_{21}}{\sigma[\tilde{\nu}^a_2]} \right)^2 \right], \tag{13.45}$$

with $\|\boldsymbol{d}^a\|^2 = \|\boldsymbol{\Delta}^a\| \cos\phi_{21}$.

Equation (13.45) shows a potential bias: normalising separately by $\sigma[\tilde{\nu}^a_i]$ can rotate the fitted King line toward the direction with larger uncertainties (*e.g.* if $\sigma[\tilde{\nu}^a_1] \gg \sigma[\tilde{\nu}^a_2]$), without reducing the overall orthogonal distance to the line. By contrast, `kifit` minimises $\|\boldsymbol{d}^a\|(\Sigma^{ab}_{\boldsymbol{d}})^{-1}\|\boldsymbol{d}^b\|$, treating all directions equally and including correlations.

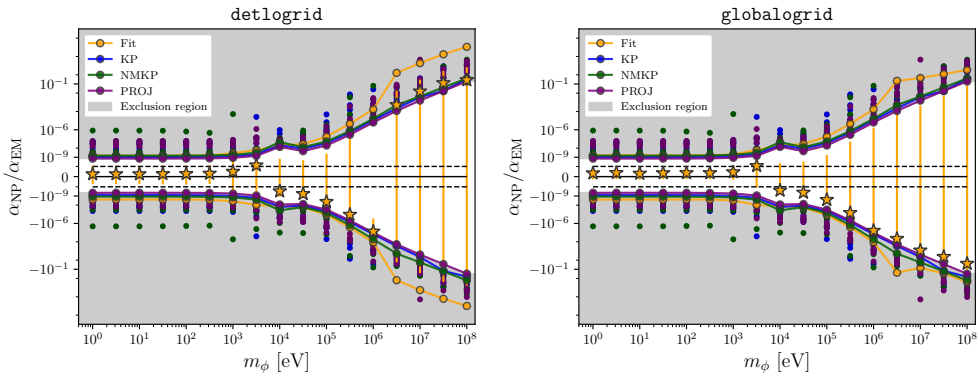


Figure 13.10: `kifit` output for dataset `Ca_WT_Aarhus_PTB_2024`: orange curves show the 2σ bounds on α_{NP}/α_{EM} vs. m_ϕ using the `detlogrid` initialization (left) and `globalogrid` initialization (right). Blue/green/purple markers are algebraic bounds from Equations (13.14), (13.15), (13.16). Grey shading indicates excluded regions. Figure from [15].

Moreover, Eq. (13.45) does not account for constant, new physics-induced shifts of all points that are parallel to the King line direction; such shifts are unidentifiable in a King-plot geometry. In `kifit` we remove them by construction: the likelihood in Eq. (13.36) is built from distances *orthogonal* to $\hat{\mathbf{e}}_F$ (the King-line direction), so any global translation along $\hat{\mathbf{e}}_F$ leaves all residuals $\|\mathbf{d}^{(a)}\|$ unchanged and thus carries no weight in the fit. See Sec. 13.2.1 for the geometric construction.

Further improvements in `kifit` include: uncertainty estimates for the best fit and the interval via a blocking-style consolidation, an extensible framework that combines multiple elements in one fit, and a suite of numerical checks ensuring internal consistency.

13.3 Comparison of Fit and Algebraic Methods

Algebraic methods (Section 13.1) and the `kifit` approach (Section 13.2) both rely on King-plot geometry, but they answer different questions:

Algebraic methods

1. $\langle\alpha_{NP}\rangle$: Which α_{NP} exactly reproduces the central King-plot points for a data set of fixed dimension?
2. $\sigma[\alpha_{NP}]$: How do the experimental uncertainties propagate to α_{NP}?

King plot fit

1. $\{K_{j1}^{\perp}, \phi_{j1}\}$: What is the best linear King line, given the data and their uncertainties?
2. Given the uncertainties on data and fit parameters, what is the likelihood for sampled α_{NP} values?
3. $[\alpha_{NP}^{LB}, \alpha_{NP}^{UB}]$: What is the interval of α_{NP} with $\Delta\chi^2$ below the critical threshold (for $2m+1$ dof at $N\sigma$)?

Algebraic Methods	`kifit`
$\det(\mathcal{N}) \to 0$	$\tilde{\gamma}^a \to \langle\tilde{\gamma}\rangle\ \forall a$
$\det(\mathcal{M}) \to 0$	$X_{j1} \to 0\ \forall j$

Table 13.2: Blind directions in algebraic methods vs. `kifit`. Here $\mathcal{M}$, $\mathcal{N}$ denote electronic and nuclear blocks in the algebraic construction; X_{j1} and $\langle\tilde{\gamma}\rangle$ are the effective electronic and nuclear combinations entering the fit.

13.3.1 Blind Directions

Both approaches inherit blind directions from the King formalism. The method is sensitive only to effects with components orthogonal to both mass shift and field shift. In transition space, this appears as a deviation from the King line (Section 13.2.1); in isotope-pair space, as an out-of-plane component (Figure 13.2). Table 13.2 summarises how these appear algebraically and in `kifit`.

A related feature is the insensitivity to uncertainties *along* the King line. For `kifit` this is explicit (Figure 13.4). For algebraic methods, the impact of uncertainties parallel to the line is negligible in $\sigma[\alpha_{\mathrm{NP}}]$.

Data Set	$(\alpha_{\mathrm{NP}} \pm \sigma[\alpha_{\mathrm{NP}}])\,\vert^{(1)}_{\mathrm{KP}}$	$(\alpha_{\mathrm{NP}} \pm \sigma[\alpha_{\mathrm{NP}}])\,\vert^{\mathrm{MC}}_{\mathrm{KP}}$	$[\alpha^{\mathrm{LB}}_{\mathrm{NP}} \pm \sigma[\alpha_{\downarrow}]_B,\ \alpha^{\mathrm{UB}}_{\mathrm{NP}} \pm \sigma[\alpha_{\uparrow}]_B]$
`Ca3pointTEST`	$(1.36 \pm 4.84) \times 10^{-10}$	$(1.36 \pm 4.81) \times 10^{-10}$	$[-3.51 \pm 0.16, 3.59 \pm 0.13] \times 10^{-11}$
`Ca4pointTEST`	$(-0.14 \pm 4.50) \times 10^{-11}$	$(-0.14 \pm 4.50) \times 10^{-11}$	$[-3.44 \pm 0.15, 3.29 \pm 0.09] \times 10^{-11}$
`Ca10pointTEST`	$(0.19 \pm 1.35) \times 10^{-10}$	$(0.19 \pm 1.31) \times 10^{-10}$	$[0.32 \pm 0.07, 1.65 \pm 0.08] \times 10^{-10}$
`Ca_PTB_2015`	$(0.65 \pm 1.10) \times 10^{-8}$	$(0.65 \pm 1.10) \times 10^{-8}$	$[-7.04 \pm 0.20, 8.04 \pm 0.50] \times 10^{-10}$
`Camin`	$(1.36 \pm 1.12) \times 10^{-9}$	$(1.36 \pm 1.1) \times 10^{-9}$	$[-0.91 \pm 0.03, 1.0 \pm 0.05] \times 10^{-10}$
`Ca24min`	$(1.36 \pm 4.70) \times 10^{-10}$	$(1.36 \pm 4.74) \times 10^{-10}$	$[-2.66 \pm 0.13, 2.55 \pm 0.07] \times 10^{-11}$
`Ca_WT_Aarhus_PTB_2024`	$(1.36 \pm 4.70) \times 10^{-10}$	$(1.36 \pm 4.72) \times 10^{-10}$	$[-1.99 \pm 0.04, 1.45 \pm 0.05] \times 10^{-9}$

Table 13.3: 1σ intervals for α_{NP} at $m_\phi = 1$ eV. Column 1: central value and linear-propagation uncertainty via the minimal algebraic formula. Column 2: Monte Carlo with 2000 samples. Both columns use the subset giving the strongest bound. Column 3: `kifit` blocking-based intervals. Search: `detlogrid` with `logrid_frac`=2, 500 α_{NP} samples, 200 input/fit samples per point. Experiment phase: 150 experiments, each with 1000 α_{NP} samples and 500 input/fit samples per point; block size 25.

13.3.2 Geometric Construction & Form of Data Sets

Algebraic methods compare volumes that combine data and predictions (Figure 13.1), yielding one constraint per fixed-size data set (Table 13.1). The fit instead minimises the orthogonal distances $\|\boldsymbol{d}^a\|$ from data points to the set of King lines $\{\ell^{(a)}\}$ (Figure 13.4), summing over all points in a likelihood (Equation 13.36). It therefore handles general (n, m) with $3 \leq n$ and $2 \leq m$, and can combine multiple, independent data sets. This is ideal for global constraints when new physics couples universally to electrons and neutrons.

Size of Data Sets

Algebraic methods work well for small, fixed-size sets. The fit needs enough diversity to avoid overfitting. With only two points, one can always draw a perfect line and then quote a finite $\sigma[\alpha_{\rm NP}]$, but it would be uninformative: the fit parameters $\{K_{j1}^{\perp}, \phi_{j1}\}$ are fully fixed by two points and the bound on $\alpha_{\rm NP}$ is meaningless. More generally, to fit $2m-1$ free parameters with one scalar constraint per point ($\|\boldsymbol{d}^a\|$), one needs $n \geq 2m-1$ points (see also Table 13.1). Sparse data bias the fit toward overly strong bounds; uncertainties shrink like $1/\sqrt{n}$.

Numerical Comparisons

Table 13.3 compares 1σ intervals at $m_\phi = 1$ eV. We show: *i)* linear-propagation uncertainties via the minimal algebraic formula; *ii)* Monte Carlo estimates with 2000 samples; and *iii)* `kifit` intervals from the blocking consolidation. Linear and MC agree well. `kifit` shows deviations for minimal $(n, m) = (3, 2)$ sets, often giving stronger bounds; agreement improves for larger sets: $(3, 4)$ or $(\geq 4, 2)$.

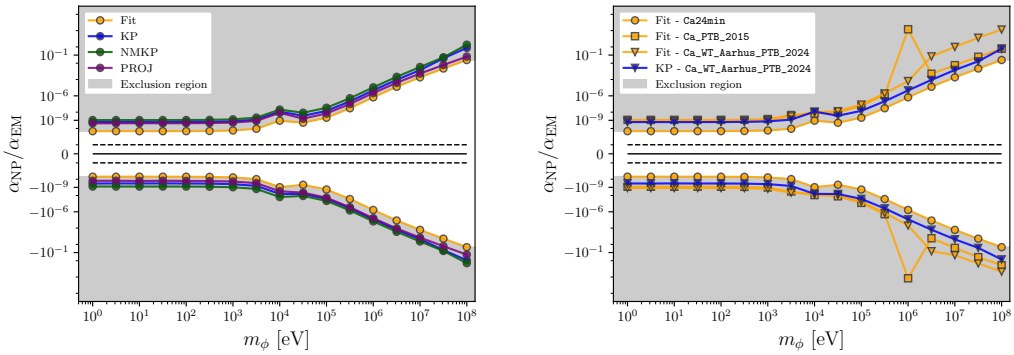


Figure 13.11: Left: minimal dataset `Ca24min` with `kifit` bounds (orange) compared to algebraic and projection methods; grey shading marks excluded regions and black dashed lines indicate where the plot scale is linear. Right: fit results for `Ca24min`, `Ca_PTB_2015` ($(n, m) = (3, 2)$), and their combination `Ca_WT_Aarhus_PTB_2024` ($(n, m) = (3, 4)$). Agreement improves for the larger combined set. Figures from [15].

Figure 13.12 compares Ca and Yb separately and combined. The Yb set includes known SM nonlinearities not modelled in the present version of `kifit`, so its fit result should be interpreted with care. Still, the combined Ca+Yb fit illustrates the flexibility of `kifit` in handling multiple elements.

At large m_ϕ, the fit may appear to lose sensitivity faster than the algebraic envelope. In our tests this traces back to specific subsets (*e.g.* `Ca_PTB_2015`) that degrade earlier; algebraic plots often show the best envelope from multiple subsets, hiding such behaviour. The fit's robustness in this region can be improved with denser m_ϕ scans and richer Monte Carlo sampling.

13.4 Quantum Devices as Sensors

In this chapter we have presented the King-plot formalism as a tool to search for new physics through isotope-shift spectroscopy. We reviewed both algebraic approaches and

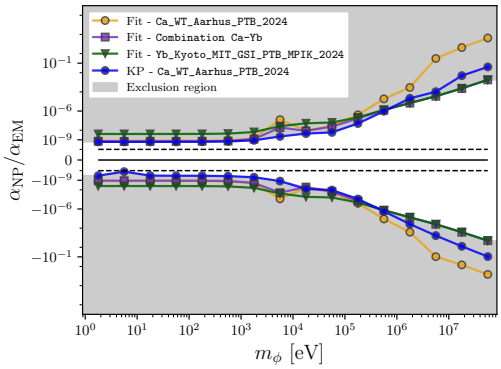


Figure 13.12: Fit results for Ca, for Yb, and for their combination. The most stringent algebraic results for Ca are also shown. Figure from [15].

the fit-based framework implemented in `kifit`, and we discussed their respective advantages and limitations. While algebraic methods provide simple analytic constraints for fixed data sets, the fit offers a more flexible strategy that accommodates larger and heterogeneous collections of measurements. This flexibility is essential for global analyses, in which data from different elements and experiments are combined.

The results presented here highlight the relevance of isotope-shift spectroscopy in extending the sensitivity of laboratory searches for new interactions. In particular, the King plot provides a way to remove dominant nuclear uncertainties and to reveal potential deviations that could be linked to new force mediators. Current and future measurements with precision at the hertz and sub-hertz level will further increase the reach of this method, although they also require a careful treatment of subleading Standard-Model effects.

It is also interesting that these developments place optical clocks and other quantum devices in the role of precision sensors for fundamental physics. The same quantum technologies that were originally designed to stabilise and control atomic transitions now enable competitive tests of new particles and forces. In this sense, isotope-shift spectroscopy illustrates how quantum metrology can act as a bridge between atomic physics and particle physics, adding to the broader landscape of quantum-enhanced sensing. This hybrid character, where devices engineered for timekeeping and information processing also serve as laboratory probes of high-energy physics, represents a particularly fruitful direction for future research.

Conclusions

In this thesis we have investigated hybrid classical-quantum approaches, with a focus on their orchestration within heterogeneous infrastructures. Rather than seeing quantum devices as isolated accelerators, we have framed them as components of a broader ecosystem, where integration, control, and real-time interaction with classical resources are key.

From the middleware perspective, we extended the `Qibo` framework with new modules that reach down to hardware and pulse levels (`Qibolab`, `Qibocal`), enabling self-hosted experiments, calibration routines, and pulse-level integration. These additions turn `Qibo` into a full-stack platform, spanning from algorithm design to hardware control, and make it a natural playground to explore orchestration strategies across all layers of a quantum-classical infrastructure. At higher levels, we developed `Qiboml`, a general-purpose library for hybrid quantum-classical machine learning. Preserving familiar machine learning interfaces, `Qiboml` allows the design of hybrid models and their execution on both simulators and hardware, facilitating experimentation and deployment. The inclusion of calibration libraries alongside machine learning workflows illustrates a central theme of this thesis: the orchestration of components that act at very different layers, yet can be brought together coherently.

