THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Optimizing the quantum stack: a machine learning approach

DAVID FITZEK

Department of Microtechnology and Nanoscience (MC2)
Applied Quantum Physics Laboratory
Chalmers University of Technology
Göteborg, Sweden, 2024

Optimizing the quantum stack: a machine learning approach
DAVID FITZEK

Technical Report 5566
ISSN 0346-718X

Applied Quantum Physics Laboratory
Department of Microtechnology and Nanoscience (MC2)
Chalmers University of Technology
SE–412 96 Göteborg, Sweden
Phone +46 (0)31-772 1000

Chalmers Digitaltryck
Göteborg, Sweden, 2024

Optimizing the quantum stack: a machine learning approach
DAVID FITZEK
Department of Microtechnology and Nanoscience (MC2)
Applied Quantum Physics Laboratory
Chalmers University of Technology

Abstract

This compilation thesis explores the intersection of machine learning and quantum computing, focusing on optimizing quantum systems and exploring use-cases for quantum computers. Motivated by the potential impact of quantum computers, we investigate several key areas.

First, we delve into machine learning and optimization techniques, establishing the foundation for our research. We then explore reinforcement learning, enabling machines to learn through interaction. Building on these concepts, we investigate variational quantum algorithms as a promising framework for near-term quantum computing. We analyze the quantum approximate optimization algorithm and introduce gradient-based optimization techniques for VQAs, aiming to assess the potential of quantum computing in solving real-world challenges. Next, we focus on quantum circuit optimization, proposing methods to combine machine learning techniques with the qubit allocation and routing problem. This work aims to narrow the gap between theoretical quantum algorithms and their practical implementation on quantum hardware. Finally, we focus on quantum error correction, developing a reinforcement learning approach to efficiently decode error syndromes. This demonstrates the potential of machine learning in enhancing the reliability of quantum computations.

Throughout the thesis, we highlight the benefits of using classical machine learning methods to optimize processes on a quantum computer, contributing to the advancement of quantum computing technologies and their practical applications.

Keywords: Quantum computing, variational quantum algorithm, quantum circuit optimization, machine learning, reinforcement learning, combinatorial op-

timization, quantum information, quantum machine learning, generative neural networks, optimization

Publications

A **Deep Q-learning decoder for depolarizing noise on the toric code**
David Fitzek, Mattias Eliasson, Anton Frisk Kockum, and Mats Granath
Physical Review Research **2**, 023230 (2020)

B **Applying quantum approximate optimization to the heterogeneous vehicle routing problem**
David Fitzek, Toheed Ghandriz, Leo Laine, Mats Granath, and Anton Frisk Kockum
arXiv:2110.06799 (2021)

C **Optimizing Variational Quantum Algorithms with qBang: Efficiently Interweaving Metric and Momentum to Navigate Flat Energy Landscapes**
David Fitzek, Robert S. Jonsson, Werner Dobrautz, and Christian Schäfer
Quantum 8, 1313 (2024)

D **RydbergGPT**
David Fitzek, Yi Hong Teoh, Hin Pok Fung, Gebremedhin A. Dagnew, Ejaaz Merali, M. Schuyler Moss, Benjamin MacLellan, and Roger G. Melko
arXiv:2405.21052 (2024)

Other publications

E

Inter-case Predictive Process Monitoring: A candidate for Quantum Machine Learning?
Stefan Hill, David Fitzek, Patrick Delfmann, and Carl Corea
arXiv:2307.00080 (2023)

Acknowledgments

This doctoral journey has been a profound experience, one that has shaped me both as a researcher and as an individual. This thesis stands as a testament not only to my efforts but also to the invaluable support and guidance of many.

First and foremost, I extend my deepest gratitude to my supervisors, Anton and Mats. Your unwavering support, insightful guidance, and trust in my abilities have been instrumental in my growth as a researcher. Thank you for providing me with the perfect balance of freedom to explore and direction when needed. I also want to acknowledge Göran Johansson, whose valuable insights and discussions greatly contributed to my research journey.

I am particularly grateful to Leo Laine for his supervision at Volvo Group. I also extend my thanks to colleagues Toheed Ghandriz, Henrik Kaijser, Petter Wirfält, Fredrik Sandblom, Jenny Erneman, Thierry Hours, and Parthasarathy Dhasarathy for many meaningful discussions and support.

The collaborative environment at MC2 has been crucial to my progress. Furthermore, a special thanks goes to the PIQuIL group at the Perimeter Institute, particularly Roger Melko. The opportunity to collaborate with your team has been invaluable, broadening my perspectives. Your expertise and insights have greatly enriched this work.

My deepest appreciation goes to my parents, whose unwavering support and encouragement have been my foundation. This gratitude extends to my entire family, whose support has been a constant source of strength, most notably Florian and Meghana.

This journey has been as much about personal growth as it has been about scientific discovery. I am deeply thankful for all the experiences, challenges, and opportunities that have shaped me into the researcher I am today. As I look forward to future endeavors, I carry with me the invaluable lessons, skills, and relationships forged during this remarkable chapter of my life.

David Fitzek, Göteborg, September 2024

Contents

Abstract i

Publications iii

Acknowledgments vii

Contents xi

List of Figures xiii

List of Acronyms xv

1 Introduction 1
- 1.1 Quantum information and computing . . . 2
- 1.2 Quantum computing stack . . . 3
- 1.3 Differentiable programming . . . 5
- 1.4 Variational quantum agorithms . . . 6
- 1.5 Thesis outline . . . 7

2 Machine learning and optimization 9
- 2.1 Basics of machine learning . . . 10
  - 2.1.1 Typical learning tasks . . . 10
  - 2.1.2 Machine learning paradigms . . . 11
- 2.2 Machine learning models . . . 12
  - 2.2.1 Neural networks . . . 12
  - 2.2.2 Autoregressive neural networks . . . 14
  - 2.2.3 Graph neural networks . . . 18
- 2.3 Neural quantum states . . . 19
  - 2.3.1 Autoregressive neural quantum states . . . 20
- 2.4 Optimization with gradients . . . 20

2.4.1 Improving gradient descent . . . 21
2.4.2 Automatic differentiation . . . 22

3 Reinforcement Learning 27
3.1 Basics of reinforcement learning . . . 28
3.2 Markov decision process . . . 29
3.2.1 Bellman equations . . . 30
3.2.2 Bellman optimality equations . . . 32
3.3 Value-based methods . . . 33
3.3.1 Q-learning . . . 34
3.4 Planning and learning . . . 36
3.4.1 AlphaZero . . . 37

4 Quantum machine learning 41
4.1 Variational quantum algorithms . . . 41
4.1.1 Quantum approximate optimization algorithm . . . 43
4.2 Optimizing variational parameters . . . 45
4.2.1 Metric-informed optimization . . . 46
4.3 The Ising model . . . 48
4.3.1 A model to describe magnetism . . . 48
4.3.2 Ising model for some NP-complete problems . . . 49

5 Quantum circuit optimization 57
5.1 Quantum circuit optimization pipeline . . . 58
5.2 Quantum circuit representation . . . 59
5.2.1 Standard circuit diagram . . . 59
5.2.2 Directed acyclic graph (DAG) . . . 59
5.2.3 Hardware representation . . . 61
5.3 Qubit allocation . . . 62
5.4 Qubit routing . . . 63

6 Quantum error correction 65
6.1 The three-qubit bit-flip code . . . 66
6.2 Stabilisers . . . 67
6.3 Surface codes . . . 68
6.3.1 Toric code . . . 68

7 Paper overview 71
7.1 Paper A: Deep Q-learning decoder for depolarizing noise on the toric code . . . 71
7.2 Paper B: Applying quantum approximate optimization to the heterogeneous vehicle routing problem . . . 72
7.3 Paper C: Optimizing variational quantum algorithms with qBang . 73
7.4 Paper D: RydbergGPT . . . 73

8 Conclusion **75**

Bibliography **77**

List of Figures

1.1 Bloch-sphere representation of a qubit state. 3
1.2 The quantum computing stack. 4
1.3 Traditional programming versus machine learning. 6
1.4 Schematic description of a variational quantum algorithm (VQA). 7

2.1 A fully connected neural network (NN). 13
2.2 The transformer encoder-decoder architecture. 16
2.3 Computational graph of the forward and backpard pass. 23

3.1 Overview of the reinforcement learning setting. 29
3.2 A backup diagram rooted at the state, s, and considering each possible action, a. 32
3.3 Backup diagram for the optimal state-value function. 33
3.4 Monte Carlo tree search in AlphaZero. 38
3.5 Overview of the AlphaZero self-play framework. 39

4.1 Main components of variational quantum algorithms. 42
4.2 A schematic description of the quantum approximate optimization algorithm. 43
4.3 Visualization of the decision variables y_{ij}^v in the Ising formulation of the routing problem. 54

5.1 Quantum circuit diagram. 60
5.2 Illustration of quantum circuit equivalence. 60
5.3 Different representations and properties of the coupling map. . . . 61
5.4 The circuit for the quantum teleportation protocol. 62
5.5 Two possible qubit allocations. 63

6.1 The three-qubit bit-flip error-correction code. 67

6.2 A $d = 9$ toric code showing the basic operations. 69

Acronyms

AD automatic differentiation.

AI artificial intelligence.

DAG directed acyclic graph.

DL deep learning.

DP differentiable programming.

DQN deep Q-network.

DRL deep reinforcement learning.

GCN graph convolutional network.

GD gradient descent.

GNN graph neural network.

GPU graphics processing unit.

HVRP heterogeneous vehicle routing problem.

ITE imaginary time evolution.

MCMC Markov chain Monte Carlo.

MCTS Monte Carlo tree search.

MDP Markov decision process.

ML machine learning.

MWPM minimum-weight-perfect-matching.

NISQ noisy intermediate-scale quantum.

NN neural network.

NQS neural quantum state.

PQC parametrized quantum circuit.

QAOA quantum approximate optimization algorithm.

qBang quantum Broyden adaptive natural gradient.

QCO quantum circuit optimization.

QEC quantum error correction.

QFIM quantum Fisher information matrix.

QIR quantum intermediate representation.

QITE quantum imaginary time evolution.

QML quantum machine learning.

QNG quantum natural gradient.

RL reinforcement learning.

RNN recurrent neural network.

SGD stochastic gradient descent.

TD temporal difference.

TSP travelling salesperson problem.

VQA variational quantum algorithm.

VRP vehicle routing problem.

Chapter 1

Introduction

The dawn of the 20th century marked the beginning of a scientific revolution stemming from a series of issues in physics. The established physical theories, now known as classical physics, suggested predictions that contradicted empirical observations and logical reasoning. Two examples of these inconsistencies were the "ultraviolet catastrophe", implying the existence of systems with infinite energy, a clear physical impossibility, and the unstable atomic model, suggesting that electrons would inevitably spiral towards the atomic nucleus. Scientists tried to resolve these issues by extending classical theories. However, these explanations became more and more convoluted as our understanding of atomic structures and radiation advanced. The crisis reached its apex in the early 1920s, resulting in the formulation of quantum mechanics. Since its inception, quantum mechanics has been widely used in science, with applications across many domains, including the structure of the atom [1], nuclear fusion [2], superconductors [3], elementary particles of nature [4], and last but not least as a new paradigm for computation, known as quantum computing [5–7].

The theory of quantum computing emerged in the 1980s when physicists began to discuss paradigms of computation that integrated quantum mechanics. Initially proposed by Feynman [8] as a means of simulating quantum mechanical systems [8–14], the field has since expanded to address a broader range of computational challenges [15–18]. However, the theoretical framework for quantum computing was only solidified by the seminal contributions of Benioff [5] and Deutsch [6], who introduced quantum Turing machines and universal quantum computation. Building upon this foundation, researchers discovered numerous quantum algorithms, with Peter Shor's algorithm for integer factorization [19] in 1994 marking a pivotal moment. Arguably, it was Shor's work that sparked the quantum computing revolution, as it exposed vulnerabilities in many classical cryptographic protocols previously considered secure [20, 21]. A new wave of

physicists and computer scientists entered the field in the hope of finding new quantum algorithms challenging today's classical computers [7]. From then on, quantum algorithms [15–18] have matured into a sophisticated subfield of quantum computing, with applications including machine learning [22–30], simulation of quantum systems [9, 31, 32], natural language processing [33–36], cryptography [19, 37], and search and optimization [22, 38–42].

In the following part of the introduction, I survey the quantum computing stack, and differentiate it from the classical computing stack. In addition, I introduce the paradigm of differential programming and explain the similarities with variational quantum algorithms. The introduction ends with an overview of the thesis.