Always within the software domain, we introduced `mpstab`, a hybrid stabilizer-tensor network simulator of quantum states based on the theoretical work presented in [176]. This combines the efficiency of stabilizer with the flexibility of tensor networks, enabling the simulation of larger quantum systems. This is an example of how hybridization between methods, even within a single computational resource, can produce practical improvements.

On the algorithmic side, we developed real-time quantum error mitigation methods, initially conceived as standalone techniques and later integrated into `Qiboml`. These methods use classical machine learning to mitigate errors in quantum computations, improving the reliability of NISQ-era results. A key point here is the integration of standalone routines within real-time workflows, making error mitigation directly usable in practical applications. In addition, low-level routines such as calibration can also be accessed through `Qiboml`, showing how hardware-proximal procedures can be embedded into higher-level pipelines like quantum machine learning.

The proposed contributions have been validated through various applications, including multi-variable integration, probability density estimation, and ground-state approximation. These serve as practical demonstrations of how the heterogeneous components of a future quantum-classical infrastructure can be coordinated in meaningful ways.

Figure 13.13: Same characters as in Figure 1, but the orchestra is now placed in an open natural environment. The outdoor setting symbolizes openness to new challenges and opportunities, while highlighting the harmonious interaction among the different elements of the hybrid infrastructure. Image generated with DALL-E 3.

As a complementary exploration, we presented `kifit`, a software package for the analysis of King plot data. While different in scope, this work illustrates how quantum technologies can also serve as measurement tools in precision physics experiments, reinforcing the broader idea that orchestration is not limited to computing tasks alone, but extends to sensing and metrology as well.

Looking ahead, several directions remain open. Hybrid algorithms can be pushed further, exploiting the strengths of both classical and quantum components. Software frameworks like `Qibo` and `Qiboml` should evolve into even more robust and scalable platforms; the goal is to enable the integration of new features in an open-source spirit that encourages collaboration and the exchange of ideas across the scientific community. In addition, extending the scope of these frameworks to encompass emerging technologies and classical tools, such as the integration of large language models, would further expand their relevance and impact.

At the same time, algorithmic strategies for real-time orchestration, including error mitigation, error correction, characterization, calibration, and dynamic resource allocation, still require substantial progress. In particular, tighter integration between control boards and quantum devices could enable more efficient parallelization, a better distribution of tasks between controllers and dedicated hardware (*e.g.*, FPGAs), and, in the long term, new architectures for hybrid execution.

As a concrete example, consider error mitigation. Future work could explore moving part of the mitigation logic directly onto the control hardware. In the case of our proposed RTQEM algorithm, discussed in Chapter 8, computing the mitigation map requires interaction between the classical and quantum nodes, but this step is not needed continuously throughout the entire execution. Instead, the map must be recomputed only when it is detected to be unreliable. As long as the map remains valid, we could implement the logic for applying it, together with sampling and expectation-value evaluation, directly on the control board. Such an approach would reduce communication overhead and latency, potentially leading to a more efficient overall execution.

Distributing workloads across heterogeneous resources remains a key challenge, and one where orchestration can provide the decisive advantage. The results presented in this thesis suggest that hybrid infrastructures should be seen as living systems, where calibration, error management, simulation, and application-level algorithms all coexist and interact. The future of quantum technologies lies in this holistic orchestration: not in isolated components, but in the deliberate integration of tools operating across the full stack.

Appendices

APPENDIX A

Digital Quantum Computing in a Nutshell

In this appendix, we provide a brief introduction to the fundamental concepts of digital quantum computing. First, we introduce the basic notions of qubits, quantum gates, measurements and quantum circuits. After that, we discuss some of the most remarkable quantum algorithms based on the query model, such as the Grover search algorithm.

As the name suggests, digital quantum computing is based on the principles of digital computation, where quantum information is processed using discrete quantum operations, or *gates*, that manipulate information in a step-by-step manner, analogous to classical digital computers.

A.1 Qubits and Quantum States

The building blocks of quantum information storage are *qubits*, which are the quantum analogs of classical bits. A qubit is, in general, a two-level quantum system whose state can be represented as a superposition of two basis states, typically denoted as $|0\rangle$ and $|1\rangle$. Mathematically, a qubit state can be expressed as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1.1}$$

where α and β are complex numbers satisfying the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. The evolution of a quantum state is governed by the Schrödinger equation

$$i\hbar\frac{d}{dt}|\psi(t)\rangle = H|\psi(t)\rangle, \tag{1.2}$$

where H is the Hamiltonian operator of the system, and $\hbar$ is the reduced Planck constant. The solution to this equation describes how the quantum state evolves over time.

In the context of quantum computing, we use unitary transformations to manipulate qubit states. These transformations are represented by unitary matrices that act on the state vector of the qubit. For example, a single qubit gate can be represented as a 2×2 unitary matrix U such that:

$$|\psi'\rangle = U|\psi\rangle \tag{1.3}$$

where $|\psi'\rangle$ is the new state of the qubit after applying gate U. By construction, it is clear that quantum computing implements a completely reversible information processing paradigm. This is a big difference with respect to classical computing, where information can be lost due to irreversible operations.

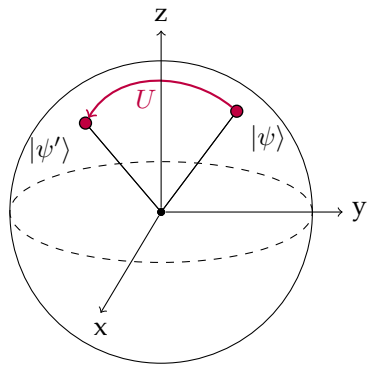


Figure A.1: The Bloch sphere representation of a single-qubit state. Any pure qubit state $|\psi\rangle = \cos(\frac{\theta}{2})\,|0\rangle + e^{i\phi}\sin(\frac{\theta}{2})\,|1\rangle$ is represented as a point on the surface of the unit sphere, determined by the polar angle θ and the azimuthal angle ϕ. A unitary transformation $U \in SU(2)$ corresponds to a rotation of the Bloch vector, mapping $|\psi\rangle$ to another state $|\psi'\rangle$. The axes are defined by the expectation values of the Pauli operators $\sigma_x, \sigma_y, \sigma_z$.

A.2 The Bloch Sphere Representation

A common and graphical way to represent the state of a qubit is the *Bloch sphere*, which is a unit sphere in three-dimensional space. Any pure state of a qubit can be represented as a point on the surface of the sphere, where the north pole corresponds to the state $|0\rangle$ and the south pole corresponds to the state $|1\rangle$. A general state is characterized by two parameters, the angles θ and ϕ, which define its position on the sphere:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle. \tag{1.4}$$

As represented in Figure A.1, the action of a gate on a qubit can be visualized as a rotation of the state vector on the Bloch sphere.

It is intuitive to evince that two points on the antipodal ends of the Bloch sphere are orthogonal, and thus represent two orthogonal states of the qubit.

Also, it is important to note that points on the surface of the Bloch sphere represent pure states, while this tool can be extended to represent mixed states by considering the interior of the sphere. Mixed states are discussed when the density matrix formalism is introduced, namely in Section 3.2.1, since it is necessary to describe realistic and noisy quantum systems.

This representation is useful to visualize how bigger is the expressivity of the state of a qubit compared to a classical bit, which can only be in one of the two states $|0\rangle$ or $|1\rangle$. To give a practical example of how different is the expressive power of a single qubit compared to a classical bit, consider recent quantum machine learning results showing how a single qubit can be used as a universal approximant when tackling learning tasks [165].

A.3 Multiple Qubits

What presented so far considered the state of a single qubit, but it is easily extendable to multiple qubits. The state of a system of n qubits is represented as a vector in a 2^n-dimensional complex Hilbert space, where each basis state is a tensor product of the

individual qubit states. The state of n qubits can be expressed as:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \tag{1.5}$$

where $|i\rangle$ are the computational basis states, and α_i are the complex amplitudes associated with each basis state. The normalization condition for the state of n qubits is:

$$\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1. \tag{1.6}$$

Operations acting on multiple qubits are also described by unitary matrices, now of dimension $2^n \times 2^n$. However, in practice, operations are rarely applied to all qubits simultaneously. Instead, they are usually expressed as a sequence of single-qubit and two-qubit gates. Some of the most common gates will be introduced in the following sections.

According to the standard notation in digital quantum computing, the composition of several gates acting on a set of qubits is referred to as a *quantum circuit*.

A.4 Measurements and Observables

Manipulating the state of qubits using quantum gates we can prepare a state as in Equation 1.5. This state can be in a superposition of multiple basis states, which is a key feature of quantum computing in contrast to classical computing. To extract information from the quantum state we need to perform *measurements*. Measurements are the only non-unitary operations[1] in quantum computing, and they collapse the quantum state into one of the basis states with a probability determined by the *Born rule*.

According to the Born rule, if we measure a state $|\psi\rangle$ in the computational basis, the probability of obtaining the outcome corresponding to the basis state $|i\rangle$ is given by

$$P(i) = |\alpha_i|^2 = |\langle i|\psi\rangle|^2, \tag{1.7}$$

and the state collapses to $|i\rangle$ after the measurement. Once the state is collapsed, any further measurement yields the same result.

A fundamental measurement in quantum computing is the *projection measurement*, which projects the quantum state onto one of the eigenstates of the observable being measured. For a measurement in the computational basis, we use projection operators $P_i = |i\rangle\langle i|$. The probability of measuring outcome i is therefore

$$P(i) = \langle\psi|P_i|\psi\rangle = |\langle i|\psi\rangle|^2, \tag{1.8}$$

which is equivalent to Equation 1.7. After measurement, the state collapses to the corresponding eigenstate:

$$|\psi\rangle \quad \longrightarrow \quad \frac{P_i|\psi\rangle}{\sqrt{P(i)}} = |i\rangle. \tag{1.9}$$

It is often convenient to express these quantities using the *density matrix formalism*, which is discussed in detail in Section 3.2.1. For a pure state $|\psi\rangle$, the density operator is $\rho = |\psi\rangle\langle\psi|$. In this notation, the Born rule can be written equivalently as

$$P(i) = \mathrm{Tr}\big[\rho P_i\big], \tag{1.10}$$

[1] Thus, measurements are the only non-reversible operations in quantum computing.

where Tr denotes the trace operation. More generally, the mathematical framework underlying measurements in quantum computing is based on the concept of *observables*, which are Hermitian operators that correspond to measurable quantities. The expectation value of an observable O with respect to a state is then expressed as

$$\langle O \rangle = \mathrm{Tr}\big[\rho O\big]. \tag{1.11}$$

For Pauli measurements, which are common in quantum computing, we measure observables corresponding to the Pauli matrices X, Y, and Z. Each Pauli operator has eigenvalues ± 1, and the expectation value gives information about the orientation of the qubit state on the Bloch sphere. For instance, measuring in the Z basis tells us about the probability amplitudes of the computational basis states, while measuring in the X or Y bases provides information about superposition and phase relationships.

A.5 Quantum Gates and Circuits

In this section, we introduce the basic operations used to manipulate qubits, which will be useful to understand the algorithms and applications presented in this thesis. Usually, we don't need more than single-qubit gates and two-qubit gates to build any complex quantum operation, as they form a universal set of gates.

Before moving to the description of the building blocks of digital quantum computing, we establish a graphical notation widely used to represent quantum circuits. A system of qubits can be represented as a set of horizontal wires, where each wire represents a qubit. As shown in Figure A.2, operations acting on the qubits are represented as boxes placed along the wires, where the height of the box represents the qubits on which the operation acts. This circuit diagram has to be read from left to right, where the first operation is applied first, and the last operation is applied last, implying the temporal order of the operations.

In the sub-Figure *(d)* of Figure A.2, it is also shown how measurements on the qubits can be represented in the circuit diagram.

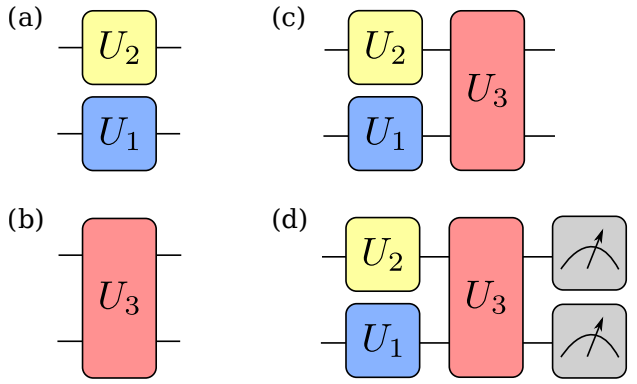


Figure A.2: *(a)* Two single-qubit gates U_1 and U_2 are applied to a system of two qubits. *(b)* A two-qubit gate U_3 is applied to a system of two qubits. *(c)* Combination of the operations in *(a)* and *(b)*. *(d)* The same circuit of *(c)* but measurements are performed on both qubits.

In the following sections, we will introduce the most common gates; for some of them we will also provide a graphical representation in the circuit diagram notation,

while, especially for single-qubit gates, we will implicitly assume that the corresponding representation in the circuit diagram is a box with the name of the gate inside it.

A.5.1 Single Qubit Gates

The most intuitive single qubit gates are the Pauli gates, which are represented by the Pauli matrices:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{1.12}$$

These gates correspond to rotations of π around the x, y, and z axes of the Bloch sphere, respectively. The Pauli gates are often used to flip the state of a qubit or to introduce phase shifts. For example, the Pauli-X gate flips the state of a qubit from $|0\rangle$ to $|1\rangle$ and vice versa, while the Pauli-Z gate introduces a phase shift of π to the state $|1\rangle$.

Another important single qubit gate is the Hadamard gate, which is represented by the matrix:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{1.13}$$

The Hadamard gate creates superposition by transforming the basis states as follows:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \equiv |+\rangle, \quad H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \equiv |-\rangle. \tag{1.14}$$

This gate is crucial for creating superposition states, which is usually an essential preparation phase in many quantum algorithms.

Generalization of the Pauli gates are the *rotation gates*, which rotate the state of a qubit around a specified axis of the Bloch sphere of an arbitrary angle θ. Rotation gates are defined as follows:

$$R_x(\theta) = \exp\left[-i\frac{\theta}{2}X\right] = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix}, \tag{1.15}$$

$$R_y(\theta) = \exp\left[-i\frac{\theta}{2}Y\right] = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}, \tag{1.16}$$

$$R_z(\theta) = \exp\left[-i\frac{\theta}{2}Z\right] = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}, \tag{1.17}$$

where σ_k is one of the Pauli matrices X, Y, or Z. Those gates are especially useful to implement algorithms which require parametric control over the quantum state, as it happens in quantum machine learning. In fact, one can tune the parameter θ to control the rotation angle, thus allowing for fine-grained manipulation of the qubit state. This enables potentially accessing any point of the Bloch sphere, manipulating the operation schematically represented in Figure A.1.

Also, we can always express a general single qubit gate in terms of rotation gates using the Euler decomposition:

$$U = R_z(\phi)R_y(\theta)R_z(\lambda), \tag{1.18}$$

where ϕ, θ, and λ are real parameters.

A.5.2 The T Gate

A particularly important single-qubit gate is the T gate, which is a single qubit gate that introduces a phase shift of $\pi/4$ to the state $|1\rangle$:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}. \tag{1.19}$$

To understand the role of the T gate, we refer the reader to Section 1.2 of Chapter 1.

A.5.3 Two-Qubit Gates

Two-qubit gates are essential for creating entanglement between qubits, which is a key resource in quantum computing. The most common two-qubit gate is the *CNOT gate* (Controlled-NOT gate), which flips the state of the target qubit if the control qubit is in the state $|1\rangle$. The CNOT gate is represented by the matrix:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{1.20}$$

The CNOT gate acts on two qubits, where the first qubit is the control qubit and the second qubit is the target qubit. The action of the CNOT gate on the computational basis states is:

$$\text{CNOT}|00\rangle = |00\rangle, \tag{1.21}$$
$$\text{CNOT}|01\rangle = |01\rangle, \tag{1.22}$$
$$\text{CNOT}|10\rangle = |11\rangle, \tag{1.23}$$
$$\text{CNOT}|11\rangle = |10\rangle. \tag{1.24}$$

This gate is crucial for creating entangled states, which are states presenting non-classical correlations between qubits. A remarkable example of an entangled states are the *Bell states*. One of the Bell states can be prepared applying an Hadamard gate to the first qubit and then a CNOT gate considering the first qubit as control and the second qubit as target:

$$\text{CNOT}_{1,2} H_1 |00\rangle = \text{CNOT}_{1,2} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \equiv |\Phi^+\rangle. \tag{1.25}$$

Other two important two-qubit gates are the *controlled phase gate* (CZ) and the iSWAP gate, which are commonly natively implemented in superconducting qubit platforms. Their matrix representations are:

$$\text{CZ} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \qquad \text{iSWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.26}$$

These gates are particularly useful for creating entangled states and for implementing quantum algorithms that require two-qubit interactions.

The representation of the presented two-qubit gates in the circuit diagram notation is shown in Figure A.3.

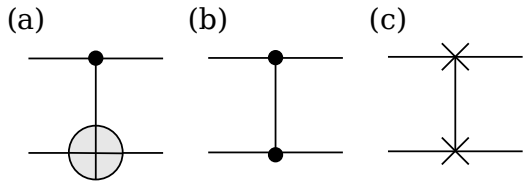


Figure A.3: Representation of the presented two-qubit gates *(a)* CNOT, *(b)* CZ, and *(c)* iSWAP in the circuit diagram notation.

A.6 Multi-Controlled Gates

Multi-controlled gates generalize the concept of controlled gates introduced in the previous section. They allow for the control of a target qubit based on the states of multiple control qubits. The simplest multi-controlled gate is the *Toffoli gate* (CCNOT gate), which is a three-qubit gate that flips the state of the target qubit if both control qubits are in the state $|1\rangle$. The Toffoli gate is represented by the matrix:

$$\text{Toffoli} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{1.27}$$

The Toffoli gate acts on three qubits, where the first two qubits are the control qubits and the third qubit is the target qubit. The importance of the Toffoli gate is inherited from the classical computation, since any reversible classical computation can be implemented using it. In fact, the Toffoli gate can be used to construct any other reversible logic gate, making it a universal gate for reversible computation.

Many quantum algorithms, such as the Grover's search algorithm that we will discuss in the following sections, require the use of multi-controlled gates to implement complex operations on multiple qubits. The ability to control a target qubit based on the states of multiple control qubits allows for the implementation of more sophisticated quantum operations.

APPENDIX B

Query Algorithms

In this appendix, we introduce the concepts at the basis of quantum query algorithms such as the Deutsch-Jozsa algorithm or the Grover's search algorithm, mentioned in Section 1.1.1 of Chapter 1.

After introducing the concept of ancilla qubits and how this can be used to describe the action of a quantum operation on a quantum state in Subsection B.1, we will introduce in Subsection B.2 a simple yet powerful routine, the *phase kickback*, which is a key ingredient in many quantum algorithms. Finally, Subsection B.3 will describe the Grover's search algorithm, which is one of the most famous quantum algorithms.

A practical implementation of these protocols can be found in the `Qiboedu` repository [238], whose material can be used to reproduce the results presented in the following sections. `Qiboedu` is an educational project whose purpose is to provide learning material for students and researchers interested in quantum computing. It is based on the `Qibo` framework [3], and this allows complete access to any stack of the quantum computing ecosystem, from the simulation of quantum circuits to the execution on real quantum hardware. The official mascotte of `Qiboedu`, Qibo the mangoose, is shown in Figure B.1 while moving its first steps with the `Qibo` API.

Figure B.1: The `Qiboedu` mascotte, a mangoose called Qibo, while is coding its first quantum circuits. Image generated using DALL-E 2 and extracted from the `Qiboedu` repository.

B.1 Ancilla Qubits

In quantum computing, *ancilla qubits* are auxiliary qubits used to facilitate quantum operations or to store intermediate results. They are not part of the main computational qubits but are essential for implementing certain quantum algorithms. Ancilla qubits can be used to perform operations that require additional resources, such as error correction, state preparation, or measurements that not demolish the main quantum state. Thus, the whole state involved in a quantum computation can be usually represented as a tensor product of the state of n computational qubits $|x\rangle$ (also called *input register*) and the state of the m ancilla qubits $|y\rangle$ (also called *output register*).

For simplicity, when dealing with many qubits, we will refer to the tensor product of identical states with the superscript $\otimes n$. For example, when all the n qubits are prepared as $|0\rangle$, the state of the system will be $|0\rangle^{\otimes n}$. We use the same notation when we refer to an operation acting on an n-qubit state.

Standard quantum computing protocols consist in embedding a function $f(x)$, acting on the input register, into the state of the output register. This operation, that we define U_f can be represented as:

$$U_f \,|x\rangle\,|y\rangle = |x\rangle\,|y \oplus f(x)\rangle\,, \tag{2.1}$$

where $\oplus$ denotes the bitwise XOR operation. The function $f(x)$ is usually defined as a Boolean function, which maps the input register to a binary output. Because of the XOR properties, if $|y\rangle$ is prepared in the state $|0\rangle$, then Equation 2.1 can be simplified to

$$U_f \,|x\rangle\,|0\rangle^{\otimes m} = |x\rangle\,|f(x)\rangle\,, \tag{2.2}$$

ending up with the output register in the state $|f(x)\rangle$.

Equation 2.2 can be combined with the generalization of the Hadamard gate to n qubits,

$$H^{\otimes n}\,|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle\,, \tag{2.3}$$

to prepare a superposition of all possible inputs to the function $f(x)$:

$$U_f\big(H^{\otimes n}\,|0\rangle^{\otimes n}\,|0\rangle^{\otimes m}\big) = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle\,|f(x)\rangle\,. \tag{2.4}$$

This state can then be used to apply the function $f(x)$ to all possible inputs in parallel, which is a key feature of quantum query algorithms known as *quantum parallelism*.

B.2 The Phase Kickback

Let's consider a system of two qubits per simplicity ($n = m = 1$). Let's also assume that the first qubit (input register) is prepared in the state $|0\rangle$, while the second qubit (output register) is prepared in the state $|1\rangle$. The state of the system can be written as:

$$|\psi\rangle = X_2\,|0\rangle\,|0\rangle = |0\rangle\,|1\rangle\,. \tag{2.5}$$

In Figure B.2 we show the probabilities of measuring the individual components of the state $|\psi\rangle$ in the computational basis.

As mentioned in the previous section, we start by applying an Hadamard gate to the first qubit, which prepares it into the superposed state $|+\rangle$:

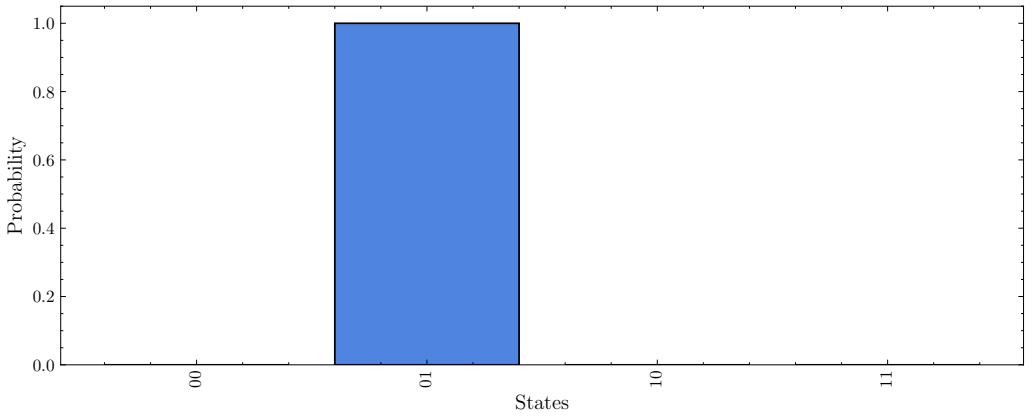


Figure B.2: The state $|\psi\rangle$ before the phase kickback. The probabilities of measuring the state in the computational basis are $P(01) = 1$ and $P(11) = P(00) = P(10) = 0$.

$$H_1 |0\rangle |1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) |1\rangle = \frac{1}{\sqrt{2}}\big(|01\rangle + |11\rangle\big) \tag{2.6}$$

Let's consider then a gate G, which represent an operation for which $|1\rangle$ is eigenstate with eigenvalue λ_G. We can construct a controlled version of this gate, which is applying G to the target qubit only if the state of the control qubit is $|1\rangle$. We apply this controlled gate CG to our system, using the input register as control and the output register as target:

$$CG\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |1\rangle = CG\frac{1}{\sqrt{2}}(|01\rangle + |11\rangle) = \tag{2.7}$$

$$= \frac{1}{\sqrt{2}}(|01\rangle + G|11\rangle) = \frac{1}{\sqrt{2}}(|01\rangle + \lambda_G |11\rangle) = \tag{2.8}$$

$$= \frac{1}{\sqrt{2}}(|0\rangle + \lambda_G |1\rangle) \otimes |1\rangle . \tag{2.9}$$

Interestingly, applying a controlled gate on the target ancilla qubit has the effect of introducing a phase shift in the state of the control qubit, leaving the state of the target qubit unchanged. This action motivates the name of this routine, known as *phase kickback* and is a key ingredient in many quantum algorithms, where such controlled (often multi-controlled) gates are used to manipulate the state of the input register.

Finally, to see the effect of the phase kickback on the system state, we need to rotate the state of the control qubit again, re-applying the Hadamard gate. In fact, the phase kickback has introduced a phase shift in the state of the control qubit, which cannot be directly observed in the computational basis. The procedure of moving from the computational basis to the superposition basis is common in quantum algorithms.

For the phase kickback to be effective, the controlled gate G must have a well-defined eigenvalue λ_G for the state of the target qubit. This ensures that its effect is realized on the input register only. For example, we can consider a controlled T gate, since $T|1\rangle = \exp\{i\frac{\pi}{4}\}|1\rangle$. In this case, the phase kickback will introduce a phase shift of $\pi/4$ to the state of the control qubit and the probabilities to measure the components of the state of the system after the phase kickback will be as represented in Figure B.3.

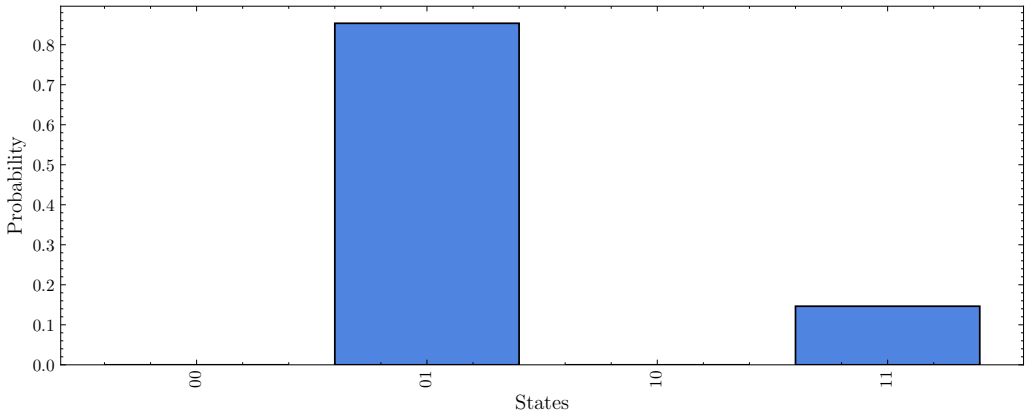


Figure B.3: The state $|\psi\rangle$ after the phase kickback. The probabilities of measuring the state in the computational basis are $P(01) = 0.85355339$ and $P(11) = 0.14644661$, while $P(00) = P(10) = 0$.

B.3 Grover's Search Algorithm

Having introduced the concept of ancilla qubits in Section B.1 and the phase kickback in Section B.2, we can now explore their application in Grover's search algorithm.

Grover's search algorithm is a quantum algorithm that provides a quadratic speedup for searching an unsorted database. It was proposed by Lov Grover in 1996 [17] and and its design is based on the idea of embedding the correct solution into one of the amplitudes of the system states, whose probability is iteratively amplified until it can be easily measured among the others.

Following the notation introduced in this `Qiboedu` example, we can consider a dummy database of $N = 32$ elements, which can be represented by a 5-qubit system. Also, we assume that the solution we are looking for is represented by one of the components of the system state and we denote the target state as $|\omega\rangle$. In particular, we assume that the target state is $|\omega\rangle = |01011\rangle$.

The structure of the Grover's algorithm can be divided into four main steps, involving n qubits in the input register and one ancilla qubit in the output register:

1. *State preparation*: Initialize the input register in a uniform superposition and the output register in $|1\rangle$.
2. *Oracle query*: Apply an operation that marks the target state by flipping its amplitude sign.
3. *Amplitude amplification*: Enhance the probability of the marked state via specific unitary operations.
4. *Iteration*: Repeat oracle and amplification steps to maximize success probability.

The corresponding circuit representation of Grover's search algorithm in case of two iteration of points 2. and 3. is shown in Figure B.4.

B.3.1 State Preparation

To prepare the initial state, we start by applying an Hadamard gate to all the qubits in the input register, which creates a uniform superposition of all the possible states

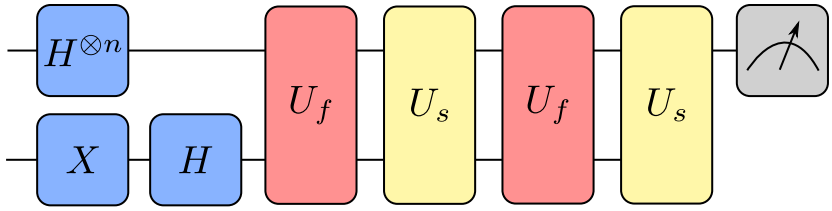


Figure B.4: The circuit representation of Grover's search algorithm in the case of two iterations.

$|s\rangle$. The output register is initialized in the state $|1\rangle$, which is the state we will use to mark the target state. This choice is not casual, and it reminds us of the phase kickback routine introduced in Section B.2. In fact, the output register will be used to perform manipulations on the input register in a similar way of the one presented in Section B.2.