1.1 Quantum information and computing

The compute stacks of classical and quantum machines share similarities, but differ in some fundamental aspects, notably their fundamental unit of information processing. While classical computers use bits with discrete values (0 or 1), quantum computers use analog quantum bits (qubits). A qubit state is represented by a vector in a complex Hilbert space, spanned by basis vectors $|0\rangle = \begin{bmatrix}1 & 0\end{bmatrix}^\intercal$ and $|1\rangle = \begin{bmatrix}0 & 1\end{bmatrix}^\intercal$. Any qubit state can be written as a superposition of these basis vectors:

$$|\psi\rangle = c_0\,|0\rangle + c_1\,|1\rangle\,; c_i \in \mathbb{C}. \tag{1.1}$$

The complex amplitudes c_i determine the probability of measuring each basis state, following the Born rule: $p_{|i\rangle} = |\langle i|\psi\rangle|^2 = |c_i|^2$. The probability interpretation of quantum mechanics necessitates that the total probability must sum to one, leading to the condition $|c_0|^2 + |c_1|^2 = 1$. This state can be alternatively expressed in polar coordinates, commonly visualized on the Bloch sphere

$$|\psi\rangle = \cos(\theta/2)\,|0\rangle + e^{i\varphi}\sin(\theta/2)\,|1\rangle\,. \tag{1.2}$$

Information in a qubit state is in this representation encoded by two continuous parameters: the polar angle θ, corresponding to population probabilities, and the azimuthal angle φ, representing the phase [Fig. 1.1]. However, measurement of a qubit always yields a discrete result, either $|0\rangle$ or $|1\rangle$, preserving its digital nature. As a result, obtaining a qubit's full state representation requires multiple iterations of state preparation followed by measurement.

Multi-qubit gates are necessary to leverage the full power of a quantum computer. To that end, consider a system of N qubits. To encode such a quantum state using classical methods, one would need in the general case to program 2^N complex amplitudes. The general form of any state in this high-dimensional space can be written as

$$|\psi_N\rangle = \sum_{i=0}^{2^N-1} c_i|i\rangle. \tag{1.3}$$

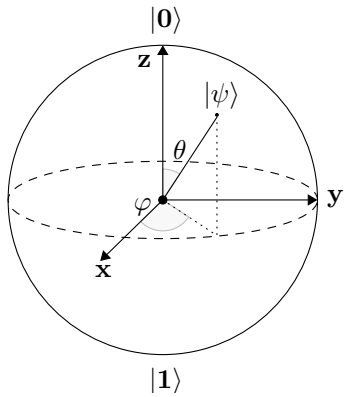


Figure 1.1: Bloch-sphere representation of a qubit state. This representation maps the quantum state to a point on the sphere's surface, with operations on the qubit corresponding to rotations of this point on the sphere.

Here, the summation index i corresponds to the decimal representation of each possible qubit state.

We can categorize these states into two fundamental classes: separable and entangled. Separable states can be expressed as a tensor product of individual qubit states

$$|\Psi\rangle = \bigotimes_i |\psi_i\rangle = |\psi_0\rangle \otimes |\psi_1\rangle \otimes \cdots \otimes |\psi_{N-1}\rangle. \tag{1.4}$$

Conversely, entangled states cannot be decomposed into localized Hilbert spaces. The Bell states are a famous example of entanglement:

$$|\Phi_\pm\rangle = \frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle) \tag{1.5}$$

$$|\Psi_\pm\rangle = \frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle). \tag{1.6}$$

These states are entangled because they cannot be expressed as tensor products of individual qubit states, i.e., there exist no single-qubit states $|\psi_1\rangle$ and $|\psi_2\rangle$ such that $|\Phi_\pm\rangle$ or $|\Psi_\pm\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$. Entanglement, in particular, uniquely separates quantum information from classical information. This property is instrumental in many quantum algorithms [6, 19, 43–45].

1.2 Quantum computing stack

As alluded in the epigraph of this chapter, quantum computation promises to run many interesting algorithms; however, running these requires building quantum

computers from the ground up. Similarly to classical computers, it is not sufficient to simply build the transistor. Several layers of abstraction, working closely with each other, are necessary to make these machines work. We call these layers of abstraction the "computing stack".

Conceptually, a computing stack is a description of different layers of a computing system. The computer stack ranges from the user interface enabling the user to interact with the system, to the control and instruction layers used to control and program the computer, to the physical implementation of the compute units. In quantum computing, the computing stack is analogous to its classical counterpart. The quantum computing stack describes the different layers of a quantum computing system, from the application layer, which includes high-level programming languages and algorithms used to program the quantum computer [Fig. 1.2], to the physical and logical systems that control and manipulate the qubits, and all the way to the elementary particles used to store and compute information (the qubits). These layers are examined in more detail below.

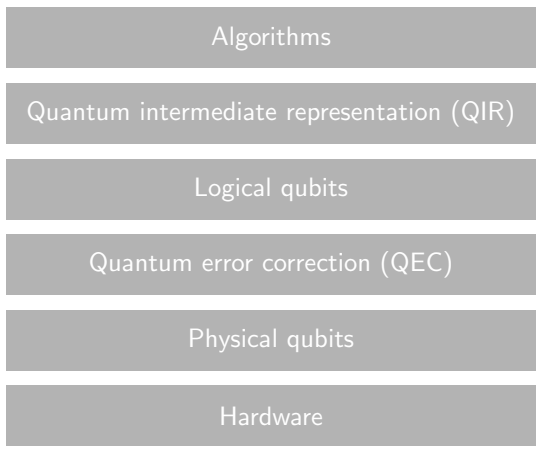


Figure 1.2: The quantum computing stack.

1. **Algorithm:** The topmost layer is the algorithm layer, which is tightly coupled to a specific application. Ideally, the machine solves a computational task that classical machines cannot do in reasonable time. The most famous example being Shor's factoring algorithm, rendering today's cryptographic protocols insecure [19]. Today, more algorithms are known that could provide advantages over classical machines [10, 16–19, 43]. We describe the paradigm of VQAs [22] in more detail in Chapter 4.

2. **Quantum intermediate representation (QIR):** In classical computing, intermediate representations serve as a bridge between source code and machine language. Analogously, QIR function as an interface in quantum com-

puting, enabling the translation of quantum algorithms into a format suitable for implementation on physical quantum devices. QIRs are typically designed with three key characteristics: compiler compatibility, hardware independence, and standardization. These features enable QIRs to effectively mediate between high-level quantum algorithms and different quantum hardware platforms.

3. **Logical qubit:** This is an error-protected quantum state encoded across multiple physical qubits, offering better robustness and reliability in the representation of quantum information. The implementation of logical qubits relies on quantum error correction (QEC) codes [Chapter 6] and additional techniques, which collectively mitigate errors and extend coherence time.

4. **Physical qubit:** As described earlier, these are the fundamental building blocks of a quantum computer. Physical qubits are inherently susceptible to errors and decoherence, which can cause quantum information to be lost or corrupted.

5. **Hardware:** The realization of physical qubits can take many forms [46, 47], such as neutral atoms, superconducting qubits, trapped ions, or photonic circuits.

All the papers included in this thesis are connected to one or more layers of the quantum computing stack. Paper A aims at the QEC layer, providing a machine learning (ML)-driven decoder for an error-correction algorithm. Paper B and Paper C both contribute to the algorithm layer, by adding a new use case and novel gradient-based optimization techniques, respectively. In Paper D, we attempt to learn the ground state properties of a specific quantum computing hardware architecture, namely Rydberg atom arrays. Our final contribution targets the QIR layer, leveraging an AlphaZero-like approach [48] to route quantum circuits, bridging the gap between abstract algorithms and hardware implementation.

1.3 Differentiable programming

As we have seen in Section 1.2, the quantum computing stack covers a wide range of tasks, which formulated as optimization problems can tap into the tool box of differentiable programming (DP). Essentially, DP enables us to build computer programs consisting of parameterized and adjustable elements of code, instead of programs where each instruction is explicitly defined to create a specific point in the program space [Fig. 1.3]. In this paradigm, programmers define the desired program behavior via a loss function. The optimal program is then identified by exploring the program space, adjusting the code parameters to minimize the specified loss function using gradient information. The underlying mechanics of DP are detailed in Chapter 2.

In most real-world applications, collecting data representative of correctly solved tasks is easier to obtain than writing an explicit program that solves the task. Those are the scenarios in which DP excels, as it enables the derivation of task-solving programs directly from data. There are numerous examples, such as face recognition [49, 50], automated driving [51, 52], natural language processing [53, 54], and playing games [55], such as chess or go [48, 56], where this paradigm achieved remarkable results.

Recently, DP has also found its way into scientific computing [57]. Algorithms such as Fourier transforms, eigenvalue solvers, singular value decompositions, or ordinary differential equations [58, 59] have been written in a fully differentiable manner. In addition, the ability to differentiate through domain-specific computational processes to solve inverse problems, such as learning or control tasks, has shown great potential [60]. Fully differentiable implementations now exist for tensor networks [61, 62], molecular dynamics [63, 64], quantum chemistry [65–68], quantum optimal control [69–74], and quantum circuits [75, 76]. Papers A, C, and D tap into this toolbox of DP, allowing us to automatically optimize the predefined task at hand.

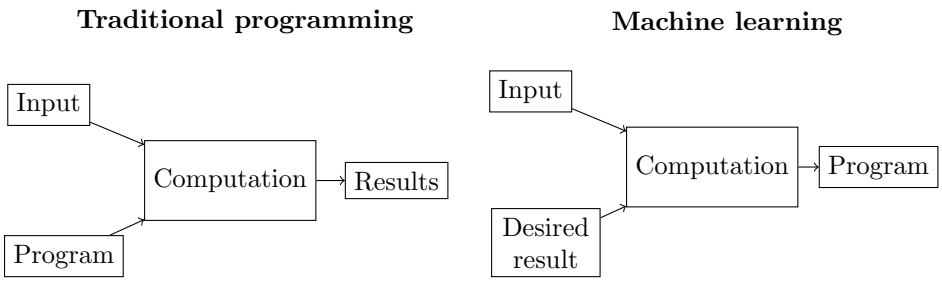


Figure 1.3: Illustration showing the difference between traditional programming, based on the algorithmic approach (left), and the experience-based/data-driven approach (right), fundamental to the ML paradigm. The ML paradigm represents the first step towards teaching computers to learn abstractions by identifying common features within data.

1.4 Variational quantum agorithms

Algorithms derived from the DP paradigm share many similarities with VQAs. Both can be reframed as optimization problems. The core structure of VQAs mirrors that of DP: a cost function defines the problem, a parametrized quantum circuit (PQC) provides the parametrized model, and a hybrid quantum-classical optimization loop fine-tunes the parameters $\boldsymbol{\theta}$ [Fig. 1.4] to solve the optimization

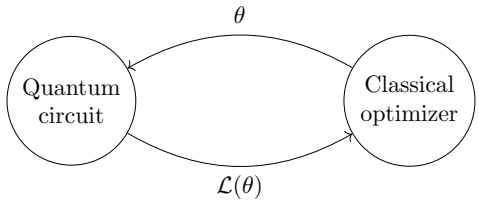


Figure 1.4: Schematic description of a VQA. The algorithm aims to minimize the cost function $\mathcal{L}(\boldsymbol{\theta})$, which represents the energy of the quantum state generated by the parameterized circuit, through iterative optimization of $\boldsymbol{\theta}$.

task

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\ \mathcal{L}(\boldsymbol{\theta})\,. \tag{1.7}$$

This class of algorithms leverages classical optimization techniques, running only the parameterized quantum circuit on the quantum computer, while outsourcing parameter optimization to a classical optimizer. The process involves iteratively minimizing the energy $\mathcal{L}(\boldsymbol{\theta})$ of the quantum state prepared by the variational circuit, by adaptively tuning the circuit parameters $\boldsymbol{\theta}$. Like DP, VQA found many potential applications, such as in chemistry [77–79], optimization [38–42], and machine learning [24, 27, 28, 30, 80], making it a promising avenue for quantum algorithms.

Despite their potential, VQAs face several challenges, particularly in trainability and accuracy. A critical issue is the efficient optimization of the variational parameters in the quantum circuit, which has been shown to be NP-hard [81]. However, this should not be discouraging, as NNs once faced similar challenges before the advent of backpropagation [82] and potent hardware. These historical parallels suggest that overcoming the current limitations of VQAs may be possible with continued research and technological advancements. In fact, in Paper C, we propose a new optimizer specifically designed to address these optimization challenges in VQAs. Furthermore, in Paper B, we explore the application of VQAs to a specific optimization problem.