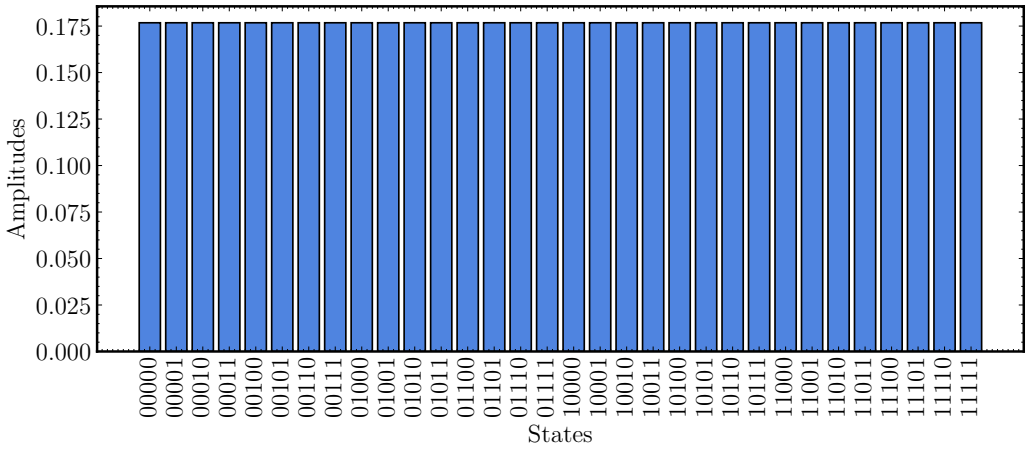


Figure B.5: Grover's algorithm, initial step: state of the input register after the state preparation.

Focusing on the input register, after the state preparation, the amplitudes are the ones represented in Figure B.5.

B.3.2 Oracle Query

Once the initial state is prepared, we need to mark the target state $|\omega\rangle$. This is done by applying an *oracle* operation U_f, which flips the sign of the amplitude associated with the target state. The oracle can be represented as:

$$U_f |s\rangle |1\rangle = |s\rangle |1 \oplus f(s)\rangle , \tag{2.10}$$

where $f(s)$ is a function that returns 1 if $s = \omega$ and 0 otherwise. This operation effectively marks the target state by flipping its amplitude sign, which can be interpreted as a phase kickback operation. The oracle operation is usually implemented in form of a multi-controlled operation, where the input register is used to trigger a phase kickback using the ancilla qubit. For this reason, it is important to mention that the oracle operation contains information about the target state $|\omega\rangle$, which is the solution we are looking for.

This can seem counterintuitive, but we have to think that the oracle is usually prepared to be used then by a user, who is not requested to study or know about its content[1].

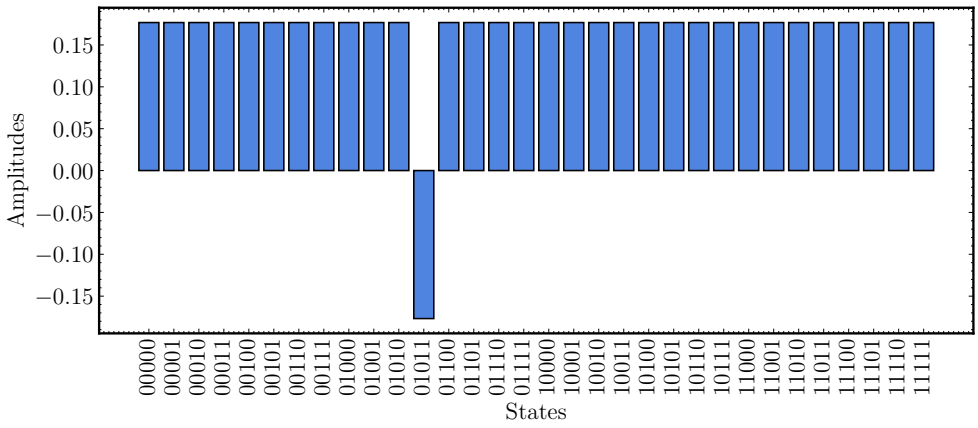


Figure B.6: Grover's algorithm, marking the target: the state of the input register after the oracle query. The oracle operation flips the sign of the amplitude associated with the target state $|\omega\rangle$.

The action of the oracle on the initial state is to flip the sign of the amplitude associated with the target state:

$$U_f\left(H^{\otimes n}\left|0\right\rangle^{\otimes n}\left|1\right\rangle\right) = \frac{1}{\sqrt{2^n}}\sum_{s=0}^{2^n-1}\left|s\right\rangle\left|1\oplus f(s)\right\rangle = \frac{1}{\sqrt{2^n}}\sum_{s=0}^{2^n-1}(-1)^{f(s)}\left|s\right\rangle\left|1\right\rangle. \tag{2.11}$$

Figure B.6 shows the state of the input register after the oracle query, where the amplitude associated with the target state $|\omega\rangle$ has been flipped.

B.3.3 Amplitude Amplification

Once the target state has been marked, we need to amplify its probability of being measured. This is done by applying a series of operations that enhance the amplitude of the target state while reducing the amplitudes of the other states. The amplitude amplification is usually implememented by means of a reflection operation. In particular, we perform a reflection around the state $|s\rangle$, namely, the uniform superposition of all the states in the input register. This is done by applying the following operation:

$$U_s = 2\left|s\right\rangle\left\langle s\right| - I, \tag{2.12}$$

where I is the identity operator. The reflection operation effectively inverts the amplitudes of the states with respect to the state $|s\rangle$, thus amplifying the amplitude of the target state $|\omega\rangle$.

The probabilities associated to the state of the input register after one round of oracle query and amplitude amplification is shown in Figure B.7. We start here representing the probabilities because, in the end, this is what we can measure once the state is prepared.

[1]The usuer is supposed to consult the oracle as a black box, and this assumption is also motivating the name oracle itself.

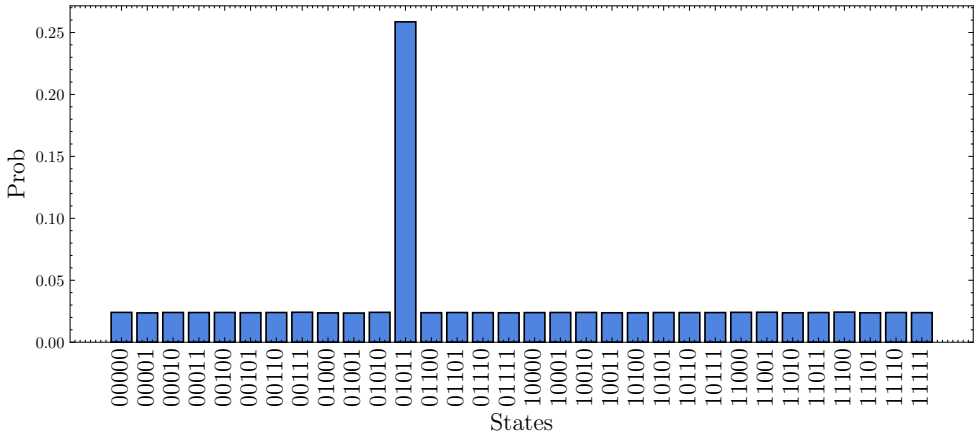


Figure B.7: Grover's algorithm, a full Grover iteration: the state of the input register after one round of oracle query and amplitude amplification. The amplitude of the target state $|\omega\rangle$ has been amplified, while the amplitudes of the other states have been reduced.

Section B.3.4 discusses the need of repeating the application of U_f and U_s an optimal number K of times to maximise the probability of measuring the target state $|\omega\rangle$. For now, we start showing the state of the input register after four rounds of oracle query and amplitude amplification in Figure B.8. It is clear that the amplitude of the target state $|\omega\rangle$ has been further amplified, while the amplitudes of the other states have been further reduced. Grover's is not a deterministic algorithm, so the probability of measuring the target state is not 1, but it is significantly higher than the initial probability, which was $1/2^n$.

B.3.4 Iteration

In this last section, we discuss the need of repeating the application of the oracle and the amplitude amplification operations to maximize the probability of measuring the target state $|\omega\rangle$.

To understand how many times we need to repeat the procedure, let's start considering a general state of the input register $|\psi\rangle$, which can always be decomposed into a component parallel to the target state $|\omega\rangle$ and a component orthogonal to it. If we start from the initialisation of the input register, since only one componend it parallel to the target one, we can write the state as:

$$|\psi\rangle = \sqrt{\frac{1}{N}}\,|\omega\rangle + \sqrt{\frac{N-1}{N}}\,|\omega^{\perp}\rangle\,. \tag{2.13}$$

The two coefficients in Equation 2.13 can be simplified if considered as the cosine and sine of an angle θ in the plane spanned by the vectors $|\omega\rangle$ and its orthogonal $|\omega^{\perp}\rangle$:

$$|\psi\rangle = \sin(\theta)\,|\omega\rangle + \cos(\theta)\,|\omega^{\perp}\rangle\,. \tag{2.14}$$

Considering a situation where the unstructured database is containing a big number of elements, we can assume that the angle θ is small, so that $\theta = \arcsin(1/\sqrt{M})$ and $\theta \approx 1/\sqrt{M}$.

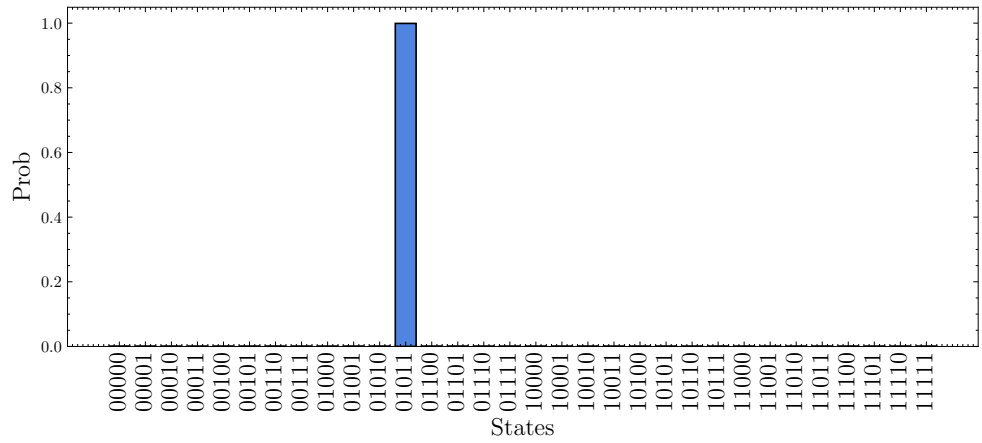


Figure B.8: Grover's algorithm, final look: The state of the input register after four rounds of oracle query and amplitude amplification. The amplitude of the target state $|\omega\rangle$ has been further amplified, while the amplitudes of the other states have been further reduced.

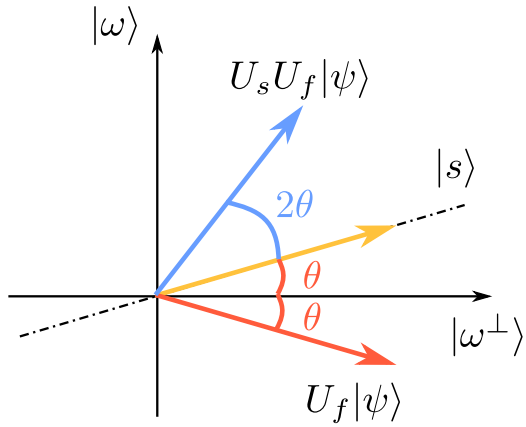


Figure B.9: Schematic representation of the Grover search algorithm: the input register is first prepared in a uniform superposition of all the states (yellow vector), then the oracle operation is applied, which flips the sign of the target amplitude only (red vector). This is equivalent to a reflection around the axis which is orthogonal to the target state. Finally, the amplitude amplification operation is applied, which reflects the state around the state $|s\rangle$ (blue vector). The process' effect is to amplify the component of the whole state which is parallel to the target state $|\omega\rangle$ (y axis).

Noting, as clearly represented in Figure B.9, that the action of the amplitude amplification operation corresponds to a rotation of 2θ, we can conclude that the angle constructed by the input register state after K iterations of the Grover's search algorithm is $\alpha = (2K + 1)\theta$. Maximizing the probability of measuring the target state means maximizing the sine of the angle α, namely, considering $\alpha = 1/2$. This helps us computing the optimal number of iterations K as:

$$K = \frac{\pi}{4\theta} - \frac{1}{2} = \frac{\pi}{4}\sqrt{M} - \frac{1}{2}. \tag{2.15}$$

Since we can perform only an integer number of iterations, Equation 2.15 can be rounded to the nearest integer to obtain the optimal number of iterations K. It is important to note that the optimal number of iterations depends on the square root of the number of elements M in the database, which is a key feature of Grover's search algorithm. This introduces a quadratic speedup compared to classical search algorithms, which would require $M/2$ iterations on average to find the target state.

In the case of our example, where $M = 32$, we have $K \approx 3.43$, which, by choice, we round to $K = 4$. The number of elements in this dummy database is too small to see an advantage, and introduces uncertainty in the approximation of the angle.

As shown in Figure B.9, the repeated application of the oracle and the amplitude amplification can be seen as a rotation of the state vector in a two-dimensional plane. This rotation is periodic, which means that after a certain number of steps the state returns to its initial position. For this reason, it is not useful to apply the procedure more than K times. Beyond this point, the algorithm would only continue the rotation and eventually reproduce states that have already appeared during the process.

APPENDIX C

Adiabatic Quantum Computing

Although digital quantum computing is the most widely used paradigm within this thesis, it can be useful to introduce the concept of adiabatic quantum computing (AQC) as well.

Adiabatic quantum computation (AQC) is an alternative model of quantum computation, based on continuous adiabatic evolution rather than discrete unitary gates, and has been proven to be computationally equivalent to the standard circuit model, up to polynomial overhead [239]. At the basis of AQC is the adiabatic theorem [240], which states that a physical system remains in its instantaneous eigenstate if a given perturbation is acting on it slowly enough and if there is a gap between the eigenvalue and the rest of the Hamiltonian's spectrum.

In a nuthsell, if we consider an initial Hamiltonian H_0 with a known ground state and presenting a gap, and a final Hamiltonian H_1, we can use them to define a time-evolving Hamiltonian

$$H(\tau) = [1 - s(\tau)]H_0 + s(\tau)H_1, \tag{3.1}$$

where $s(\tau)$ is a smooth function, called *scheduling function*, that varies from 0 to 1 as τ goes from 0 to T, where T is the total evolution time of the system. Then, if the scheduling function is slow enough (see theorem 2.1 of [239]), the system will remain in the ground state of H while it is evolving.

A quantum system whose ground state is easy to prepare is required as H_0 and we need to encode the solution of a problem in the ground state of H_1. This last condition is what makes AQC an intriguing approach to optimization problems, but introduces the theoretical and technical challenge of finding a suitable and relizable H_1 for a given problem.

A remarkable example of AQC application is the quantum annealing implemented by D-Wave systems [241], which is a commercial quantum computing platform based on superconducting qubits. D-Wave's quantum annealers are designed to solve optimization problems by encoding them into the ground state of an Ising Hamiltonian:

$$H_1 = -\sum_{i,j} J_{ij}\sigma_i^z\sigma_j^z - \sum_i h_i\sigma_i^z, \tag{3.2}$$

where σ_i^z are the Pauli Z operators acting on the i-th qubit, J_{ij} are the coupling constants between the qubits, and h_i are the local magnetic fields. The quantum annealer starts from a simple Hamiltonian H_0 that is easy to prepare, and then it slowly evolves to the Ising Hamiltonian H_1 using a scheduling function $s(\tau)$, which is typically chosen to be linear or polynomial in τ.