1.5 Thesis outline

This is a compilation thesis of the appended papers, discussing aspects ranging from quantum computing to ML. We present several classical and quantum algorithms that tackle different challenges around the quantum computing stack. The thesis is organized as follows.

Chapters 2 and 3 are devoted to the broad field of learning machines. This field is fundamental to many of the papers appended, namely, Papers A, C and D. First, we review the different learning tasks in ML and describe the different NN

architectures used throughout the appended papers. We also introduce neural quantum states (NQSs), which leverage this machinery to learn representations of quantum states, a key concept utilized in Paper D. We describe how automatic differentiation (AD) allows us to estimate the gradients of arbitrary computer programs, which is at the heart of modern deep learning (DL). Further, we introduce in Chapter 3 the paradigm of learning from interaction, e.g., reinforcement learning (RL). We lay the groundwork underlying all RL algorithms, namely the Markov property, derive the Bellman optimality equations and further introduce value-based methods and connect RL to ML, resulting in deep RL. The framework of learning from interaction plays a key role in Paper A, where we use deep Q-learning to train an NN to act as a decoder for the toric code.

In Chapter 4, we move the concepts previously introduced to the quantum computer. We describe the class of VQAs as an analogous concept to NNs and introduce the concept of gradient-based optimization for quantum computers. This chapter discusses ideas such as the parameter-shift rule, enabling gradient-based optimization and higher-order optimization methods for quantum computers, which is developed further in Paper C. Furthermore, we discuss the Ising model and demonstrate how certain optimization problems can be mapped onto it, facilitating their integration into the VQA framework, which is used in Paper B.

Chapters 5 and 6 both provide the playground for our RL agent. Chapter 5 covers the different challenges comprising quantum circuit optimization (QCO). We outline the different steps necessary to transpile a quantum circuit to its target hardware and describe the circuit routing and allocation problem, which is the environment used in our final research project. Chapter 6 introduces concepts for QEC. We describe the need for QEC, introduce the concept of repetition codes and from there the stabiliser formalism. Once that is established, we describe the toric code, the environment used to train the RL agent in Paper A.

Finally, in Chapters 7 and 8 we conclude by summarizing our work and looking to the future. There are indeed many interesting directions to pursue in the intersection of quantum computing and machine learning. Especially with the rise of large language models, the time is ripe to unleash these machines to help us build better quantum machines.

Chapter 2

Machine learning and optimization

Building upon the concept of DP introduced in Section 1.3, we now explore its principles and applications. This chapter explores the computational methodology that enables programs to learn and adapt through gradients. We examine how DP allows for the optimization of complex systems by leveraging automatic differentiation and gradient-based techniques. This chapter is based on the ideas outlined in Dawid *et al.* [60].

Creating intelligent machines has been a long-standing goal of human civilization. The aim is to build machines with the ability to conceptualize and create abstractions, which are fundamental mechanisms underlying human learning and reasoning. Conceptualization and abstraction enable us to consider various levels of detail within a representation or switch between levels while preserving relevant information. Representations can vary; we can encode music digitally, on vinyl, or in a music score, yet the core concept remains unchanged, indicating that abstract ideas are independent of their data sources. Our brains excel at extracting abstract ideas from various representations of knowledge, allowing us to process information from multiple sources describing the same concept differently. This ability to reason and connect high-level ideas forms what we call intelligence. Delegating these properties to a computer would create a general problem-solving machine.

Currently, human brains and computers excel at different sets of tasks. Computers outperform humans in solving problems defined by formal mathematical rules, such as logic, algebra, geometry, and optimization, which can be addressed with hard-coded solutions or knowledge-based artificial intelligence (AI). However, problems that are not easily formalized mathematically, such as face recognition [49, 50] or the detection of new quantum phases [83–86], pose challenges. A

promising direction to overcome these challenges is the development of algorithms that learn from experience or data, leading to the rise of ML, shifting from traditional programming to a data-driven paradigm, as illustrated in Fig. 1.3. DL, a subset of ML, operates large hierarchical models to analyze complex patterns in real-world data [60].

This chapter is laying the groundwork to explain how computers can be taught from real-world data rather than from handcrafted rules. To enable a computer to learn, we need several components. First of all, a task to solve and data that can be considered as an equivalent of experience. This data can be provided in the form of, for example, an interactive environment or a static dataset. Lastly, a model that learns how to solve the task. To assess a computer's success in learning a task, we establish a performance measure. This can be as simple as comparing the model's predictions to expected answers. The learning process then becomes an iterative procedure, either minimizing model error or maximizing model performance on the given task and dataset. Several of the ideas introduced are key components in the appended Papers.

2.1 Basics of machine learning

We have already discussed the shift from traditional programming, based on the algorithmic approach, to the data-driven programming paradigm (see Fig. 1.3). In this section, we discuss the different learning tasks and learning types that are fundamental to ML.

2.1.1 Typical learning tasks

The first component needed for a computer to learn is the concept of a learning task. The typical ML task involves examining a response variable, $y(x)$, influenced by an explanatory variable, x. There are no inherent restrictions on whether y, x, or both are continuous, discrete, or categorical.

We start by considering regression tasks, which typically assumes a deterministic relationship between two variables, x and y. The goal is to express the output y as a function of the input x. Generally, both variables can be multidimensional. The objective is to find the function f that maps $y = f(x)$ for all tuples (x, y). Practically, we use a finite data set to determine a model that maps each input x to its corresponding target y. The model, often predefined with parameters, is tuned to fit the data.

Classification represents another broad class of tasks, where the objective is to use an algorithm to assign discrete class labels to instances. Unlike regression, where the goal is to optimize a model to establish a mapping from an input vector x to a target y, classification focuses on identifying a representation of distinct classes. The most basic example of this type of task is binary classification, where an algorithm differentiates between two classes, such as true or false. When the

task requires distinguishing between more than two classes, it is referred to as multiclass classification. A classic example of such a task is the classification of handwritten digit images from the MNIST [87] dataset, which covers ten classes, one for each digit from zero to nine.

Both regression and classification tasks require a training dataset containing examples of inputs x along with their corresponding labels y. However, there are tasks that do not require explicit labels. One example is density estimation [88, 89], where the objective is to infer the probability density function of the dataset. This is closely related to generative problems, which aim to produce new data instances that mimic the given input data. The key distinction between these fields is that generative problems do not need explicit knowledge or reconstruction of the underlying data distribution to generate new samples.

In quantum science, two primary learning tasks have emerged. The first being quantum state [88, 90] or process tomography [91], which aims to reconstruct quantum states or processes from experimental data. This technique allows for the estimation of observables not directly accessible through experiments. The second learning task utilizes the ML framework to simulate quantum systems, employing a data-driven [92, 93] or a Hamiltonian-driven [94] optimization approach, or both [95–97]. Notably, the Hamiltonian-driven simulation task does not require training data, making it distinct from traditional ML paradigms. Using a generative model for this task allows for the generation of new samples similar to the target distribution, effectively simulating the quantum system, a key insight utilized in Paper D. We cover the second learning task in more detail in Section 2.3.

This list of tasks is not comprehensive. Other examples that do not fit directly into the categories mentioned above include text translation, anomaly detection, and data denoising, among others [60]. This should give the reader an idea of the diverse set of learning tasks suitable for the ML framework.

2.1.2 Machine learning paradigms

The second key element in learning is data. Task and data are closely related, as certain tasks can only be accomplished with sufficient data, and a richer dataset allows for easier transitions between tasks. In the ML community, data is defined as a dataset $\mathcal{D}$ containing data points x_i, either alone or paired with labels y_i. Data can be viewed as synonymous with experience, generated through repeated interactions with an environment. The type of accessible data defines the learning methods our model can adopt, typically categorized into supervised, unsupervised, and RL (see Chapter 3).

Supervised learning can be viewed as a generalized notion of regression and classification, introduced in Section 2.1.1, and covers ML algorithms that learn from labeled data, i.e., $\mathcal{D} = \{(x_i, y_i)\}$. A key requirement of supervised learning is accurately labeled data, which is often seen as a major drawback since obtaining perfectly matched labels is not always feasible and often requires manual human

annotation.

However, in many real-world scenarios, we encounter datasets without prior information such as labels, i.e., $\mathcal{D} = \{x_i\}$. The scarcity of labeled data can make traditional supervised learning approaches ineffective. In such cases, unsupervised learning techniques offer an alternative approach. These methods can be employed for initial preprocessing tasks, like dimensionality reduction, or for representation learning, such as clustering. Recently, self-supervised learning [98], a subset of unsupervised learning, has received significant attention. Self-supervised learning involves a machine-learning process, where the model trains itself by learning one part of the input from another part of the input. In this process, the unsupervised problem is converted into a supervised one by automatically generating labels. To effectively leverage the vast amounts of unlabeled data, it is crucial to set appropriate learning objectives to derive supervision from the data itself.

Unlike the previously described types of learning, RL typically lacks a predefined dataset. Instead, it involves interacting with an environment to achieve a specific task. This interaction is supported by feedback, which provides information on whether an action was beneficial or harmful in achieving the task. This feedback is important because there is no predefined path to achieve the task, and often, we do not even know the relevant components needed for success [60]. From these interactions, we can derive a dataset. The dataset in RL consists of the states s of the environment visited, actions a taken, and rewards r or penalties received ($\mathcal{D} = \{(s_i, a_i, r_i)\}$). The field of RL introduces a mathematical framework to learn from interaction and we dedicate Chapter 3 to it. Notably, RL serves as the primary driver behind Paper A and our final research project, demonstrating its crucial role in our work.

2.2 Machine learning models

We discussed the different tasks the framework of ML can target. They can be used for image classification, learning the optimal action to play complex games, solve control problems, machine translation tasks, and more. For each problem different model architectures are beneficial. In this section, we review different architectures excelling at different tasks.

2.2.1 Neural networks

In ML, NN represent a wide range of models used for data processing tasks. These models are parameterized functions made up of several simple functions. The initial inspiration for NNs came from the NNs found in our brains [99]. They generally consist of interconnected layers that process information in sequence, as shown in Fig. 2.1. Each layer contains multiple nodes, called artificial neurons or perceptrons. Each node i receives an input vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$, representing the activations of all nodes from the previous layer. It then produces

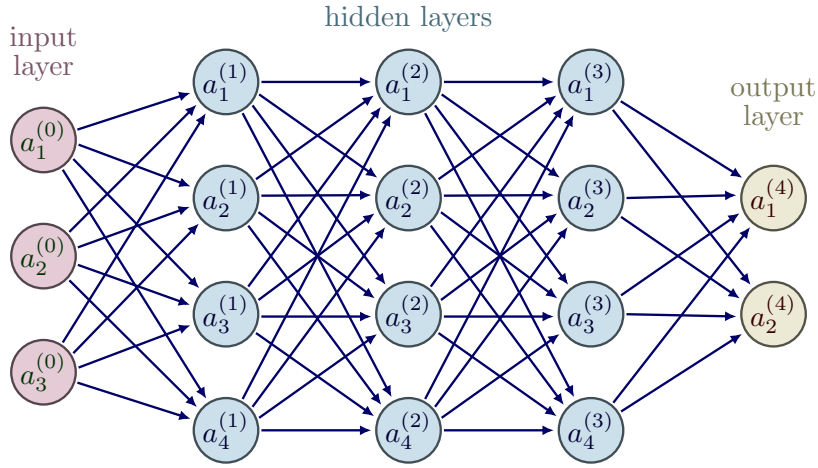


Figure 2.1: Illustration of a fully connected NN. The network consists of three sections: the input layer, which processes the data vector x; followed by n hidden layers (in this case, three); and finally, the output layer. Each layer incorporates a non-linear activation function.

a scalar value $a_i \in \mathbb{R}$ (referred to as its activation), computed as,

$$a_i = \zeta\left(\sum_j w_{ij}x_j + b_i\right), \tag{2.1}$$

where $\{w_{ij}\}_{j=1}^{d}$ and $b_i \in \mathbb{R}$ are the weights and bias of node i, respectively. Typically, we denote the weights and biases as parameters θ. The weights of a node determine the strength of its connection to the neurons in the previous layer. The function ζ is a nonlinear activation function, commonly chosen as the rectified linear unit (ReLU), defined as $\zeta(z) = \max(0, z)$. However, a wide variety of nonlinear functions can be chosen [60, 100, 101].

The first layer is known as the input layer, where node activations are set based on the vector $\boldsymbol{x}$ representing the input data. The final layer is called the output layer, and the activations of its nodes form the output of the NN. All layers in between are known as hidden layers. NNs where each node is by default connected to all nodes in the subsequent layer are called fully connected. The number of layers, nodes, and their connections is known as the architecture of a NN.

The function implemented by a feedforward NN with L layers ($L-1$ hidden layers and one output layer) can be expressed as:

$$\mathrm{NN}(x) = \boldsymbol{a}^{(L)}(x) = \zeta^{(L)}(\boldsymbol{W}^{(L)}\boldsymbol{a}^{(L-1)}(\boldsymbol{x}) + \boldsymbol{b}^{(L)}), \tag{2.2}$$

where $\boldsymbol{a}^{(0)}(\boldsymbol{x}) = \boldsymbol{x}$ represents the input vector. Specifically, the output relies on $\boldsymbol{x}$ exclusively through the activations of the previous layer $\boldsymbol{a}^{(L-1)}$, which in turn depends on $\boldsymbol{x}$ only through $\boldsymbol{a}^{(L-2)}$, and so on. This cascade of dependencies is a direct consequence of the layer-by-layer information processing inherent to feedforward NNs. This hierarchical information processing is a fundamental characteristic of feedforward NNs [60].

The trainable parameters θ of an NN are typically optimized using gradient-based methods to minimize a specified loss function $\mathcal{L}$. The gradient of the loss function with respect to the NN parameters is usually computed numerically through backpropagation, which is discussed in more detail in Section 2.4.

While feedforward NNs process information in a straightforward manner from input to output, certain applications require models that can capture sequential dependencies in data. This need has led to the development of specialized architectures, such as autoregressive NNs, which we will explore in the following section.

2.2.2 Autoregressive neural networks

Autoregressive NNs cover a diverse set of applications, ranging from time-series forecasting [102] to machine translation [103, 104] to a wide range of physics use cases [83, 92, 93, 95–97, 105–115]. These models are structured such that the output adheres to a conditional framework:

$$f(x) = \prod_{i=1}^{m} f_i(x_i \mid x_{i-1}, \ldots, x_1). \tag{2.3}$$

Here, $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$ denotes the inputs. For time-series data, the inputs x_i represent the values of a variable at times t_i, and the model f aims to forecast future values based on past observations. A common example of such networks is the recurrent neural network (RNN) [82], which has gained popularity in natural language processing tasks. The key concept in these models is that information "loops back" into the network, creating correlations between different parts of the model, unlike feedforward NNs [60].

However, the current state-of-the-art model architecture for these tasks is the transformer, which replaces recurrences with the attention mechanism [116–118]. The following section provides a detailed description of the transformer architecture introduced in Vaswani *et al.* [116].

Transformer

The transformer encoder-decoder architecture, central to Paper D, depicted in Fig. 2.2, consists of an encoder composed of multiple identical layers. Each layer comprises two sublayers: a multihead self-attention mechanism followed by a position-wise feedforward NN. The model uses residual connections [119]. More

precisely, for any input $\boldsymbol{x} \in \mathbb{R}^d$ at any position in the sequence, it is necessary that sublayer$(\boldsymbol{x}) \in \mathbb{R}^d$ allows the residual connection $\boldsymbol{x} + \text{sublayer}(\boldsymbol{x}) \in \mathbb{R}^d$. This residual connection is followed by layer normalization [120]. The transformer encoder outputs a d-dimensional vector representation for each position in the input sequence.

The transformer decoder also consists of multiple identical layers, incorporating residual connections [119] and layer normalization [120]. Besides the two sublayers present in the encoder, the decoder includes a third sublayer: the encoder-decoder attention, conditioning the decoder on the encoder. In the encoder-decoder attention mechanism, the queries originate from the decoder's self-attention sublayer outputs, while the keys and values come from the transformer encoder outputs. In the decoder's self-attention, the queries, keys, and values are derived from the previous decoder layer's outputs. Note that each position in the decoder can only attend to all preceding positions, preserving the autoregressive property. This masked attention ensures that the prediction relies solely on the previously generated tokens.

Multi-head attention The self-attention mechanism is one of the key elements of the transformer. The objective is to design an attention mechanism where any element in a sequence can attend to any other element while maintaining computational efficiency. The dot-product attention operates on a set of queries $Q \in \mathbb{R}^{S \times d_k}$, keys $K \in \mathbb{R}^{S \times d_k}$, and values $V \in \mathbb{R}^{S \times d_v}$, where S is the sequence length and d_k and d_v are the hidden dimensions for queries/keys and values, respectively. For simplicity, the batch dimension is omitted here. The attention value from element i to j is determined by the similarity between the query Q_i and the key K_j, using the dot product as the similarity metric. Mathematically, the dot-product attention is computed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.4}$$

The matrix multiplication QK^T computes the dot product for each pair of queries and keys, resulting in an $S \times S$ matrix. Each row represents the attention logits (raw, unnormalized attention scores) for a specific element i relative to all other elements in the sequence. A softmax function is then applied, followed by multiplying with the value vector to obtain a weighted average (with weights determined by the attention). The scaling factor $\frac{1}{\sqrt{d_k}}$ is crucial to maintain an appropriate variance of attention values after initialization.

The scaled dot-product attention mechanism enables a network to attend on different parts of a sequence. However, when multiple aspects of a sequence element need attention, a single weighted average is insufficient. To address this, the attention mechanism is extended to multiple heads, meaning several different query-key-value triplets on the same features. Specifically, given a query, key, and value matrix, we convert these into h sub-queries, sub-keys, and sub-values,

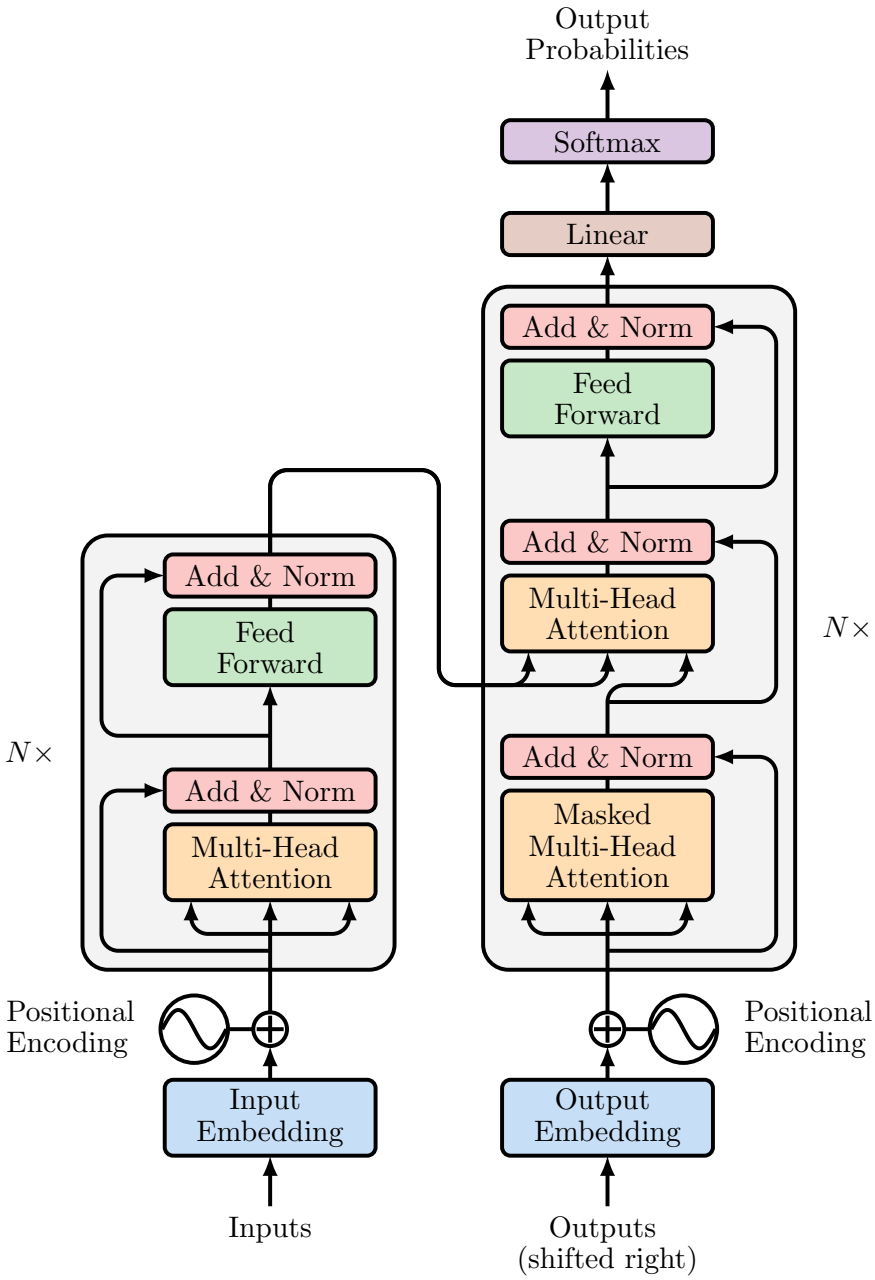


Figure 2.2: The transformer encoder-decoder architecture. The encoder block (left) and decoder block (right) share common features. Adapted from [116].

which are then processed through the scaled dot-product attention independently. Subsequently, we concatenate these heads and combine them using a final weight matrix. This process is mathematically represented as follows:

$$\text{Multihead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{2.5}$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V). \tag{2.6}$$

This is referred to as the multi-head attention layer, with the learnable parameters $W_i^Q \in \mathbb{R}^{d \times d_k}$, $W_i^K \in \mathbb{R}^{d \times d_k}$, $W_i^V \in \mathbb{R}^{d \times d_v}$, and $W^O \in \mathbb{R}^{h \cdot d_v \times d_{out}}$ (remember d is the input dimensionality).

Position-wise feedforward network In addition to the attention sub-layers, each layer of the encoder and decoder contains a fully connected feedforward NN (as introduced in Section 2.2.1), which is applied separately and identically to each position. This NN consists of two linear transformations with a ReLU activation function in between. The feedforward NN is defined as

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2. \tag{2.7}$$

While the linear transformations remain consistent across different positions, they use different parameters from layer to layer. Typically, the dimensionality ratio of input to output is 1:4. For example, if $d = 512$, then the inner layer has a dimensionality of $d_{\text{ff}} = 2048$.

Positional encoding The multi-head attention block cannot distinguish the order of inputs in a sequence. However, in tasks like language understanding, the position of each word is important. To address this, position information is added to the input features. While it is possible to learn an embedding for every possible position, this approach does not generalize well to varying input sequence lengths. A more effective method is to use feature patterns that can be identified by the network from the features and generalize to longer sequences. Vaswani *et al.* [116] chose sine and cosine functions of different frequencies for this purpose, expressed as

$$PE_{\text{pos},i} = \begin{cases} \sin\left(\frac{\text{pos}}{10000^{2i/d}}\right) & \text{if } i \mod 2 = 0 \\ \cos\left(\frac{\text{pos}}{10000^{(2i-1)/d}}\right) & \text{otherwise.} \end{cases} \tag{2.8}$$

Here, $PE_{\text{pos},i}$ represents the positional encoding at position pos and hidden dimension i. These values are added to the input features of each layer (as illustrated in Fig. 2.2 under "Positional Encoding"). We differentiate between even ($i \mod 2 = 0$) and odd ($i \mod 2 = 1$) hidden dimensions, applying sine and cosine functions, respectively. The rationale behind this encoding is that $PE_{(\text{pos}+k,:)}$ can be expressed as a linear function of $PE_{(\text{pos},:)}$, enabling the model to easily attend to relative positions. The wavelengths of the functions span from 2π to $10000 \cdot 2\pi$ [116].

2.2.3 Graph neural networks

So far we have not discussed the different ways in which data is represented. Sometimes it is in text form, sometimes in form of tables, but more often than not the data is graph structured. Take for example a social network, in the academic context we can think of a citation network, the world wide web, proteins, or even quantum circuits. All of these types of data are naturally represented as a graph. Graph neural networks (GNNs) are neural architectures specifically designed for graph-structured data.

Formally, a graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. We denote $A \in \mathbb{R}^{N \times N}$ as the adjacency matrix, with N representing the total node count. The node-attribute matrix is given by $X \in \mathbb{R}^{N \times D}$, where D denotes the number of features for each node. The primary objective of GNNs is to learn effective node representations, denoted as $H \in \mathbb{R}^{N \times F}$ (where F is the dimensionality of the node representations), by synthesizing information from both the graph structure and node attributes. These learned representations serve as a foundation for a wide range of graph-based ML tasks.