This quantum annealing approach resonates with a well known, classical, optimization technique: simulated annealing [242]. In fact, the name of the D-Wave's quantum

annealer is inspired by the classical simulated annealing, which is a probabilistic technique for approximating the global minimum of a given function. In a nutshell, simulated annealing is a probabilistic optimization technique inspired by the physical process of annealing in metallurgy. In this classical algorithm, the system starts at a high fictitious temperature T, where it can freely explore the solution space by accepting both better and worse solutions with certain probabilities. The acceptance probability for a solution with energy difference ΔE is given by the Boltzmann factor:

$$P(\Delta E) = \exp\left(-\frac{\Delta E}{k_B T}\right) \tag{3.3}$$

where k_B is the Boltzmann constant and ΔE represents the energy difference between the current and proposed solution states. The energy of a solution is directly associated with the objective function value we wish to minimize: higher energy corresponds to worse solutions, while lower energy represents better solutions. This mapping allows us to treat optimization as finding the ground state (minimum energy configuration) of the system.

As the algorithm progresses, the temperature is gradually reduced according to a cooling schedule $T(t)$, typically following an exponential or polynomial decay. This cooling process reduces the probability of accepting worse solutions over time, allowing the system to settle into increasingly better local minima. At high temperatures, the system can readily accept worse solutions, enabling exploration of the solution space and escape from local optima through thermal fluctuations. As temperature decreases, the acceptance probability becomes more selective, focusing the search on promising regions. The algorithm terminates when the temperature reaches a sufficiently low value or when convergence criteria are met.

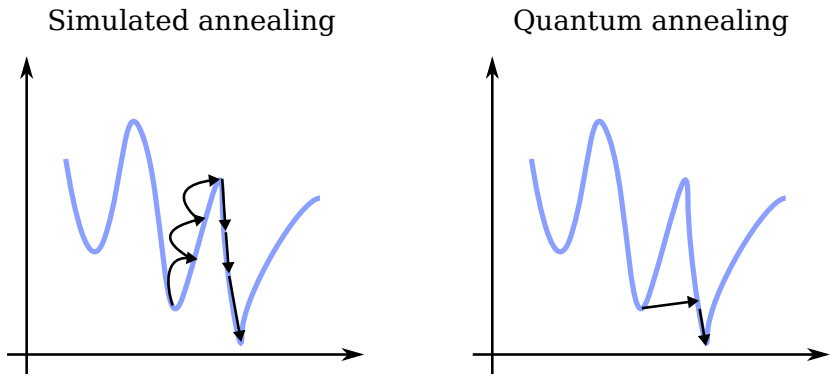


Figure C.1: Comparison between simulated annealing and quantum annealing. In simulated annealing, the search of the optimal solution is forced to follow the surface of the energy landscape, while in quantum annealing, the system can explore the solution space by tunneling through energy barriers, which can lead to finding the global minimum more efficiently.

Similarly, in quantum annealing, the system starts from a simple Hamiltonian and then it is slowly evolved to a more complex Hamiltonian, which encodes the solution to the optimization problem. The main difference is that quantum annealing exploits quantum effects, such as quantum tunneling, to explore the solution theoretically more efficiently than classical simulated annealing.

A schematic representation of the difference between simulated annealing and quantum annealing is shown in Figure C.1.

Given the limited connectivity of the qubits in D-Wave's quantum annealers, the problems that can be solved are usually formulated as quadratic unconstrained binary optimization (QUBO) problems, which, as mentioned earlier, can be mapped to the Ising model. This limitation makes D-Wave a specific implementation of AQC, rather than a general-purpose quantum computer. Also, as it is for digital quantum computers, also quantum annealers' technological development suffers from the same challenges, such as noise and decoherence, which can affect the quality of the solutions obtained.

On the other hand, it is true that this kind of optimization appears often as a subroutine in many larger algorithms (classical or quantum), so the ability to solve optimization problems efficiently is a valuable asset.

In Chapter 11 we will discuss a dummy exercise of quantum machine learning, where classical computing, adiabatic quantum computing and digital quantum computing are combined to learn the density profile of a statistical distribution. This cooperation between different computing paradigms is a promising direction for the future of computing, where the strengths of each paradigm can be leveraged to solve complex problems more efficiently.

APPENDIX D

Hamiltonian Simulation

One of the most fundamental tasks in quantum computing is the simulation of quantum systems governed by a Hamiltonian H. As originally envisioned by Feynman [19] and later formalized by Lloyd [54], a quantum computer can efficiently reproduce the dynamics of another quantum system, a task that is generally intractable for classical computers due to the exponential growth of the Hilbert space.

The problem of Hamiltonian simulation can be stated as follows: given a Hamiltonian H acting on n qubits, we want to implement the unitary time-evolution operator

$$U(t) = e^{-iHt}, \tag{4.1}$$

for some evolution time t. This operator describes the continuous-time dynamics of a quantum system according to the Schrödinger equation. In the context of digital quantum computing, the challenge is to approximate $U(t)$ using a sequence of discrete quantum gates from a universal gate set. This makes Hamiltonian simulation essential for turning theoretical models into circuits executable on quantum hardware.

In the context of this thesis, Hamiltonian simulation will play a crucial role in at least two of the applications we will explore. In particular, in Chapter 11 we will introduce an hybrid routine which makes use of adiabatic computing together with digital quantum circuits. The bridge between these two paradigms is given by Hamiltonian simulation. Again, in Chapter 12 we propose a quantum algorithm for approximating ground states of target Hamiltonians. The theoretical framework of the algorithm involves several Hamiltonian simulation steps. The tools we use to synthetise quantum circuits from these Hamiltonian simulation are among the ones presented in this section.

D.1 Trotter-Suzuki Decomposition

A widely used approach to Hamiltonian simulation is the *Trotter-Suzuki decomposition*. The central task is to approximate the time-evolution operator

$$U(t) = e^{-iHt}, \tag{4.2}$$

for a Hamiltonian that can be decomposed as a sum of simpler pieces,

$$H = \sum_{j=1}^{m} H_j, \tag{4.3}$$

where each H_j acts on only a few qubits (typically one- or two-qubit operators). Because the H_j generally satisfy $[H_j, H_k] \neq 0$, $U(t)$ cannot in general be factorized exactly into a product of exponentials of the terms.

Lie product formula. The starting point is the Lie-Trotter product formula,

$$e^{-iHt} = \lim_{r\to\infty} \left(\prod_{j=1}^{m} e^{-iH_j t/r} \right)^r . \tag{4.4}$$

In practice, one truncates this limit at a finite number r of "Trotter steps," giving

$$e^{-iHt} \approx \left(\prod_{j=1}^{m} e^{-iH_j t/r} \right)^r . \tag{4.5}$$

As r increases, the approximation improves, though at the expense of more circuit depth since each step requires implementing all m exponentials.

Error analysis via BCH. The quality of the approximation can be quantified using the Baker-Campbell-Hausdorff (BCH) expansion. For two operators H_1 and H_2, one finds

$$e^{\delta(H_1+H_2)} = e^{\delta H_1} e^{\delta H_2} \exp\Big(-\tfrac{\delta^2}{2}[H_1, H_2] + O(\delta^3) \Big) . \tag{4.6}$$

Thus one Trotter step of size $\delta = t/r$ introduces a local error of order $O(\delta^2)$ due to commutator terms. After r such steps, the global error scales as $O(t^2/r)$.

Symmetric splitting. The leading error can be canceled by employing a symmetric ordering. For example, with two terms one uses

$$e^{\delta(H_1+H_2)} \approx e^{\frac{\delta}{2}H_1} e^{\delta H_2} e^{\frac{\delta}{2}H_1}, \tag{4.7}$$

which yields a local error of $O(\delta^3)$ and hence a global error scaling as $O(t^3/r^2)$. Suzuki showed how to construct general higher-order formulas recursively. For instance, fourth-order formulas achieve local error $O(\delta^5)$. These higher-order formulas involve more exponentials per step, but can reduce the required number of steps r for a given target precision.

Circuit implementation. Each short-time evolution operator $e^{-iH_j t/r}$ acts nontrivially on only a few qubits, and can therefore be compiled into a shallow subcircuit. For example, if H_j is a Pauli string (a product of single-qubit Pauli operators), then $e^{-iH_j\theta}$ is equivalent to a rotation about that Pauli axis. The standard compilation involves a basis change (to map H_j into Z operators), a controlled-Z rotation or single-qubit R_z gate, and the inverse basis change. Thus the decomposition gives a constructive prescription for translating abstract Hamiltonian dynamics into circuits of elementary gates.

D.2 Approximation Errors

Scaling. For the first-order product formula, the global error is bounded by

$$\epsilon = \mathcal{O}\left(\frac{t^2}{r} \sum_{j<k} \|[H_j, H_k]\| \right), \tag{4.8}$$

where $\|\cdot\|$ denotes the operator norm. Achieving error ϵ requires $r = \mathcal{O}(t^2/\epsilon)$ steps.

The scaling improves with higher-order product formulas:

$$\text{Second order:} \quad r = \mathcal{O}\left(\frac{t^{3/2}}{\sqrt{\epsilon}}\right), \tag{4.9}$$

$$\text{Fourth order:} \quad r = \mathcal{O}\left(\frac{t^{5/4}}{\epsilon^{1/4}}\right). \tag{4.10}$$

In general, increasing the order reduces the number of steps required but adds more exponentials per step.

Trade-offs and context. The Trotter-Suzuki hierarchy provides a flexible balance between accuracy and circuit depth. Simulation is efficient if the target precision can be reached with resources polynomial in the system size n, the evolution time t, and the inverse accuracy $1/\epsilon$. At the same time, it is known that Trotter-Suzuki formulas are not optimal in asymptotic scaling: more advanced methods such as Taylor series simulation, qubitization, or quantum signal processing achieve polylogarithmic scaling in $1/\epsilon$. Nonetheless, Trotter-Suzuki is conceptually simple, physically transparent, and often competitive in practical or near-term scenarios.

D.3 Advanced Simulation Techniques

Beyond Trotterization, several more advanced Hamiltonian simulation methods have been developed. The time-evolving block decimation (TEBD) algorithm is a classical technique for one-dimensional systems with limited entanglement, built on the same Trotter-Suzuki ideas [243, 95]. On the quantum side, quantum signal processing (QSP) and qubitization achieve near-optimal scaling in time and precision [244], while linear combination of unitaries (LCU) provides a flexible probabilistic framework that underlies many modern algorithms.

These approaches show the diversity of strategies for Hamiltonian simulation. TEBD is powerful for classical low-entanglement systems, while QSP, qubitization, and LCU target fault-tolerant quantum computers. The method of choice depends on hardware, Hamiltonian structure, and required precision.

D.4 Connection to Adiabatic Computing

Hamiltonian simulation also connects the digital and analog paradigms of quantum computing. In adiabatic computing, the system evolves under a time-dependent Hamiltonian $H(t)$. A digital quantum computer can approximate this by discretizing time and applying Trotter steps with updated $H(t)$. This allows adiabatic algorithms to be implemented on gate-based devices, a connection we will explore in a practical application described in Chapter 11.

Bibliography

[1] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[2] Stavros Efthymiou, Marco Lazzarin, Andrea Pasquale, and Stefano Carrazza. Quantum simulation with just-in-time compilation. *Quantum*, 6:814, September 2022.

[3] Stavros Efthymiou, Sergi Ramos-Calderer, Carlos Bravo-Prieto, Adrián Pérez-Salinas, Diego García-Martín, Artur Garcia-Saez, José Ignacio Latorre, and Stefano Carrazza. Qibo: a framework for quantum simulation with hardware acceleration. *Quantum Science and Technology*, 7(1):015018, December 2021.

[4] Marco Ballarin, Francesco Pio Barone, Alberto Coppi, Daniel Jaschke, Simone Montangero, Guillermo Muñoz Menés, Davide Rattacaso, and Nora Reinić. Quantum tea: qmatchatea, 2025. Cite for qmatchatea.

[5] Harun Bayraktar, Ali Charara, David Clark, Saul Cohen, Timothy Costa, Yao-Lung L. Fang, Yang Gao, Jack Guan, John Gunnels, Azzam Haidar, Andreas Hehn, Markus Hohnerbach, Matthew Jones, Tom Lubowe, Dmitry Lyakh, Shinya Morino, Paul Springer, Sam Stanwyck, Igor Terentyev, Satya Varadhan, Jonathan Wong, and Takuma Yamaguchi. cuquantum sdk: A high-performance library for accelerating quantum science, 2023.

[6] Stavros Efthymiou, Alvaro Orgaz-Fuertes, Rodolfo Carobene, Juan Cereijo, Andrea Pasquale, Sergi Ramos-Calderer, Simone Bordoni, David Fuentes-Ruiz, Alessandro Candido, Edoardo Pedicillo, Matteo Robbiati, Yuanzheng Paul Tan, Jadwiga Wilkens, Ingo Roth, José Ignacio Latorre, and Stefano Carrazza. Qibolab: an open-source hybrid quantum operating system. *Quantum*, 8:1247, February 2024.

[7] Andrea Pasquale, Edoardo Pedicillo, Juan Cereijo, Sergi Ramos-Calderer, Alessandro Candido, Gabriele Palazzo, Rodolfo Carobene, Marco Gobbo, Stavros Efthymiou, Yuanzheng Paul Tan, Ingo Roth, Matteo Robbiati, Jadwiga Wilkens, Alvaro Orgaz-Fuertes, David Fuentes-Ruiz, Andrea Giachero, Frederico Brito, José Ignacio Latorre, and Stefano Carrazza. Qibocal: an open-source framework for calibration of self-hosted quantum devices, 2024.

[8] M. A. Rol, F. Battistel, F. K. Malinowski, C. C. Bultink, B. M. Tarasinski, R. Vollmer, N. Haider, N. Muthusubramanian, A. Bruno, B. M. Terhal, and L. DiCarlo. Fast, high-fidelity conditional-phase gate exploiting leakage interference in weakly anharmonic superconducting qubits. *Phys. Rev. Lett.*, 123:120502, Sep 2019.

[9] Matteo Robbiati, Alejandro Sopena, Andrea Papaluca, and Stefano Carrazza. Real-time error mitigation for variational optimization on quantum hardware, 2023.

[10] D. Maître and R. Santos-Mateos. Multi-variable integration with a neural network. *Journal of High Energy Physics*, 2023(3), March 2023.

[11] Juan M Cruz-Martinez, Matteo Robbiati, and Stefano Carrazza. Multi-variable integration with a variational quantum circuit. *Quantum Science and Technology*, 9(3):035053, June 2024.

[12] Matteo Robbiati, Juan M. Cruz-Martinez, and Stefano Carrazza. Determining probability density functions with adiabatic quantum computing. *Quantum Machine Intelligence*, 7(1), January 2025.

[13] Marek Gluza. Double-bracket quantum algorithms for diagonalization. *Quantum*, 8:1316, April 2024.

[14] Matteo Robbiati, Edoardo Pedicillo, Andrea Pasquale, Xiaoyue Li, Andrew Wright, Renato M. S. Farias, Khanh Uyen Giang, Jeongrak Son, Johannes Knörzer, Siong Thye Goh, Jun Yong Khoo, Nelly H. Y. Ng, Zoë Holmes, Stefano Carrazza, and Marek Gluza. Double-bracket quantum algorithms for high-fidelity ground state preparation, 2024.

[15] Elina Fuchs, Fiona Kirk, Agnese Mariotti, Jan Richter, and Matteo Robbiati. Towards a global search for new physics with isotope shifts, 2025.

[16] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997.

[17] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996.

[18] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15), October 2009.

[19] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6/7):467–488, 1982.

[20] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, August 2018.

[21] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), July 2014.

[22] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, September 2017.

[23] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C. Benjamin, and Xiao Yuan. Quantum computational chemistry. *Reviews of Modern Physics*, 92(1), March 2020.

[24] Nathan Wiebe, Christopher Granade, Christopher Ferrie, and D.G. Cory. Hamiltonian learning and certification using quantum resources. *Physical Review Letters*, 112(19), May 2014.