The fundamental operation of GNNs relies on the principle of iteratively refining node representations by integrating information from neighboring nodes. This process begins with an initial node representation $H^0 = X$ and proceeds through a series of layers. Each layer executes two key operations: an *aggregation* function, which consolidates information from a node's local neighborhood, and a *combine* function, which updates the node's representation by merging the aggregated neighborhood data with its existing representation. The mathematical description of this framework is as follows:

Algorithm 1 Framework graph neural network

1: Initialization: $H^0 = X$
2: **for** k=1 to K **do**
3: $\quad a_v^k = \text{AGGREGATE}^k\{H_u^{k-1} : u \in N(v)\}$
4: $\quad H_v^k = \text{COMBINE}^k\{H_v^{k-1}, a_v^k\}$
5: **end for**

Here, $N(v)$ represents the set of neighbors for node v. The node representations from the final layer, H^K, are considered the definitive node representations.

These learned representations can be used in various downstream applications. Typical tasks evolve around node edge or graph classification. Once we define the task-specific loss function, we can use the same machinery used for NNs to optimize the GNN [121–123].

Graph convolutional networks

Graph convolutional networks (GCNs) update the node representations according to the equation

$$H^{k+1} = \zeta(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^k W^k). \tag{2.9}$$

Here, $\tilde{A} = A + I$ represents the adjacency matrix of the undirected graph $\mathcal{G}$, including self-connections to incorporate node features during updates. $I \in \mathbb{R}^{N\times N}$ denotes the identity matrix, while $\tilde{D}$ is a diagonal matrix where $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ corresponds to the degree of node i. The function $\zeta(\cdot)$ is an activation function, commonly ReLU or tanh. The trainable linear transformation matrix $W^k \in \mathbb{R}^{F\times F'}$, where F and F' represent the dimensions of node representations in the k-th and $(k+1)$-th layers, respectively [121, 124]. In Paper E, we utilize GCNs to learn a representation of a graph structure that can be used in combination with a transformer architecture (see Section 2.2.2).

Having explored various NN architectures, including those designed for graph-structured data, we now turn our attention to a specialized application of NNs in quantum physics. This brings us to the concept of NQSs, where NNs are used to represent quantum systems.

2.3 Neural quantum states

Let us return to the quantum many-body system introduced in Section 1.1. The state of such a system is described by a wave function that exists in an exponentially large Hilbert space. For a system of N-qubit states, there are 2^N amplitudes to describe the wave function. As the size of the system increases, the number of coefficients required to describe the state exactly becomes increasingly large. For instance, storing the wave function of 60 qubits would require approximately 18 exabytes of memory, far exceeding the capabilities of today's largest supercomputers.

However, physically relevant states often occupy only a small portion of the Hilbert space, typically constrained by local interactions. This observation motivates the use of variational methods to find efficient representations of quantum states [125]. A promising approach in finding these representations is offered by NQSs [125, 126]. By encoding the complex amplitudes of the wave function onto a parametrized function (often an NN), we can represent the quantum states of large systems using a number of parameters that scale polynomially with system size. We can express the variational state in its computational basis as

$$|\psi_{\boldsymbol{\theta}}\rangle = \sum_{\boldsymbol{\sigma}} \psi_{\boldsymbol{\theta}}(\boldsymbol{\sigma})\,|\boldsymbol{\sigma}\rangle \tag{2.10}$$

where $\boldsymbol{\theta}$ represents the parameters of the NN, $|\boldsymbol{\sigma}\rangle = |\sigma_1\rangle \otimes |\sigma_2\rangle \otimes \cdots \otimes |\sigma_N\rangle$ are the basis vectors of the Hilbert space that describes the N-qubit system, and $\psi_{\boldsymbol{\theta}}(\boldsymbol{\sigma}) = \langle\boldsymbol{\sigma}|\,\psi_{\boldsymbol{\theta}}\rangle$ is the probability amplitude for the state $|\boldsymbol{\sigma}\rangle$. The task then

becomes finding the optimal parameters $\boldsymbol{\theta}$ that best describe the quantum state of interest, such as the ground state of a given Hamiltonian [60].

2.3.1 Autoregressive neural quantum states

Using the autoregressive NNs previously discussed in Section 2.2.2, one can construct NQSs. A key advantage of these NQS models is their normalized Born probability distribution, which allows for direct autoregressive sampling. This sampling method is more easily parallelizable compared to Markov chain Monte Carlo (MCMC) techniques [127]. In this framework, we express the many-body wave function as a product of conditional complex amplitudes, similar to Eq. (2.3):

$$\psi_\theta(\boldsymbol{\sigma}) = \prod_{i=1}^{N} \psi_\theta(\sigma_i|\sigma_{i-1}, \ldots, \sigma_1), \tag{2.11}$$

based on the normalization condition $\sum_\sigma |\psi_\theta(\sigma|\sigma_{i-1}, \ldots, \sigma_1)|^2 = 1$. It enables direct sampling of state configurations for computing expectation values, bypassing the need for Markov chain construction via methods such as the Metropolis-Hastings algorithm [127, 128]. Sampling occurs in sequence: beginning with σ_1 drawn from $|\psi_\theta(\sigma_1)|^2$, subsequent spins are sampled from conditional distributions $|\psi_\theta(\sigma_i|\sigma_{i-1}, \ldots, \sigma_1)|^2$ until σ_N is reached [60]. This sampling procedure yields independent, identically distributed samples. Autoregressive NQS played a key role in Paper D.

2.4 Optimization with gradients

Among optimization algorithms, gradient-based methods have gained widespread adoption and success, serving as the foundation for modern DL and numerous scientific disciplines. These methods optimize the parameters θ of a function $\mathcal{L}(\theta)$ through iterative gradient calculations, employing a step size $\eta > 0$:

$$\theta' = \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}. \tag{2.12}$$

The standard gradient-descent algorithm is adaptable to various contexts. In cases where $\mathcal{L}(\theta)$ depends on data, such as the least-squares loss function $\mathcal{L}(\theta) = \sum_i^N ||f(x_i;\theta) - y_i||_2^2$, a single gradient update requires calculations across the entire dataset. Stochastic gradient descent (SGD) offers an alternative by updating parameters based on individual data points (x_i, y_i), while mini-batch gradient descent processes small subsets of n examples. Despite its effectiveness, this approach may face challenges including convergence to local minima, slow convergence rates, strong dependence on learning-rate selection, and issues such as sparse gradients [88, 129].

2.4.1 Improving gradient descent

To overcome some of the challenges associated with standard SGD, several algorithmic improvements have been developed [129–134]. A notable enhancement is the momentum technique, which accelerates SGD by reducing oscillations around local minima and maintaining the trajectory established by previous updates. This is implemented using a velocity vector:

$$r_t = \gamma r_{t-1} + \eta \frac{\partial \mathcal{L}}{\partial \theta}. \tag{2.13}$$

Here, γ is commonly assigned a value of 0.9. The SGD update rule is then adjusted to

$$\theta' = \theta - r_t. \tag{2.14}$$

This modification allows the algorithm to accumulate momentum in consistent directions, potentially leading to faster convergence and improved optimization outcomes.

A commonly used optimization algorithm is the Adam optimizer, which combines the benefits of momentum and adaptive learning rates. Adam maintains a separate learning rate for each parameter, adapting the learning rate based on the first and second moments of the gradients [130]. It maintains moving averages of the gradient m_t and the squared gradient v_t, defined as

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{2.15}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_1) g_t^2. \tag{2.16}$$

Here, $g_t = \nabla_{\theta_t} \mathcal{L}(\theta_t)$ denotes the gradient at step t, with β_1 and β_2 being positive coefficients. To address the bias introduced by initializing m_t and v_t to 0, Adam applies a correction:

$$m_t' = \frac{m_t}{1 - (\beta_1)^t}, \tag{2.17}$$

$$v_t' = \frac{v_t}{1 - (\beta_2)^t}. \tag{2.18}$$

The resulting update rule is

$$\theta' = \theta - \frac{\eta}{\sqrt{v_t'} + \epsilon} m_t', \tag{2.19}$$

where $\epsilon \approx 10^{-8}$ ensures numerical stability. Typically, $\beta_1 = 0.9$ and $\beta_2 = 0.999$, although these values can be fine-tuned through hyperparameter optimization [88, 129, 130].

The effectiveness of gradient-based optimization methods relies on the ability to compute gradients $\nabla_\theta \mathcal{L}(\theta)$ for arbitrary computer programs. The introduction of the backpropagation algorithm [82] marked a significant milestone, enabling

efficient gradient computation in NN architectures. Building on this foundation, the recent emergence of AD tools has further streamlined the implementation of gradient calculations in software, making gradient-based optimization more accessible and widely applicable in computational tasks.

2.4.2 Automatic differentiation

While traditional programming requires explicit coding of every instruction, automatic differentiation (AD) uses parameterized code segments that can be modified (see Fig. 1.3). In this paradigm, programmers define the intended behavior of the program through a loss function. The solution is then identified by exploring the program space, a process that relies on the efficient computation of derivatives.

Derivative computation methods in computer programs can be categorized into four types: manually deriving and coding derivatives, numerical differentiation via finite difference approximations, symbolic differentiation using expression manipulation, and AD. Except for AD, the other methods have some pitfalls that make them infeasible to use: finite difference methods introduce a computational overhead when calculating the gradients, and symbolic differentiation becomes increasingly difficult to calculate when the expressions are complex.

AD enables the calculation of gradients with the same time complexity as computing the function itself. There are two flavors of it, forward-mode and reverse-mode AD. For functions with many inputs (i.e., where $m \gg n$ in $f : \mathbb{R}^m \to \mathbb{R}^n$), the reverse-mode is computationally more efficient. In the case of $f : \mathbb{R}^m \to \mathbb{R}$, a single reverse-mode application is sufficient to calculate the complete gradient, contrasting with m forward-mode passes. Applications in ML typically involve evaluating derivatives of a loss function $y = \mathcal{L} : \mathbb{R}^m \to \mathbb{R}$ with respect to m trainable parameters, where m is usually large. Therefore, reverse-mode AD is the preferred approach for automatic gradient computation due to its computational efficiency compared to forward-mode AD [60].

Reverse-mode AD

Reverse-mode AD is a two-stage process. First, the original function code is executed forward, populating intermediate variables and tracking their dependencies within the computational graph. In the subsequent phase, derivatives are calculated by propagating adjoints in reverse, from the outputs back to the inputs (see Fig. 2.3).

To illustrate the process, consider the function

$$f(x) = \sin\big(2x^2 - 3 + x^2\big). \tag{2.20}$$

The derivative of the trial function can be explicitely calculated as

$$\frac{df}{dx} = \cos\big(2x^2 - 3 + x^2\big) \cdot 6x. \tag{2.21}$$

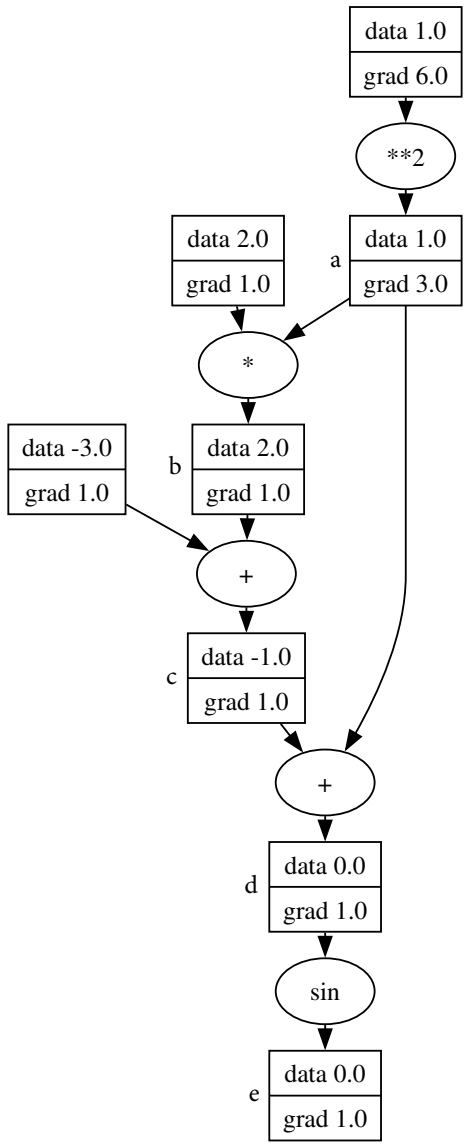


Figure 2.3: Computational graph. Illustrating the forward pass (data) and the backward pass (grad). The AD process contains two steps, the first one being the forward pass that evaluates each intermediate data point. Once each value is computed the gradients can be evaluated during the backward pass.

However, this method becomes challenging when the expressions become large as it is typical for NNs with billions of parameters. Another way of approaching the problem is to define intermediate variables

$$\begin{aligned} a &= x^2, \\ b &= 2 \cdot a, \\ c &= b - 3, \\ d &= c + a, \\ e &= \sin(d). \end{aligned} \tag{2.22}$$

From this set of equations we can write the computational graph and show the relationship between all the variables (see Fig. 2.3). Next, we can manually write down the derivatives of each individual term, and given all of this, we can work backward to calculate the derivative of f with respect to each variable, by applying the chain rule:

$$\begin{aligned} \frac{de}{dd} &= \cos(d), \\ \frac{de}{dc} &= \frac{de}{dd}\frac{dd}{dc} = \frac{de}{dd}, \\ \frac{de}{db} &= \frac{de}{dc}\frac{dc}{db} = \frac{de}{dc}, \\ \frac{de}{da} &= \frac{de}{db}\frac{db}{da} + \frac{de}{dd}\frac{dd}{da} = 2\frac{de}{db} + \frac{de}{dd}, \\ \frac{de}{dx} &= \frac{de}{da}\frac{da}{dx} = 2\frac{de}{da}. \end{aligned} \tag{2.23}$$

AD can be understood as a formalization and generalization of this process. It allows for systematic computation of derivatives for a wide class of functions $f : \mathbb{R}^m \to \mathbb{R}^n$ by representing them as expression graphs. In the forward propagation phase, we compute intermediate values sequentially (see Algorithm 2).

Algorithm 2 Forward propagation

1: **for** i=1 to M **do**
2: $\quad x_i \leftarrow g_i(\boldsymbol{x}_{\mathrm{Pa}(i)})$
3: **end for**

In this formulation, $x_1, ..., x_m$ represent the input variables, $x_{m+1}, ..., x_{M-1}$ denote intermediate computational values, and x_M is the final output of the function. The functions g_i correspond to elementary operations applied to the "parent" variables $\mathrm{Pa}(i)$ of the i-th node in the computational graph (see Fig. 2.3).

Given a function represented as an expression graph, we can systematically apply the chain rule to compute derivatives. By definition, the function output f

is equivalent to x_M, leading to the initial condition

$$\frac{df}{dx_M} = 1. \tag{2.24}$$

For all other intermediate variables x_i, we can express the derivative with respect to x_i as

$$\frac{df}{dx_i} = \sum_{j:i\in \mathrm{Pa}(j)} \frac{df}{dx_j}\frac{dx_j}{dx_i} \tag{2.25}$$

$$= \sum_{j:i\in \mathrm{Pa}(j)} \frac{df}{dx_j}\frac{dg_j}{dx_i}. \tag{2.26}$$

This formulation allows for an efficient backward propagation of derivatives through the computational graph (see Fig. 2.3) [135].

This concludes our general introduction to ML. Next, we dive deeper into the paradigm of learning through interaction, known as RL.

Chapter 3

Reinforcement Learning

Up to this point, we have looked at various ML concepts, such as supervised and unsupervised learning tasks, where goals include predicting labels, estimating specific values, or identifying patterns within data. In this chapter, we explore the computational methodology of learning through interaction. This chapter is based on the ideas outlined in Dawid *et al.* [60] and Sutton and Barto [136].

In the context of supervised learning, one can imagine a student learning from a teacher holding the correct answers to all questions within a specific domain. In this scenario, the student's knowledge is restricted to that of the teacher and cannot exceed it or address questions beyond the teacher's competence. To overcome this limitation, in RL, the teacher is removed, allowing the student to experiment and learn from the resulting experiences. The student is referred to as the agent, as it can take actions independently. Similar to humans, the agent learns through interaction with its environment. The agent receives feedback based on the outcome of its actions, and comes up with strategies to achieve certain objectives.

Consider the example of teaching an agent to play chess. In a supervised learning setting, the model is trained to mimic the moves from games played by expert players. Therefore, the agent's performance is limited by the quality of the training data, making it unlikely for the agent to surpass the skill level of the baseline players. Instead, we can allow the agent to play chess games, either with different opponents or even against itself, without giving any additional knowledge beyond the rules. This results in an agent with far greater potential than the previous one, as it is not constrained by its teacher. However, learning through experience can be difficult, given that the quality of actions is only assessed at the end of the game when the win or loss is determined. Therefore, the agent must establish a broad understanding of the long-term effects of its actions based on the limited feedback from the environment [60].

Describing problems as games to discover strategies has many applications.

In the context of quantum computing, control and optimization problems are a natural fit. However, we can design games to obtain any protocol or algorithm of interest, from new quantum circuit optimization routines [137] to faster matrix multiplication [138] or sorting algorithms [139]. In our work, we leverage these game-based strategies across different layers of the quantum computing stack. Paper A applies this approach to the QEC layer, developing a ML-driven decoder for error-correction algorithms. Additionally, our research extends to the QIR layer, where we employ game-based techniques to optimize the routing of quantum circuits, effectively bridging the gap between abstract algorithms and hardware implementation.

In this chapter, we introduce the field of RL, exploring the concepts of learning from experience and its mathematical foundation. We present the two main paradigms in RL: model-free methods, exemplified by value-based RL, and model-based methods, represented by planning algorithms. This comprehensive coverage allows us to examine both approaches that learn directly from interactions without an explicit model of the environment, and those that leverage a model for planning and decision making.

3.1 Basics of reinforcement learning

The two main elements in the RL framework are an *agent* and an *environment* that it interacts with. The environment consists of all the information that defines the problem and provides the agent with observations and feedback according to its *actions*. The environment spans the set of all possible *states*, $s \in \mathcal{S}$.

The agent can observe (sometimes only partially[1]) the state s of the environment and suggests an action a. The action is chosen from a set of possible actions, $a \in \mathcal{A}$, which is determined by the environment. The actions can alter the state in which the environment is found, and they can have deterministic or stochastic outcomes. Whenever the agent performs an action, the environment provides it with an observation of the new state together with a feedback called the *reward*, r. The reward can take any numerical value. The agent obtains higher rewards when accomplishing the objective task, for example, winning a game, while receiving penalties when performing bad actions, e.g., losing a game. Figure 3.1 summarizes these concepts.

Rewards may only be received at the end of the game. Therefore, we need a method to manage delayed rewards. The concept involves considering future rewards obtained along a trajectory through the state space. However, we can diminish the value of rewards that are far in the future using a discount factor $\gamma \in [0, 1]$. The discount factor adjusts the importance of rewards based on their

[1]Partially observable Markov decision processes (POMDP) are essential in QEC. In this context, only parity measurements are conducted to prevent interference with the computation. While these measurements offer only partial system information, they are adequate for suggesting error correction strategies. Details can be found in Chapter 6.

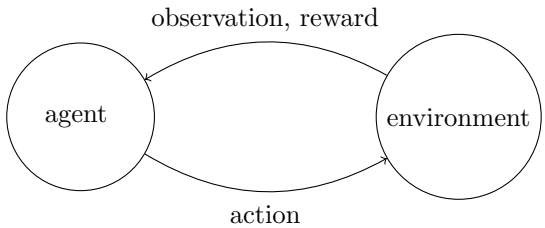


Figure 3.1: Overview of the RL setting. The agent receives an observation from the environment. Given the observation the agent suggests the next action based on its policy. The environment processes the action and returns an observation and a reward.

temporal distance. As such, immediate rewards are given more weight than those occurring further in the future. The objective in RL is to maximize the discounted return, defined as the weighted sum of future rewards:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \tag{3.1}$$

$$= r_{t+1} + \gamma G_{t+1}. \tag{3.2}$$

This sum considers the rewards obtained from time t until the final time T. Note that in Eq. (3.2), the return is defined in a recursive manner crucial for many RL algorithms.

Formally, we maximize the discounted return by learning the *optimal policy*, π^*. The policy determines which action to take given the observations, and therefore dictates the strategy followed by the agent. In general, a policy can take many different forms, ranging from look-up tables that map actions to states to ML models [60].

3.2 Markov decision process

All RL problems are represented as a Markov decision process (MDP). A general framework for modeling environments with sequential relationships between states is provided by MDPs. In such environments, the future is independent of the past given the present, which is known as the Markov property. Consequently, there are no memory effects from previously visited states. Formally, at any time step t,

$$p(s_{t+1}|s_0, \ldots, s_t) = p(s_{t+1}|s_t). \tag{3.3}$$

Mathematically, an MDP is expressed as the tuple $(\mathcal{S}, \mathcal{A}, p, G, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, p represents the dynamics, G is the set of

total returns, and γ is the discount factor. In this context, the return G and the discount factor γ define the objective, while p encapsulates the dynamics of the environment:

$$p(s', r|s, a) = p(s_{t+1} = s', r_{t+1} = r|s_t = s, a_t = a), \tag{3.4}$$

and represents the joint probability of transitioning to a new state s' and receiving a reward r after taking action a in state s. In fully deterministic environments, $p(s', r|s, a)$ takes values of either zero or one.

In the interactive setting between the agent and the environment, the agent selects actions according to a policy, which maps states to the probability of executing each possible action:

$$\pi(a|s) = p(a_t = a|s_t = s). \tag{3.5}$$

The goal in RL is to adapt the policy based on the experience gained from interacting with the environment to maximize the return. These interactions generate sequences, or *trajectories*, of the form

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, \ldots, s_T, \tag{3.6}$$

where all states, actions, and rewards are random variables. Thus, the agent follows a trajectory through the state-action space $\tau = a_0, s_1, a_1, \ldots, s_T$ with probability

$$p(\tau) = \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t), \tag{3.7}$$

starting from an initial state s_0. The discounted return associated with the trajectory is defined as $G(\tau) = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$.

This entire formalism assumes the Markov property from Eq. (3.3), which implies that the environment is memoryless. However, there are situations where the environment exhibits memory effects, such as in games where executing a sequence of actions has an additional effect at the end. In such cases, the Markov property is restored by considering an extended state space that includes this memory. Consequently, even deterministic Markovian dynamics in the full state space can lead to non-deterministic and non-Markovian dynamics in the reduced state space [60].

3.2.1 Bellman equations

As mentioned in previous sections, the goal in RL is to find the optimal policy π^* that maximizes the return, as introduced in Eq. (3.2). This clear objective allows us to define value functions that estimate how beneficial it is for the agent to be in a given state or to perform a certain action to achieve the task. These value

functions quantify the expected future return that the agent can obtain given the current conditions. Since future rewards depend heavily on the actions the agent will take, value functions are defined with respect to the policy.

The *state-value function*, $V_\pi(s)$, for a state s under policy π represents the expected return when starting at state s and subsequently adhering to policy π. Formally, it is defined as

$$V_\pi(s) = \mathbb{E}[G_t|s_t = s] = \mathbb{E}\left[\sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \middle| s_t = s\right]. \tag{3.8}$$

Likewise, the *action-value function*, $Q_\pi(s, a)$, denotes the expected return starting from state s, taking action a, and then following policy π:

$$Q_\pi(s, a) = \mathbb{E}[G_t|s_t = s, a_t = a] = \mathbb{E}\left[\sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right]. \tag{3.9}$$

The value functions satisfy a recursive relationship used by many RL algorithms. This relationship originates from the recursive nature of the return [Eq. (3.2)] and allows us to write the state-value function $V_\pi(s)$ as a function of the next states:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}[G_t|s_t = s] = \mathbb{E}[r_{t+1} + \gamma G_{t+1}|s_t = s] \\ &= \sum_a \pi(a, s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma \mathbb{E}[G_{t+1}|s_{t+1} = s']\right) \\ &= \sum_a \pi(a, s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma V_\pi(s')\right) \\ &= \mathbb{E}[r_{t+1} + \gamma V_\pi(s_{t+1})|s_t = s]. \end{aligned} \tag{3.10}$$

A similar derivation can be performed for the action-value function $Q_\pi(s, a)$:

$$\begin{aligned} Q_\pi(s, a) &= \mathbb{E}[G_t|s_t = s, a_t = a] = \mathbb{E}[r_{t+1} + \gamma G_{t+1}|s_t = s, a_t = a] \\ &= \sum_{s',r} p(s', r|s, a) \left(r + \gamma \mathbb{E}[G_{t+1}|s_{t+1} = s']\right) \\ &= \sum_{s',r} p(s', r|s, a) \left(r + \gamma V_\pi(s')\right) \\ &= \mathbb{E}[r_{t+1} + \gamma V_\pi(s_{t+1})|s_t = s, a_t = a], \end{aligned} \tag{3.11}$$

from which the relationship $V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s, a)$ becomes clear as displayed in the backup diagram in Fig. 3.2. The term "backup" refers to the process of propagating value information backwards from future states to the current state s. These are the *Bellman equations* for the value functions, which are fundamental to RL. They establish the relationship between the value of a state s and its successors s', recursively incorporating information about the future [60].