[25] Antonio A. Gentile, Brian Flynn, Sebastian Knauer, Nathan Wiebe, Stefano Paesani, Christopher E. Granade, John G. Rarity, Raffaele Santagati, and Anthony Laing. Learning models of quantum systems from experiments. *Nature Physics*, 17(7):837–843, April 2021.

[26] Edoardo Pedicillo, Andrea Pasquale, and Stefano Carrazza. Benchmarking ma-

chine learning models for quantum state classification, 2023.

[27] Brian Flynn, Antonio A Gentile, Nathan Wiebe, Raffaele Santagati, and Anthony Laing. Quantum model learning agent: characterisation of quantum systems through machine learning. *New Journal of Physics*, 24(5):053034, jun 2022.

[28] Simone Bordoni, Andrea Papaluca, Piergiorgio Buttarini, Alejandro Sopena, Stefano Giagu, and Stefano Carrazza. Quantum noise modeling through reinforcement learning, 2024.

[29] Changjun Kim, Kyungdeock Daniel Park, and June-Koo Rhee. Quantum error mitigation with artificial neural network. *IEEE Access*, 8:188853–188860, 2020.

[30] Yuan-Hang Zhang, Pei-Lin Zheng, Yi Zhang, and Dong-Ling Deng. Topological quantum compiling with reinforcement learning. *Physical Review Letters*, 125(17), October 2020.

[31] Lorenzo Moro, Matteo G. A. Paris, Marcello Restelli, and Enrico Prati. Quantum compiling by deep reinforcement learning. *Communications Physics*, 4(1), August 2021.

[32] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. Predicting good quantum circuit compilation options. In *2023 IEEE International Conference on Quantum Software (QSW)*, page 43–53. IEEE, July 2023.

[33] David Kremer, Victor Villar, Hanhee Paik, Ivan Duran, Ismael Faro, and Juan Cruz-Benito. Practical and efficient quantum circuit synthesis and transpiling with reinforcement learning, 2025.

[34] Q Ansel, E Dionis, F Arrouas, B Peaudecerf, S Guérin, D Guéry-Odelin, and D Sugny. Introduction to theoretical and experimental aspects of quantum optimal control. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 57(13):133001, June 2024.

[35] Marko Kuzmanović, Ilya Moskalenko, Yu-Han Chang, Ognjen Stanisavljević, Christopher Warren, Emil Hogedal, Anuj Aggarwal, Irshad Ahmad, Janka Biznárová, Mamta Dahiya, Marcus Rommel, Andreas Nylander, Giovanna Tancredi, and Gheorghe Sorin Paraoanu. Neural-network-based design and implementation of fast and robust quantum gates, 2025.

[36] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5), November 2004.

[37] Jacob Biamonte and Ville Bergholm. Tensor networks in a nutshell, 2017.

[38] Sergey Bravyi and David Gosset. Improved classical simulation of quantum circuits dominated by clifford gates. *Physical Review Letters*, 116(25), June 2016.

[39] Peter W. Shor. Fault-tolerant quantum computation, 1997.

[40] S. Carrazza, S. Efthymiou, M. Lazzarin, and A. Pasquale. An open-source modular framework for quantum computing. *Journal of Physics: Conference Series*, 2438(1):012148, February 2023.

[41] Rodolfo Carobene et al. qiboteam/qibosoq: Qibosoq 0.0.3, July 2023.

[42] Qibolab benchmarks. https://github.com/qiboteam/qibolab-benchmarks/tree/v0.1.0.

[43] N. David Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, USA, 2007.

[44] Guido Bacciagaluppi and Antony Valentini. Quantum theory at the crossroads: Reconsidering the 1927 solvay conference, 2009.

[45] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of*

Statistical Physics, 22(5):563–591, 1980.

[46] Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science*, 560:7–11, December 2014.

[47] C. H. Bennett and G. Brassard. Experimental quantum cryptography: the dawn of a new era for quantum cryptography: the experimental prototype is working]. *SIGACT News*, 20(4):78–80, November 1989.

[48] Charles H. Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K. Wootters. Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. *Phys. Rev. Lett.*, 70:1895–1899, Mar 1993.

[49] Dik Bouwmeester, Jian-Wei Pan, Klaus Mattle, Manfred Eibl, Harald Weinfurter, and Anton Zeilinger. Experimental quantum teleportation. *Nature*, 390(6660):575–579, December 1997.

[50] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

[51] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[52] Daniel R. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997.

[53] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996.

[54] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996.

[55] David P. DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik*, 48(9–11):771–783, September 2000.

[56] Joschka Roffe. Quantum error correction: an introductory guide. *Contemporary Physics*, 60(3):226–245, July 2019.

[57] Zhenyu Cai, Ryan Babbush, Simon C. Benjamin, Suguru Endo, William J. Huggins, Ying Li, Jarrod R. McClean, and Thomas E. O'Brien. Quantum error mitigation. *Reviews of Modern Physics*, 95(4), December 2023.

[58] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, November 1995.

[59] Adam M. Meier, Bryan Eastin, and Emanuel Knill. Magic-state distillation with the four-qubit code, 2012.

[60] Lawrence Livermore National Laboratory. El capitan supercomputer. https://hpc.llnl.gov/hardware/compute-platforms/el-capitan, 2024. Accessed: 2025-09-16.

[61] Mar Tejedor, Berta Casas, Javier Conejero, Alba Cervera-Lierta, and Rosa M. Badia. Distributed quantum circuit cutting for hybrid quantum-classical high-performance computing, 2025.

[62] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, October 2014.

[63] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, September 2017.

[64] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015.

[65] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensor-*

Flow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, Inc., 2nd edition, 2019.

[66] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning representations by back-propagating errors*, page 696–699. MIT Press, Cambridge, MA, USA, 1988.

[67] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1), November 2018.

[68] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, August 2019.

[69] Léo Monbroussou, Eliott Z. Mamon, Jonas Landman, Alex B. Grilo, Romain Kukla, and Elham Kashefi. Trainability and expressivity of hamming-weight preserving quantum circuits for machine learning. *Quantum*, 9:1745, May 2025.

[70] M. Cerezo, Martin Larocca, Diego García-Martín, N. L. Diaz, Paolo Braccia, Enrico Fontana, Manuel S. Rudolph, Pablo Bermejo, Aroosa Ijaz, Supanut Thanasilp, Eric R. Anschuetz, and Zoë Holmes. Does provable absence of barren plateaus imply classical simulability? *Nature Communications*, 16(1), August 2025.

[71] Ricard Puig, Marc Drudis, Supanut Thanasilp, and Zoë Holmes. Variational quantum simulation: A case study for understanding warm starts. *PRX Quantum*, 6(1), January 2025.

[72] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii. Quantum circuit learning. *Physical Review A*, 98(3), September 2018.

[73] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3), March 2019.

[74] Andrea Mari, Thomas R. Bromley, and Nathan Killoran. Estimating the gradient and higher-order derivatives on quantum hardware. *Physical Review A*, 103(1), January 2021.

[75] Amira Abbas, Robbie King, Hsin-Yuan Huang, William J. Huggins, Ramis Movassagh, Dar Gilboa, and Jarrod R. McClean. On quantum backpropagation, information reuse, and cheating measurement collapse, 2023.

[76] Paolo Solinas, Simone Caletti, and Giovanni Minuto. Quantum gradient evaluation through quantum non-demolition measurements. *The European Physical Journal D*, 77(5), May 2023.

[77] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews*, 6(2), June 2019.

[78] David C. McKay, Christopher J. Wood, Sarah Sheldon, Jerry M. Chow, and Jay M. Gambetta. Efficient ¡mml:math xmlns:mml="http://www.w3.org/1998/math/mathml"¿¡mml:mi¿z¡/mml:mi¿¡/mml:math¿ gates for quantum computing. *Physical Review A*, 96(2), August 2017.

[79] F. Motzoi, J. M. Gambetta, P. Rebentrost, and F. K. Wilhelm. Simple pulses for elimination of leakage in weakly nonlinear qubits. *Physical Review Letters*, 103(11), September 2009.

[80] J. M. Gambetta, F. Motzoi, S. T. Merkel, and F. K. Wilhelm. Analytic control methods for high-fidelity unitary operations in a weakly nonlinear oscillator. *Physical Review A*, 83(1), January 2011.

[81] Nic Ezzell, Bibek Pokharel, Lina Tewala, Gregory Quiroz, and Daniel A. Lidar. Dy-

namical decoupling for superconducting qubits: A performance survey. *Physical Review Applied*, 20(6), December 2023.

[82] Qibo Developers. Qibo documentation: Noise channels. https://qibo.science/qibo/stable/api-reference/qibo.html#channels, 2025. Accessed: 2025-09-18.

[83] Dayue Qin, Yanzhu Chen, and Ying Li. Error statistics and scalability of quantum error mitigation formulas. *npj Quantum Information*, 9(1), April 2023.

[84] Kristan Temme, Sergey Bravyi, and Jay M. Gambetta. Error mitigation for short-depth quantum circuits. *Physical Review Letters*, 119(18), November 2017.

[85] Tudor Giurgica-Tiron, Yousef Hindy, Ryan LaRose, Andrea Mari, and William J. Zeng. Digital zero noise extrapolation for quantum error mitigation. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, page 306–316. IEEE, October 2020.

[86] Samson Wang, Piotr Czarnik, Andrew Arrasmith, M. Cerezo, Lukasz Cincio, and Patrick J. Coles. Can error mitigation improve trainability of noisy variational quantum algorithms? *Quantum*, 8:1287, March 2024.

[87] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[88] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[89] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[90] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[91] IBM Quantum. IBM Quantum Platform. https://quantum.cloud.ibm.com/, 2025. Accessed: 2025-09-18.

[92] Inc. Amazon Web Services. Amazon Braket Quantum Computing Service. https://aws.amazon.com/it/braket/, 2025. Accessed: 2025-09-18.

[93] Inc. IonQ. IonQ Quantum Hardware via AWS Braket. https://ionq.com/, 2024. Accessed: 2025-09-18.

[94] Tameem Albash and Daniel A. Lidar. Adiabatic quantum computation. *Reviews of*

Modern Physics, 90(1), January 2018.

[95] Sebastian Paeckel, Thomas Köhler, Andreas Swoboda, Salvatore R. Manmana, Ulrich Schollwöck, and Claudius Hubig. Time-evolution methods for matrix-product states. *Annals of Physics*, 411:167998, December 2019.

[96] Qibo Team. Qiboml: Machine learning tools for Qibo. https://github.com/qiboteam/qiboml, 2025. Accessed: 2025-09-18.

[97] Qibo Team. Qibochem: Quantum chemistry tools for Qibo. https://github.com/qiboteam/qibochem, 2025. Accessed: 2025-09-18.

[98] Rodolfo Carobene, Alessandro Candido, Javier Serrano, Alvaro Orgaz-Fuertes, Andrea Giachero, and Stefano Carrazza. ¡tt¿qibosoq¡/tt¿: an open-source framework for quantum circuit rfsoc programming. *Quantum Science and Technology*, 10(3):035010, April 2025.

[99] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

[100] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. Cupy: A numpy-compatible library for nvidia gpu calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[101] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020.

[102] Wikipedia contributors. Bqp. https://en.wikipedia.org/wiki/BQP. Accessed: 2025-09-27.

[103] Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349, October 2014.

[104] Jacob C Bridgeman and Christopher T Chubb. Hand-waving and interpretive dance: an introductory course on tensor networks. *Journal of Physics A: Mathematical and Theoretical*, 50(22):223001, May 2017.

[105] Johnnie Gray. quimb: A python package for quantum information and many-body calculations. *J. Open Source Softw.*, 3:819, 2018.

[106] Johannes Hauschild, Jakob Unfried, Sajant Anand, Bartholomew Andrews, Marcus Bintz, Umberto Borla, Stefan Divic, Markus Drescher, Jan Geiger, Martin Hefel, Kévin Hémery, Wilhelm Kadow, Jack Kemp, Nico Kirchner, Vincent S. Liu, Gunnar Moller, Daniel Parker, Michael Rader, Anton Romen, Samuel Scalet, Leon Schoonderwoerd, Maximilian Schulz, Tomohiro Soejima, Philipp Thoma, Yantao Wu, Philip Zechmann, Ludwig Zweng, Roger Mong, Mike Zaletel, and Frank Pollmann. Tensor network python (tenpy) version 1. *SciPost Physics Codebases*, November 2024.

[107] Matthew Fishman, Steven White, and Edwin Stoudenmire. The itensor software library for tensor network calculations. *SciPost Physics Codebases*, August 2022.

[108] Manuel S. Rudolph, Enrico Fontana, Zoë Holmes, and Lukasz Cincio. Classical surrogate simulation of quantum systems with lowesa, 2023.

[109] Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, July 2021.

[110] Manuel S. Rudolph, Tyson Jones, Yanting Teng, Armando Angrisani, and Zoë Holmes. Pauli propagation: A computational framework for simulating quantum systems, 2025.

[111] Qibo Developers. Qibo documentation: Clifford simulation. https://qibo.science/qibo/stable/api-reference/qibo.html#clifford-simulation. Accessed: 2025-09-25.

[112] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, February 2017.

[113] Asja Fischer and Christian Igel. An introduction to restricted boltzmann machines. In Luis Alvarez, Marta Mejail, Luis Gomez, and Julio Jacobo, editors, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 14–36, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[114] Matus Telgarsky. Representation benefits of deep feedforward networks, 2015.

[115] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[116] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[117] Eduardo Ibarra-García-Padilla, Hannah Lange, Roger G. Melko, Richard T. Scalettar, Juan Carrasquilla, Annabelle Bohrdt, and Ehsan Khatami. Autoregressive neural quantum states of fermi hubbard models. *Physical Review Research*, 7(1), February 2025.

[118] Yangjun Wu, Chu Guo, Yi Fan, Pengyu Zhou, and Honghui Shang. Nnqs-transformer: an efficient and scalable neural network quantum states approach for ab initio quantum chemistry, 2023.

[119] Matthew L. Goh, Martin Larocca, Lukasz Cincio, M. Cerezo, and Frédéric Sauvage. Lie-algebraic classical simulations for quantum computing, 2025.

[120] Ismael Faro, Iskandar Sitdikov, David Garcia Valiñas, Francisco Jose Martin Fernandez, Christopher Codella, and Jennifer Glick. Middleware for quantum: An orchestration of hybrid quantum-classical systems. In *2023 IEEE International Conference on Quantum Software (QSW)*, pages 1–8, 2023.

[121] A. Pasquale. *Open-source Middleware for Quantum Computing*. PhD thesis, Milan U., 2024.

[122] Jens Hedegaard Nielsen, Mikhail Astafev, William H.P. Nielsen, Dominik Vogel, lakhotiaharshit, Alex Johnson, AUC Hardal, Akshita, sohail chatoor, Farzad Bonabi, Liang, Giulio Ungaretti, Sebastian Pauka, Trevor Morgan, Adriaan, Pieter Eendebak, Bas Nijholt, qSaevar, Stefan Droege, Samantha, Jana Darulova, Ruben van Gulik, Natalie Pearson, ThorvaldLarsen, and Andrea Corna. QCoDeS/Q-codes: QCoDeS 0.43.0. Software, January 2024.

[123] M.A. Rol, C. Dickel, S. Asaad, N.K. Langford, C.C. Bultink, R. Sagastizabal, G. de Lange, X. Fu, S.R. de Jong, F. Luthi, and W. Vlothuizen. DiCarloLab-Delft/PycQED_py3: Initial public release. Software, October 2016.