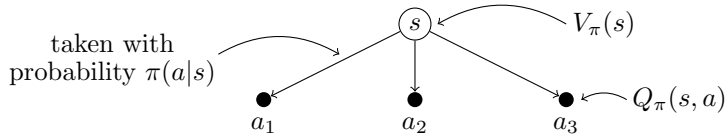


Figure 3.2: A backup diagram rooted at the state, s, and considering each possible action, a.

3.2.2 Bellman optimality equations

With the concepts of a state-value and action-value functions in place, we are ready to introduce the concept of partial orderings of policies. A policy π is considered better than or equivalent to another policy π' if its expected return is at least as great as that of π' for every state. Specifically, $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in \mathcal{S}$. There is always at least one policy that is as good as or better than all other policies. This is referred to as an optimal policy. Even though there might be multiple optimal policies, we denote all of them by π^*.

These optimal policies share common value functions, referred to as the *optimal state-value function*, denoted as V_{π^*}, and the *optimal action-value function*, represented as Q_{π^*}. They are defined as follows:

$$V_{\pi^*}(s) \doteq \max_\pi V_\pi(s), \tag{3.12}$$

$$Q_{\pi^*}(s,a) \doteq \max_\pi Q_\pi(s,a), \tag{3.13}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$.

By combining the Bellmann equations Eq. (3.10) and Eq. (3.11), we obtain

$$\begin{aligned} V_{\pi^*}(s) &= \max_a \mathbb{E}\left[G_t | s_t = s, a_t = a\right] \\ &= \max_a \mathbb{E}\left[r_{t+1} + \gamma V_{\pi^*}(s_{t+1}) | s_t = s, a_t = a\right] \\ &= \max_a Q_{\pi^*}(s,a), \end{aligned} \tag{3.14}$$

thereby establishing the relationship $V_{\pi^*}(s) = \max_a Q_{\pi^*}(s,a)$, showing how the optimal state-value $V_{\pi^*}(s)$ is determined by selecting the maximum Q-value among all possible actions a. The "max" label in the figure emphasizes this selection process [Fig. 3.3].

Note that this new Bellman equation maximizes over the first action, rather than taking the expectation over actions as in Eq. (3.10). This is because the value of a state under the optimal policy must equal the expected return for the

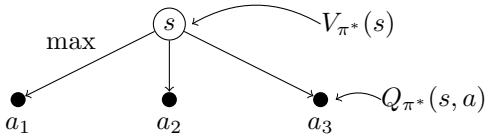


Figure 3.3: A backup diagram for the optimal state-value function rooted at the state, considering each possible action but selecting only the action that maximises the cumulative reward.

best action. Now we can define the set of *Bellman optimality equations*:

$$V_{\pi^*}(s) = \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_{\pi^*}(s')\right], \tag{3.15}$$

$$Q_{\pi^*}(s, a) = \sum_{s',r} p(s', r|s, a) \left[r + \gamma \max_{a'} Q_{\pi^*}(s', a')\right]. \tag{3.16}$$

We can define the optimal policy $\pi^*(a|s)$ and optimal action a^* at a given state s as:

$$\pi^* = \arg\max_\pi V_{\pi^*}(s), \tag{3.17}$$

$$a^* = \arg\max_a Q_{\pi^*}(s, a). \tag{3.18}$$

The optimal policy π^* corresponds to the deterministic choice of the best action a^* for a given state s based on the optimal action-value function $Q_{\pi^*}(s, a)$. Due to the recursive nature of the value functions, a greedy action according to V_{π^*} or Q_{π^*} is optimal in the long term [60].

3.3 Value-based methods

In value-based RL, the objective is to derive the optimal policy $\pi^*(a|s)$ by learning the optimal value functions, as described in Eqs. (3.15) and (3.16). We begin with an initial estimation of the value function for each state, $V_\pi(s)$, or state-action pairs, $Q_\pi(s, a)$. These estimations are then progressively updated based on the experience accumulated by the agent as it follows its policy.

One of the most straightforward methods to learn the value function is to sample trajectories $\tau \sim p(\tau)$ [Eq. (3.7)], and then use the return G_t to update our value function estimation for every visited state s_t:

$$V_\pi(s_t) = V_\pi(s_t) + \eta\Big(G_t - V_\pi(s_t)\Big), \tag{3.19}$$

where η is the learning rate. Note, we can also learn $Q_\pi(s, a)$ for every visited state and action along the trajectory. However, this approach only allows learning

at the end of each trajectory, known as *episodes*, which can be highly inefficient for problems involving long or infinite episodes. This estimate of the value function introduces a low bias but high variance, which is typical for Monte Carlo methods. In contrast, temporal difference (TD) algorithms take advantage of the recursive nature of the value functions, as shown in Eq. (3.10) and Eq. (3.11), to learn at each time step:

$$V_\pi(s_t) = V_\pi(s_t) + \eta\Big(r_{t+1} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)\Big). \tag{3.20}$$

It is important to note that while $V_\pi(s_t)$ is an estimate, $V_\pi(s_{t+1})$ is also an estimate. This is referred to as a bootstrapping method, where the update is partially based on another estimate and thus introducing a high bias with low variance on the estimates. The term in brackets is known as the TD error.

The algorithm that implements Eq. (3.20) is referred to as TD(0), which is a special instance of the TD(λ) algorithms [60, 136]. An equivalent algorithm for the action-value function is called SARSA [60, 136]:

$$Q_\pi(s,a) = Q_\pi(s,a) + \eta\left(r + \gamma Q_\pi(s',a') - Q_\pi(s,a)\right), \tag{3.21}$$

where the symbols s', a', and r represent the next state, action, and reward, respectively [60]. If we maximize over the next action $[\max_{a'} Q_\pi(s',a')]$ we obtain Q-learning [140], for which we provide a detailed introduction in the following section.

3.3.1 Q-learning

Q-learning is a widely used RL algorithm known for its simplicity. Typically, we start by arbitrarily initializing our estimates $Q_\pi(s,a) \forall s \in \mathcal{S}, a \in \mathcal{A}$, in table form. However, for many scenarios having a table for the entire state-action space is not feasible. Therefore we have to rely on an efficient way to represent $Q_\pi(s,a) \forall s \in \mathcal{S}, a \in \mathcal{A}$. A promising candidate is a function approximator such as an NN instead of a look-up table [55]. Using an NN to learn the Q-values is known as deep Q-learning and the NN representing the value estimates is typically referred to as a deep Q-network (DQN). The DQN takes a representation of the state $\phi(s)$ as input and outputs $Q_\pi(s,a;\boldsymbol{\theta}) \quad \forall a \in \mathcal{A}$, where $\boldsymbol{\theta}$ denotes the set of trainable parameters.

Naively implementing the algorithm outlined in Algorithm 3 with the update rule for the parameter as

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \eta\left(r + \gamma \max_{a'} Q_\pi(s',a';\boldsymbol{\theta}) - Q_\pi(s,a;\boldsymbol{\theta})\right) \nabla_{\boldsymbol{\theta}} Q_\pi(s,a;\boldsymbol{\theta}) \tag{3.22}$$

is highly unstable. The instabilities are mainly due to correlations in consecutive observations along the trajectories, correlations between target and prediction, and significant changes in the data distribution due to small variations in the

Algorithm 3 Deep Q-learning with Experience Replay

Require: learning rate, η, replay memory capacity N
  Initialize replay memory $\mathcal{D}$ to capacity N
  Initialize action-value function Q with random weights
  for episode $= 1, M$ **do**
    Initialise sequence s_1
    for $t = 1, T$ **do**
      $\xi \leftarrow$ uniform $\in [0, 1]$
      $a \leftarrow$ uniform a **if** $\xi \leq \epsilon$ **else** $\arg\max_a Q(s, a; \boldsymbol{\theta})$ $\triangleright$ ϵ-greedy policy
      Execute action a and observe reward r and next state s'
      Store transition (s, a, r, s') in $\mathcal{D}$
      Sample random minibatch of transitions (s, a, r, s') from $\mathcal{D}$
      Set $y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}^-) & \text{for non-terminal } s' \end{cases}$
      $\boldsymbol{\theta} = \boldsymbol{\theta} + \eta \left(y - Q_\pi(s, a; \boldsymbol{\theta})\right) \nabla_{\boldsymbol{\theta}} Q_\pi(s, a; \boldsymbol{\theta})$
    end for
  end for

parameters. We overcome these challenges with experience replay [141] and introducing a target network.

Using experience replay, rather than learning at every time step, we store the experience gathered during episodes in a memory that retains the information of each transition (s, a, r, s'). Once the agent has accumulated sufficient experience, it replays a randomly sampled batch of transitions from its memory to compute the loss and update the DQN parameters. In this manner, the agent alternates between episodes to gather experience and replaying the stored transitions to perform the learning process.

To mitigate the correlation between target and prediction, we employ a target network, which is a duplicate of the DQN, updated at a different rate. While we update the DQN parameters $\boldsymbol{\theta}$ at every iteration, the target network parameters $\boldsymbol{\theta}^-$ are updated less frequently by copying $\boldsymbol{\theta}$. This target network is then used to predict the target term $\max_{a'} Q_\pi(s', a'; \boldsymbol{\theta}^-)$, ensuring that the prediction, $Q_\pi(s, a; \boldsymbol{\theta})$, and the target remain uncorrelated. Thus, we obtain:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \left(r_i + \gamma \max_{a'} Q_\pi(s', a'; \boldsymbol{\theta}^-) - Q_\pi(s_i, a_i; \boldsymbol{\theta}) \right)^2, \tag{3.23}$$

where $\boldsymbol{\theta}^-$ indicates the frozen copy of $\boldsymbol{\theta}$ and n the number of samples for the batch update [60]. Finally we update the target network every few iterations, $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$. The complete algorithm is described in Algorithm 3.

3.4 Planning and learning

Having examined model-free algorithms such as Q-learning (Section 3.3.1), we now turn to model-based RL. Unlike model-free approaches, model-based RL algorithms require a model of the environment. While model-based methods primarily use planning, and model-free methods focus on learning, they share significant similarities. Both approaches focus on computing value functions, anticipating future events, calculating backed-up values, and using these to update approximate value functions.

In the context of model-based RL, a model of the environment refers to any tool an agent can use to predict how the environment will respond to its actions. Given a state s and an action a, a model forecasts the resulting next state s' and reward r. Stochastic models present multiple possible outcomes, each with an associated probability. To illustrate, in chess, this would be akin to a chess engine that an agent can interact with to evaluate potential moves and their consequences, allowing for strategic planning.

A family of algorithms, including AlphaGo, AlphaGo Zero, AlphaZero, and MuZero algorithms, have emerged as promising candidates in model-based RL to tackle complex decision-making tasks. AlphaGo [56] was the first computer program to achieve superhuman performance in the game of Go, demonstrating its ability to model and navigate the vast state space of the game. It defeated world champion Lee Sedol in 2016, marking an important moment in AI history. Building on this success, AlphaGo Zero [142] introduced the concept of *zero-knowledge learning*, eliminating the need for external human knowledge. Unlike its predecessor, AlphaGo Zero does not rely on expert game records to predict moves. Instead, it constructs its environmental model solely through self-play, learning from scratch. Experiments showed that this approach resulted in even stronger play, with AlphaGo Zero decisively beating AlphaGo 100-0.

The success of zero-knowledge learning paved the way for broader applications. Without the need for domain-specific knowledge, self-play training could be conducted across a wider class of games. This led to AlphaZero [48], which demonstrated that the same approach could be extended to chess and shogi, beating the state-of-the-art programs in each game.

MuZero [143], the latest in this lineage, takes a significant leap forward. It improves upon its predecessors by removing the need for explicit knowledge of the task environment, including game rules and state transitions. By learning additional representation and dynamics models, MuZero can plan ahead without directly interacting with the environment. This allows it to master both board games and Atari games, opening up the potential for extensions to complex real-world scenarios where the rules or dynamics may not be explicitly known [144].

All of these algorithm rely on a combination of Monte Carlo tree search (MCTS) and guidance by an NN. In the following, we will discuss in detail the AlphaZero algorithm, which powers our final research project enabling efficient

circuit routing.