[124] Keysight Technologies. Labber: Measurement Automation Software. https://www.keysight.com/us/en/lib/software-detail/instrument-firmware-software/labber-3113052.html, 2022. Instrument control and measurement automation software.

[125] Qblox. https://www.qblox.com.

[126] QuantumMachines. https://www.quantum-machines.co/.

[127] Zurich Instruments. https://www.zhinst.com/.

[128] Leandro Stefanazzi, Kenneth Treptow, Neal Wilcer, Chris Stoughton, Collin Bradford, Sho Uemura, Silvia Zorzetti, Salvatore Montella, Gustavo Cancelo, Sara Sussman, Andrew Houck, Shefali Saxena, Horacio Arnaldi, Ankur Agrawal, Helin Zhang, Chunyang Ding, and David I. Schuster. The QICK (quantum instrumentation control kit): Readout and control for qubits and detectors. *Review of Scientific*

Instruments, 93(4), April 2022.
[129] Qblox. https://qblox-qblox-instruments.readthedocs-hosted.com/en/master/cluster/qrm_rf.html, 2023.
[130] Qblox. https://qblox-qblox-instruments.readthedocs-hosted.com/en/master/cluster/qcm_rf.html, 2023.
[131] Qblox. https://qblox-qblox-instruments.readthedocs-hosted.com/en/master/cluster/qcm.html, 2023.
[132] Qblox. https://qblox-qblox-instruments.readthedocs-hosted.com/en/master/cluster/synchronization.html#synq.
[133] Qblox. https://qblox-qblox-instruments.readthedocs-hosted.com/en/master/, 2023.
[134] Qcodes. https://github.com/erainstruments, 2021.
[135] Qblox. https://qblox-qblox-instruments.readthedocs-hosted.com/en/master/tutorials/q1asm_tutorials.html, 2023.
[136] OPX+. https://www.quantum-machines.co/products/opx/.
[137] Lior Ella, Lorenzo Leandro, Oded Wertheim, Yoav Romach, Ramon Szmuk, Yoel Knol, Nissim Ofek, Itamar Sivan, and Yonatan Cohen. Quantum-classical processing and benchmarking at the pulse-level, 2023.
[138] ZurichInstruments. https://www.zhinst.com/others/en/products/shfqc-qubit-controller, 2023.
[139] ZurichInstruments. https://www.zhinst.com/others/en/products/hdawg-arbitrary-waveform-generator, 2023.
[140] ZurichInstruments. https://www.zhinst.com/others/en/products/pqsc-programmable-quantum-system-controller, 2023.
[141] ZurichInstruments. https://www.zhinst.com/others/en/quantum-computing-systems/labone-q, 2023.
[142] Xilinx-(AMD). Rfsoc 4x2 specifications. https://www.xilinx.com/support/university/xup-boards/RFSoC4x2.html, 2022.
[143] Xilinx-(AMD). Zcu111 specifications. https://www.xilinx.com/products/boards-and-kits/zcu111.html, 2022.
[144] Xilinx-(AMD). Zcu216 specifications. https://www.xilinx.com/products/boards-and-kits/zcu216.html, 2022.
[145] Erasynth. https://www.erasynth.com, 2021.
[146] Rohde&Schwarz. https://www.rohde-schwarz.com/, 2025.
[147] Yvonne Y. Gao, M. Adriaan Rol, Steven Touzard, and Chen Wang. A practical guide for building superconducting quantum devices, 2021.
[148] J. Emerson, R. Alicki, and K. Zyczkowski. Scalable noise estimation with random unitary operators. *J. Opt. B*, 7:S347–S352, 2005.
[149] 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. *Physical Review A*, 77(1), jan 2008.
[150] B. Lévi, C. C. López, J. Emerson, and D. G. Cory. Efficient error characterization in quantum information processing. *Phys. Rev. A*, 75:022314, 2007.
[151] C. Dankert, R. Cleve, J. Emerson, and E. Livine. Exact and approximate unitary 2-designs and their application to fidelity estimation. *Phys. Rev. A*, 80:012304, 2009.
[152] J. Helsen, I. Roth, E. Onorati, A. H. Werner, and J. Eisert. A general framework for randomized benchmarking. *arXiv:2010.07974*, 3:020357, 2022.
[153] Martin Kliesch and Ingo Roth. Theory of quantum system certification. *PRX Quan-*

tum, 2:010201, 2021.

[154] John F. Clauser, Michael A. Horne, Abner Shimony, and Richard A. Holt. Proposed experiment to test local hidden-variable theories. *Phys. Rev. Lett.*, 23:880–884, Oct 1969.

[155] Ewout van den Berg, Zlatko K. Minev, and Kristan Temme. Model-free readout-error mitigation for quantum expectation values. *Physical Review A*, 105(3), mar 2022.

[156] J. S. Bell. On the einstein podolsky rosen paradox. *Physics Physique Fizika*, 1:195–200, Nov 1964.

[157] Richard D. Ball, Stefano Carrazza, Juan Cruz-Martinez, Luigi Del Debbio, Stefano Forte, Tommaso Giani, Shayan Iranipour, Zahari Kassabov, Jose I. Latorre, Emanuele R. Nocera, Rosalyn L. Pearson, Juan Rojo, Roy Stegeman, Christopher Schwan, Maria Ubiali, Cameron Voisey, and Michael Wilson. The path to proton structure at 1% accuracy. *The European Physical Journal C*, 82(5), may 2022.

[158] Adriá n Pérez-Salinas, Juan Cruz-Martinez, Abdulla A. Alhajri, and Stefano Carrazza. Determining the proton content with a quantum computer. *Physical Review D*, 103(3), feb 2021.

[159] Technology Innovation Institute. Quantum Computation Research at TII. https://www.tii.ae/quantum/our-research/quantum-computation, 2025. Accessed: 2025-09-18.

[160] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[161] Qibocal documentation. https://qibo.science/qibocal/stable/, 2023.

[162] Grafana Labs. Grafana: Open-source analytics and monitoring platform. https://grafana.com/, 2025. Accessed: 2025-09-18.

[163] Carlos Bravo-Prieto, Julien Baglio, Marco Cè, Anthony Francis, Dorota M. Grabowska, and Stefano Carrazza. Style-based quantum generative adversarial networks for monte carlo events. *Quantum*, 6:777, August 2022.

[164] Matteo Robbiati, Andrea Papaluca, Andrea Pasquale, Edoardo Pedicillo, Renato M. S. Farias, Alejandro Sopena, Mattia Robbiano, Ghaith Alramahi, Simone Bordoni, Alessandro Candido, Niccolò Laurora, Jogi Suda Neto, Yuanzheng Paul Tan, Michele Grossi, and Stefano Carrazza. Qiboml: towards the orchestration of quantum-classical machine learning, 2025.

[165] Adrián Pérez-Salinas, David López-Núñez, Artur García-Sáez, P. Forn-Díaz, and José I. Latorre. One qubit as a universal approximant. *Physical Review A*, 104(1), July 2021.

[166] Qibo Team. Qiboml: Encoder Documentation. https://qibo.science/qiboml/stable/advanced/encoder.html, 2025. Accessed: 2025-09-18.

[167] Qibo Team. Qiboml: Decoder Documentation. https://qibo.science/qiboml/stable/advanced/decoder.html, 2025. Accessed: 2025-09-18.

[168] Tyson Jones and Julien Gacon. Efficient calculation of gradients in classical simulations of variational quantum algorithms, 2020.

[169] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M. Sohaib Alam, Guillermo Alonso-Linaje, B. AkashNarayanan, Ali Asadi, Juan Miguel Arrazola, Utkarsh Azad, Sam Banning, Carsten Blank, Thomas R Bromley, Benjamin A. Cordier, Jack Ceroni, Alain Delgado, Olivia Di Matteo, Amintor Dusko, Tanya Garg, Diego Guala, Anthony Hayes, Ryan Hill, Aroosa Ijaz, Theodor Isacsson, David Ittah, Soran Jahangiri, Prateek Jain, Ed-

ward Jiang, Ankit Khandelwal, Korbinian Kottmann, Robert A. Lang, Christina Lee, Thomas Loke, Angus Lowe, Keri McKiernan, Johannes Jakob Meyer, J. A. Montañez-Barrera, Romain Moyard, Zeyue Niu, Lee James O'Riordan, Steven Oud, Ashish Panigrahi, Chae-Yeun Park, Daniel Polatajko, Nicolás Quesada, Chase Roberts, Nahum Sá, Isidor Schoch, Borun Shi, Shuli Shu, Sukin Sim, Arshpreet Singh, Ingrid Strandberg, Jay Soni, Antal Száva, Slimane Thabet, Rodrigo A. Vargas-Hernández, Trevor Vincent, Nicola Vitucci, Maurice Weber, David Wierichs, Roeland Wiersema, Moritz Willmann, Vincent Wong, Shaoming Zhang, and Nathan Killoran. Pennylane: Automatic differentiation of hybrid quantum-classical computations, 2022.

[170] IBM. https://quantum.cloud.ibm.com/docs/en/guides/calibration-jobs, 2025.

[171] Samson Wang, Enrico Fontana, M. Cerezo, Kunal Sharma, Akira Sone, Lukasz Cincio, and Patrick J. Coles. Noise-induced barren plateaus in variational quantum algorithms. *Nature Communications*, 12(1), November 2021.

[172] Piotr Czarnik, Andrew Arrasmith, Patrick J. Coles, and Lukasz Cincio. Error mitigation with clifford quantum-circuit data. *Quantum*, 5:592, November 2021.

[173] Technology Innovation Institute. Technology Innovation Institute (TII). https://www.tii.ae/, 2025. Accessed: 2025-09-18.

[174] QuantWare B.V. QuantWare: Superconducting quantum processors. https://www.quantware.com/, 2025. Accessed: 2025-09-18.

[175] IQM Quantum Computers. IQM: Scalable quantum computers. https://meetiqm.com/, 2025. Accessed: 2025-09-18.

[176] Antonio Francesco Mello, Alessandro Santini, and Mario Collura. Hybrid stabilizer matrix product operator. *Physical Review Letters*, 133(15), October 2024.

[177] Alberto Di Meglio, Karl Jansen, Ivano Tavernelli, Constantia Alexandrou, Srinivasan Arunachalam, Christian W. Bauer, Kerstin Borras, Stefano Carrazza, Arianna Crippa, Vincent Croft, Roland de Putter, Andrea Delgado, Vedran Dunjko, Daniel J. Egger, Elias Fernández-Combarro, Elina Fuchs, Lena Funcke, Daniel González-Cuadra, Michele Grossi, Jad C. Halimeh, Zoë Holmes, Stefan Kühn, Denis Lacroix, Randy Lewis, Donatella Lucchesi, Miriam Lucio Martinez, Federico Meloni, Antonio Mezzacapo, Simone Montangero, Lento Nagano, Vincent R. Pascuzzi, Voica Radescu, Enrique Rico Ortega, Alessandro Roggero, Julian Schuhmacher, Joao Seixas, Pietro Silvi, Panagiotis Spentzouris, Francesco Tacchino, Kristan Temme, Koji Terashi, Jordi Tura, Cenk Tüysüz, Sofia Vallecorsa, Uwe-Jens Wiese, Shinjae Yoo, and Jinglei Zhang. Quantum computing for high-energy physics: State of the art and challenges. *PRX Quantum*, 5(3), August 2024.

[178] Erich Novak and Klaus Ritter. High dimensional integration of smooth functions over cubes. *Numerische Mathematik*, 75:79–97, 10 1996.

[179] Narayan Kovvali. Theory and applications of gaussian quadrature methods. In *Theory and Applications of Gaussian Quadrature Methods*, 2011.

[180] William J. Morokoff and Russel E. Caflisch. Quasi-monte carlo integration. *Journal of Computational Physics*, 122(2):218–230, 1995.

[181] J.-C. Walter and G.T. Barkema. An introduction to monte carlo methods. *Physica A: Statistical Mechanics and its Applications*, 418:78–87, January 2015.

[182] Steffan Lloyd, Rishad A. Irani, and Mojtaba Ahmadi. Using neural networks for fast numerical integration and optimization. *IEEE Access*, 8:84519–84531, 2020.

[183] David B. Lindell, Julien N. P. Martel, and Gordon Wetzstein. Autoint: Automatic integration for fast neural volume rendering, 2021.

[184] Jorge J. Martínez de Lejarza, Michele Grossi, Leandro Cieri, and Germán Rodrigo. Quantum Fourier Iterative Amplitude Estimation. In *2023 International Conference on Quantum Computing and Engineering*. IEEE, 5 2023.

[185] Jorge J. Martínez de Lejarza, Leandro Cieri, Michele Grossi, Sofia Vallecorsa, and Germán Rodrigo. Loop Feynman integration on a quantum computer. *Phys. Rev. D*, 110(7):074031, 2024.

[186] Jorge J. Martínez de Lejarza, David F. Rentería-Estrada, Michele Grossi, and Germán Rodrigo. Quantum integration of decay rates at second order in perturbation theory. *Quantum Sci. Technol.*, 10(2):025026, 2025.

[187] Konstantinos Pyretzidis, Jorge J. Martínez de Lejarza, and Germán Rodrigo. Unlocking Multi-Dimensional Integration with Quantum Adaptive Importance Sampling. 6 2025.

[188] R. Fletcher. *Practical methods of optimization; (2nd ed.)*. Wiley-Interscience, USA, 1987.

[189] Qibo Team. QiNNtegrate: Quantum-inspired neural network integration. https://github.com/qiboteam/QiNNtegrate, 2025. Accessed: 2025-09-18.

[190] Richard D. Ball, Stefano Carrazza, Juan Cruz-Martinez, Luigi Del Debbio, Stefano Forte, Tommaso Giani, Shayan Iranipour, Zahari Kassabov, Jose I. Latorre, Emanuele R. Nocera, Rosalyn L. Pearson, Juan Rojo, Roy Stegeman, Christopher Schwan, Maria Ubiali, Cameron Voisey, and Michael Wilson. An open-source machine learning framework for global analyses of parton distributions, 2021.

[191] Nikolaus Hansen. The cma evolution strategy: A tutorial, 2023.

[192] David Wierichs, Josh Izaac, Cody Wang, and Cedric Yen-Yu Lin. General parameter-shift rules for quantum gradients. *Quantum*, 6:677, March 2022.

[193] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[194] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[195] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. Effect of data encoding on the expressive power of variational quantum-machine-learning models. *Phys. Rev. A*, 103(3):032430, 2021.

[196] A. Yu. Kitaev. Quantum measurements and the abelian stabilizer problem, 1995.

[197] Li Xiaoyue, Matteo Robbiati, Andrea Pasquale, Edoardo Pedicillo, Andrew Wright, Stefano Carrazza, and Marek Gluza. Strategies for optimizing double-bracket quantum algorithms, 2024.

[198] Franz Wegner. Flow-equations for hamiltonians. *Annalen der Physik*, 506(2):77–91, 1994.

[199] Stanisław D. Głazek and Kenneth G. Wilson. Renormalization of hamiltonians. *Phys. Rev. D*, 48:5863–5872, Dec 1993.

[200] Franz Wegner. Flow equations and normal ordering: a survey. *Journal of Physics A: Mathematical and General*, 39(25):8221, jun 2006.

[201] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Sergio Boixo, Michael Broughton, Bob B. Buckley, David A. Buell, Brian Burkett, Nicholas Bushnell, Yu Chen, Zijun Chen, Benjamin Chiaro, Roberto

Collins, William Courtney, Sean Demura, Andrew Dunsworth, Edward Farhi, Austin Fowler, Brooks Foxen, Craig Gidney, Marissa Giustina, Rob Graff, Steve Habegger, Matthew P. Harrigan, Alan Ho, Sabrina Hong, Trent Huang, William J. Huggins, Lev Ioffe, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Cody Jones, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Seon Kim, Paul V. Klimov, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Pavel Laptev, Mike Lindmark, Erik Lucero, Orion Martin, John M. Martinis, Jarrod R. McClean, Matt McEwen, Anthony Megrant, Xiao Mi, Masoud Mohseni, Wojciech Mruczkiewicz, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Hartmut Neven, Murphy Yuezhen Niu, Thomas E. O'Brien, Eric Ostby, Andre Petukhov, Harald Putterman, Chris Quintana, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Doug Strain, Kevin J. Sung, Marco Szalay, Tyler Y. Takeshita, Amit Vainsencher, Theodore White, Nathan Wiebe, Z. Jamie Yao, Ping Yeh, and Adam Zalcman. Hartree-fock on a superconducting qubit quantum computer. *Science*, 369(6507):1084–1089, August 2020.

[202] Sam McArdle, Tyson Jones, Suguru Endo, Ying Li, Simon C. Benjamin, and Xiao Yuan. Variational ansatz-based quantum simulation of imaginary time evolution. *npj Quantum Information*, 5(1), September 2019.

[203] Guang Hao Low and Isaac L. Chuang. Hamiltonian simulation by qubitization. *Quantum*, 3:163, July 2019.

[204] Christopher M. Dawson and Michael A. Nielsen. The solovay-kitaev algorithm, 2005.

[205] Renato M.S. Farias, Thiago O. Maciel, Giancarlo Camilo, Ruge Lin, Sergi Ramos-Calderer, and Leandro Aolita. Quantum encoder for fixed-hamming-weight subspaces. *Phys. Rev. Appl.*, 23:044014, Apr 2025.

[206] El Amine Cherrat, Iordanis Kerenidis, Natansh Mathur, Jonas Landman, Martin Strahm, and Yun Yvonna Li. Quantum vision transformers. *Quantum*, 8:1265, February 2024.

[207] G. Breit. Theory of Isotope Shift. *Rev. Mod. Phys.*, 30:507–516, Apr 1958.

[208] W. H. King. Comments on the Article "Peculiarities of the Isotope Shift in the Samarium Spectrum". *J. Opt. Soc. Am.*, 53(5):638–639, May 1963.

[209] D N Stacey. Isotope shifts and nuclear charge distributions. *Reports on Progress in Physics*, 29(1):171, jan 1966.

[210] K. Heilig and A. Steudel. Changes in mean-square nuclear charge radii from optical isotope shifts. *Atomic Data and Nuclear Data Tables*, 14(5):613–638, 1974. Nuclear Charge and Moment Distributions.

[211] CWP Palmer. Reformulation of the theory of the mass shift. *Journal of Physics B: Atomic and Molecular Physics*, 20(22):5987, 1987.

[212] Meng Wang, W. J. Huang, F. G. Kondev, G. Audi, and S. Naimi. The AME 2020 atomic mass evaluation (II). Tables, graphs and references. *Chin. Phys. C*, 45(3):030003, 2021.

[213] H. E. Haber, Gordon L. Kane, and T. Sterling. The Fermion Mass Scale and Possible Effects of Higgs Bosons on Experimental Observables. *Nucl. Phys. B*, 161:493–532, 1979.

[214] Cédric Delaunay, Roee Ozeri, Gilad Perez, and Yotam Soreq. Probing Atomic Higgs-like Forces at the Precision Frontier. *Phys. Rev. D*, 96(9):093001, 2017.

[215] E.V. Kahl and J.C. Berengut. ambit: A programme for high-precision relativistic atomic structure calculations. *Computer Physics Communications*, 238:232–243, 2019.

[216] Julian C. Berengut et al. Probing New Long-Range Interactions by Isotope Shift Spectroscopy. *Phys. Rev. Lett.*, 120:091801, 2018.

[217] Julian C. Berengut, Cédric Delaunay, Amy Geddes, and Yotam Soreq. Generalized King linearity and new physics searches with isotope shifts. *Phys. Rev. Res.*, 2:043444, 2020.

[218] Cyrille Solaro, Steffen Meyer, Karin Fisher, Julian C. Berengut, Elina Fuchs, and Michael Drewsen. Improved isotope-shift-based bounds on bosons beyond the Standard Model through measurements of the $^2\mathrm{D}_{3/2}-^2\mathrm{D}_{5/2}$ interval in Ca^+. *Phys. Rev. Lett.*, 125(12):123003, 2020. [Erratum: Phys.Rev.Lett. 127, 029901 (2021)].

[219] A R Striganov, V A Katulin, and V V Eliseev. Peculiarities of the isotopic shift in the samarium spectrum. *Optics and Spectroscopy (U.S.S.R.) (English Translation)*, 2, 2 1962.

[220] D.N. Stacey. A note on the interpretation of isotope shifts. *Physics Letters*, 20(6):644–645, 1966.

[221] Ian Counts, Joonseok Hur, Diana P. L. Aude Craik, Honggi Jeon, Calvin Leung, Julian C. Berengut, Amy Geddes, Akio Kawasaki, Wonho Jhe, and Vladan Vuletić. Evidence for Nonlinear Isotope Shift in Yb^+ Search for New Boson. *Phys. Rev. Lett.*, 125(12):123002, 2020.

[222] Joonseok Hur et al. Evidence of Two-Source King Plot Nonlinearity in Spectroscopic Search for New Boson. *Phys. Rev. Lett.*, 128(16):163201, 2022.

[223] N. L. Figueroa, J. C. Berengut, V. A. Dzuba, V. V. Flambaum, D. Budker, and D. Antypas. Precision Determination of Isotope Shifts in Ytterbium and Implications for New Physics. *Phys. Rev. Lett.*, 128(7):073001, 2022.

[224] Koki Ono, Yugo Saito, Taiki Ishiyama, Toshiya Higomoto, Tetsushi Takano, Yosuke Takasu, Yasuhiro Yamamoto, Minoru Tanaka, and Yoshiro Takahashi. Observation of nonlinearity of generalized king plot in the search for new boson. *Phys. Rev. X*, 12:021033, May 2022.

[225] Akio Kawasaki, Takumi Kobayashi, Akiko Nishiyama, Takehiko Tanabe, and Masami Yasuda. Isotope shift analysis with the $4f^{14}6s^2\ {}^1S_0 - 4f^{13}5d6s^2(J=2)$ transition in ytterbium, 2024.

[226] Menno Door et al. Probing New Bosons and Nuclear Structure with Ytterbium Isotope Shifts. *Phys. Rev. Lett.*, 134(6):063002, 2025.

[227] Alexander Wilzewski et al. Nonlinear Calcium King Plot Constrains New Bosons and Nuclear Properties. *Phys. Rev. Lett.*, 134(23):233002, 2025.

[228] B. Ohayon, S. Hofsäss, J. E. Padilla-Castillo, S. C. Wright, G. Meijer, S. Truppe, K. Gibble, and B. K. Sahoo. Isotope shifts in cadmium as a sensitive probe for physics beyond the standard model. *New J. Phys.*, 24(12):123040, 2022.

[229] Simon Hofsäss, J. Eduardo Padilla-Castillo, Sid C. Wright, Sebastian Kray, Russell Thomas, Boris G. Sartakov, Ben Ohayon, Gerard Meijer, and Stefan Truppe. High-resolution isotope-shift spectroscopy of $\mathrm{Cd\,I}$. *Phys. Rev. Res.*, 5(1):013043, 2023.

[230] Anna V. Viatkina, Vladimir A. Yerokhin, and Andrey Surzhykov. Calculation of isotope shifts and King-plot nonlinearities in Ca^+. *Phys. Rev. A*, 108(2):022802, 2023.

[231] Cédric Delaunay, Claudia Frugiuele, Elina Fuchs, and Yotam Soreq. Probing new spin-independent interactions through precision spectroscopy in atoms with few electrons. *Phys. Rev. D*, 96(11):115002, 2017.

[232] Matthew P. A. Jones, Robert M. Potvliege, and Michael Spannowsky. Probing new physics using Rydberg states of atomic hydrogen. *Phys. Rev. Res.*, 2(1):013244, 2020.

[233] Robert M. Potvliege, Adair Nicolson, Matthew P. A. Jones, and Michael Spannowsky. Deuterium spectroscopy for enhanced bounds on physics beyond the standard model. *Phys. Rev. A*, 108(5):052825, 2023.

[234] R. M. Potvliege. Spectroscopy of light atoms and bounds on physics beyond the standard model. 12 2024.

[235] Claudia Frugiuele, Elina Fuchs, Gilad Perez, and Matthias Schlaffer. Constraining New Physics Models with Isotope Shift Spectroscopy. *Phys. Rev. D*, 96(1):015011, 2017.

[236] QTI-TH. Kifit, 2025. GitHub repository.

[237] Henrik Flyvbjerg and H.G. Petersen. Error estimates on averages of correlated data. *The Journal of Chemical Physics*, 91, 07 1989.

[238] Qibo Team. Qiboedu: Educational quantum computing package using qibo. https://github.com/qiboteam/qiboedu, 2023. Accessed: 2025-09-22.

[239] Dorit Aharonov, Wim van Dam, Julia Kempe, Zeph Landau, Seth Lloyd, and Oded Regev. Adiabatic quantum computation is equivalent to standard quantum computation, 2005.

[240] Max Born and Vladimir Fock. Beweis des adiabatensatzes. *Zeitschrift fur Physik*, 51(3-4):165–180, 1928.

[241] D-Wave Quantum Inc. D-wave quantum. https://www.dwavequantum.com/. Accessed: 2025-09-24.

[242] Peter J. M. van Laarhoven and Emile H. L. Aarts. *Simulated annealing*, pages 7–15. Springer Netherlands, Dordrecht, 1987.

[243] Guifré Vidal. Efficient simulation of one-dimensional quantum many-body systems. *Physical Review Letters*, 93(4), July 2004.

[244] Guang Hao Low and Isaac L. Chuang. Optimal hamiltonian simulation by quantum signal processing. *Physical Review Letters*, 118(1), January 2017.

List of Publications

As of 22th September 2025

Refereed publications

1. Masina, I., Lo Presti, G., Robbiati, M., Grossi, M. (2025). *Simulating Bell inequalities with Qibo*. *European Journal of Physics*, 46, 035401. https://iopscience.iop.org/article/10.1088/1361-6404/adcd13

2. Robbiati, M., Cruz-Martinez, J. M., Carrazza, S., (2025). *Determining probability density functions with adiabatic quantum computing*. *Quantum Machine Intelligence*, 7, 5. https://link.springer.com/article/10.1007/s42484-024-00228-2

3. Cruz-Martinez, J. M., Robbiati, M., Carrazza, S., (2024). *Multi-variable integration with a variational quantum circuit*. *Quantum Science and Technology*, 9, 035053. https://iopscience.iop.org/article/10.1088/2058-9565/ad5866

4. Efthymiou, S., Orgaz-Fuertes, A., Carobene, R., Cereijo, J., Pasquale, A., Ramos-Calderer, S., Bordoni, S., Fuentes-Ruiz, D., Candido, A., Pedicillo, E., Robbiati, M., Tan, Y. P., Wilkens, J., Roth, I., Latorre, J. I., & Carrazza, S. (2024). *Qibolab: an open-source hybrid quantum operating system*. *Quantum*, 8, 1247. http://dx.doi.org/10.22331/q-2024-02-12-1247

5. D'Elia, A., Alfakes, B., Alkhazaleh, A., Banchi, L., Beretta, M., Carrazza, S., Chiarello, F., Di Gioacchino, D., Giachero, A., Henrich, F., Piedjou Komnang, A. S., Ligi, C., Maccarrone, G., Macucci, M., Palumbo, E., Pasquale, A., Piersanti, L., Ravaux, F., Rettaroli, A., Robbiati, M., Tocci, S., & Gatti, C. (2024). *Characterization of a Transmon Qubit in a 3D Cavity for Quantum Machine Learning and Photon Counting*. *Applied Sciences*, 14(4), 1478. http://dx.doi.org/10.3390/app14041478

Publications under review

1. Robbiati, M., Papaluca, A., Pasquale, A., Pedicillo e., Farias, M. S. R., Sopena, A., Robbiano, M., Al Rahmani, G., Bordoni S., Candido, A., Laurora, N., Suda Neto, J., Tan, P., Grossi, M., Carrazza, S. (2025). *`Qiboml`: towards the orchestration of quantum-classical machine learning*. https://arxiv.org/abs/2510.11773

2. Fuchs, E., Kirk, F., Mariotti, A., Richter, J., Robbiati, M. (2025). *Towards a Global Search for New Physics with Isotope Shifts*. arXiv preprint. https://arxiv.org/abs/2506.07303

3. Robbiati, M., Sopena, A., Papaluca, A., Carrazza, S. (2025). *Real-time error mitigation for variational optimization on quantum hardware*. arXiv preprint. https://arxiv.org/abs/2311.05680

4. Robbiati, M., Pedicillo, E., Pasquale, A., Li, X., Wright, A., Farias, R. M. S., Giang, K. U., Son, J., Knörzer, J., Goh, S. T., Khoo, J. Y., Ng, N. H. Y., Holmes, Z., Carrazza, S., & Gluza, M. (2024). *Double-bracket quantum algorithms for high-fidelity ground state preparation.* arXiv preprint. https://arxiv.org/abs/2408.03987

5. Pasquale, A., Pedicillo, E., Cereijo, J., Ramos-Calderer, S., Candido, A., Palazzo, G., Carobene, R., Gobbo, M., Efthymiou, S., Tan, Y. P., Roth, I., Robbiati, M., Wilkens, J., Orgaz-Fuertes, A., Fuentes-Ruiz, D., Giachero, A., Brito, F., Latorre, J. I., & Carrazza, S. (2024). *Qibocal: an open-source framework for calibration of self-hosted quantum devices*. arXiv preprint. https://arxiv.org/abs/2410.00101.

Publications in preparation

1. Crognaletti, G., Grossi, M., Robbiati, M. (2025). *`mpstab`: a cutting-edge quantum circuit simulator*.

Publications in conference proceedings

1. Xiaoyue, L., Robbiati, M., Pasquale, A., Pedicillo, E., Wright, A., Carrazza, S., & Gluza, M. (2024). *Strategies for optimizing double-bracket quantum algorithms.* https://arxiv.org/abs/2408.07431

2. Pasquale, A., Papaluca, A., Farias, R. M. S., Robbiati, M., Pedicillo, E., & Carrazza, S. (2024). *Beyond full statevector simulation with Qibo.* https://arxiv.org/abs/2408.00384

3. Pedicillo, E., Candido, A., Efthymiou, S., Sargsyan, H., Tan, Y. P., Cereijo, J., Khoo, J. Y., Pasquale, A., Robbiati, M., & Carrazza, S. (2024). arXiv preprint. *An open-source framework for quantum hardware control*. https://arxiv.org/abs/2407.21737

4. Robbiati, M., Efthymiou, S., Pasquale, A., & Carrazza, S. (2022). *A quantum analytical Adam descent through parameter shift rule using Qibo*. https://arxiv.org/abs/2210.10787