3.4.1 AlphaZero

AlphaZero integrates MCTS with an NN for guidance. The network f_θ, parameterized by θ, processes a state representation s and produces action probabilities p and a value estimate v: $(p, v) = f_\theta(s)$. For each available action a, $p_a = \Pr(a|s)$ represents its selection probability, while v estimates the game outcome. In the context of two-player games, it estimates the current player's winning probability from state s.

The NN f_θ guides the MCTS simulations in AlphaZero, as illustrated in Fig. 3.4. Each tree edge (s, a) maintains a prior probability $P(s, a)$, visit count $N(s, a)$, and action value $Q(s, a)$. Simulations begin at the root, selecting moves that maximize $Q(s, a)+U(s, a)$, where $U(s, a) \propto P(s, a)/\big(1+N(s, a)\big)$, upon reaching a leaf node as depicted in Fig. 3.4(a). Once the leaf node is reached, the network evaluates it once, producing $\big(P(s'), V(s')\big) = f_\theta(s')$ [Fig. 3.4(b)]. Traversed edges update their visit counts and action values, with $Q(s, a) = 1/N(s, a) \sum_{s,a\to s'} V(s')$, where $s, a \to s'$ denotes a simulation path from s to s' via action a [Fig. 3.4(c)]. In the context of self-play, MCTS can be understood as an algorithm that generates move recommendations. Given network parameters θ and a starting position s, it calculates a probability vector $\pi = \alpha_\theta(s)$. Each move's probability π_a is proportional to its visit count raised to a power: $\pi_a \propto N(s, a)^{1/\tau}$, where τ represents a temperature parameter controlling exploration [Fig. 3.4(d)].

The NN in AlphaZero is trained through a self-play RL algorithm that employs MCTS for move selection. The process begins with random initialization of network weights θ_0. For each subsequent iteration $i \geq 1$, self-play games are generated as illustrated in Fig. 3.5(a). Importantly, while this framework is often associated with two-player games, it can also be adapted for single-player scenarios [137, 138]. At each time step t within a game, an MCTS search using the previous iteration's NN $f_{\theta_{i-1}}$ produces move probabilities $\pi_t = \alpha_{\theta_{i-1}}(s_t)$, from which moves are sampled. Games conclude when predefined conditions are met, such as both players passing, reaching a maximum game length, or any other stopping criterion reasonable for the environment. Upon termination, a final reward r is assigned. The algorithm stores data for each time step t as a tuple (s_t, π_t, z_t), where z_t represents the outcome of the game. In a two-player game, its the winner; in the context of optimization tasks it is the terminal reward obtained. Concurrently, as shown in Fig. 3.5(b), new network parameters θ_i are trained using data (s, π, z) sampled uniformly from recent self-play iterations. The neural network $f_{\theta_i}(s) = (\mathbf{p}, v)$ is optimized to minimize the difference between the predicted values v and the actual game results z, while maximizing the similarity between the network's move probabilities $\mathbf{p}$ and the MCTS-derived probabilities π. This optimization is achieved through gradient descent on a loss function that

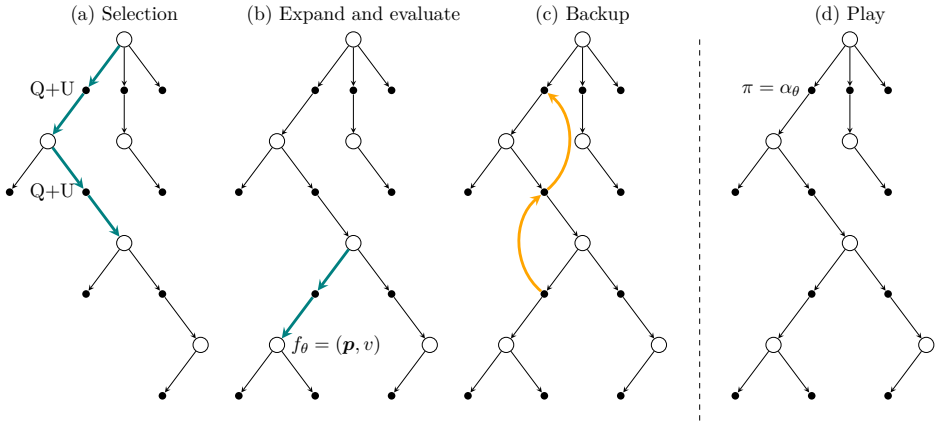


Figure 3.4: AlphaZero's MCTS process: (a) Tree traversal selects edges maximizing $Q + U$, where U is based on prior probability P and visit count N. (b) In the leaf nodes, the neural network $f_\theta(s)$ evaluates position s, generating $(P(s), V(s))$; P values initialize new edges. (c) The Q values are updated to reflect the mean evaluations in the subtrees. (d) The final search probabilities π are computed as $N^{1/\tau}$, with N being the root move visit counts and τ controlling the exploration temperature.

combines mean-squared error and cross-entropy losses:

$$(\mathbf{p}, v) = f_\theta(s), \quad l = (z - v)^2 - \boldsymbol{\pi}^\top \log \mathbf{p} + c\|\theta\|^2. \tag{3.24}$$

Here, c controls the level of L2 weight regularization [145, 146], introduced to mitigate overfitting. This iterative process of self-play and network optimization continues, progressively enhancing the NN's ability to evaluate game states and suggest strong moves across various scenarios [48].

This marks the end of our exploration of ML and RL methodologies. The subsequent chapter will investigate how analogous concepts can be transferred to quantum computers, where qubits replace classical bits as the computational units.

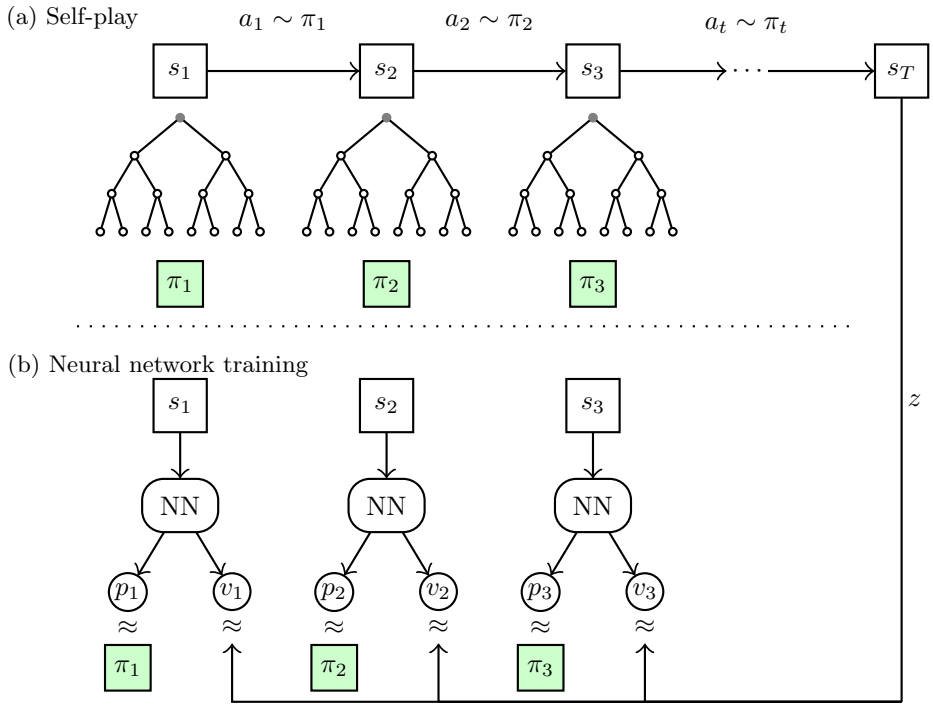


Figure 3.5: Overview of the AlphaZero self-play framework. a) The program played multiple games, illustrated by a sequence of positions s_i, $i = 1, 2, \ldots, T$, with moves a_i and final outcome z. Each move a_i was determined by action probabilities π_i from the MCTS, guided by an NN. The network's input is a representation of the state s_i, outputting move probability vectors p for MCTS and outcome estimates v for each position s_i. b) Network training used randomly sampled steps from recent self-play games. Weights were updated to align policy vector p with the MCTS probabilities π, and to incorporate the outcome z in estimates v.

Chapter 4

Quantum machine learning

Up to this point, we have examined various ML approaches, including supervised and unsupervised learning tasks, where the goals are to predict based on labels, estimating specific values, or identifying patterns within data. These algorithms typically run on accelerator devices such as graphics processing units (GPUs). Near-term quantum computers may be used in a similar manner, acting as accelerators for learning problems.

The branch of quantum technologies dedicated to these algorithms is known as quantum machine learning (QML). Its goal is to use quantum hardware instead of classical accelerators such as GPUs. A promise of QML is its ability to enhance the capabilities of ML algorithms by leveraging the laws of quantum physics, addressing problems that are simply too complex for classical computers. In the rest of the chapter, we focus on a type of QML algorithm that is at its core similar to classical ML algorithms: variational quantum algorithms (VQAs).

This chapter begins with an introduction to VQAs, followed by a detailed description of a specific algorithm, the quantum approximate optimization algorithm (QAOA). We then explore the application of gradient-based optimization methods, a framework introduced in Chapter 2, to these quantum algorithms, which is central to our work in Paper C. The chapter concludes with an introduction to the Ising model, a key component of Paper B.

4.1 Variational quantum algorithms

VQAs are a class of hybrid quantum-classical algorithms that have emerged to run on the current generation of noisy intermediate-scale quantum (NISQ) devices [15, 22, 147][1]. Many experimental proposals for NISQ devices involve train-

[1] In this context, intermediate-scale refers to the size of quantum computers with 50 to a few hundred qubits. "Noisy" implies that we cannot protect the system well against losses and

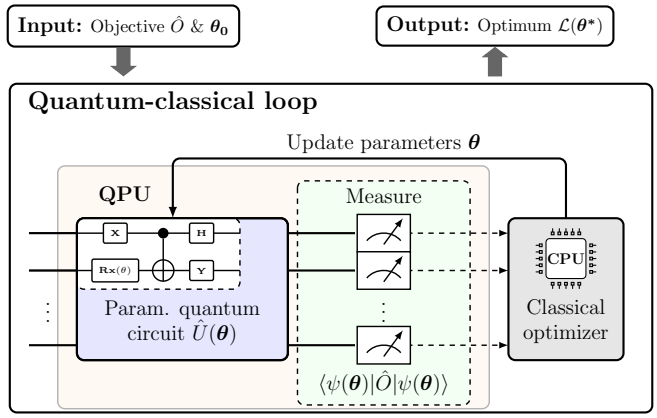


Figure 4.1: A diagrammatic representation of a VQA consists of three main elements: an objective function that defines the problem to be solved, a PQC $\hat{U}(\boldsymbol{\theta})$ in which parameters $\boldsymbol{\theta}$ are adjusted to minimize the objective, and a classical optimizer that performs this minimization. The inputs for a VQA are the circuit ansatz and initial parameter $\boldsymbol{\theta}_0$ values, while the outputs are the optimized parameter values $\boldsymbol{\theta}^*$ and the minimum value of the objective function, $\langle\psi(\boldsymbol{\theta})|\,\hat{O}\,|\psi(\boldsymbol{\theta})\rangle$.

ing a closed-loop optimization between a quantum device and a classical computer. Such hybrid quantum-classical algorithms are popular for chemistry [77–79], optimization [38–42], and machine learning [24, 27, 28, 30, 80] applications.

The first step is to define a loss (or energy) function $\mathcal{L}$, which encodes the problem. Next, one proposes an ansatz, i.e., a quantum operation depending on a set of continuous or discrete parameters $\boldsymbol{\theta}$ that can be optimized. This ansatz is analogous to the NN functions described in Chapter 2, with the key difference being its execution on a QPU rather than a GPU. This ansatz is then optimized in a hybrid quantum-classical loop (see Fig. 4.1) to solve the optimization task

$$\boldsymbol{\theta^*} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\ \mathcal{L}(\boldsymbol{\theta})\,. \tag{4.1}$$

This optimization task poses several requirements for the classical optimizer. Specifically, the optimizer must identify solutions in the presence of noise and scale favorably with the number of variational parameters, i.e., the running time must not increase exponentially as the number of parameters increases. In addition, the classical optimizer should be able to locate good parameter settings while limiting the number of queries it sends to the QPU. It is not surprising that classical optimizers have been widely explored given all these requirements. To this end, several optimization algorithms ranging from classical noise-resilient

errors, which will lead to severe limitations in near-term quantum devices. We discuss the aspect of errors in quantum systems in Chapter 6.

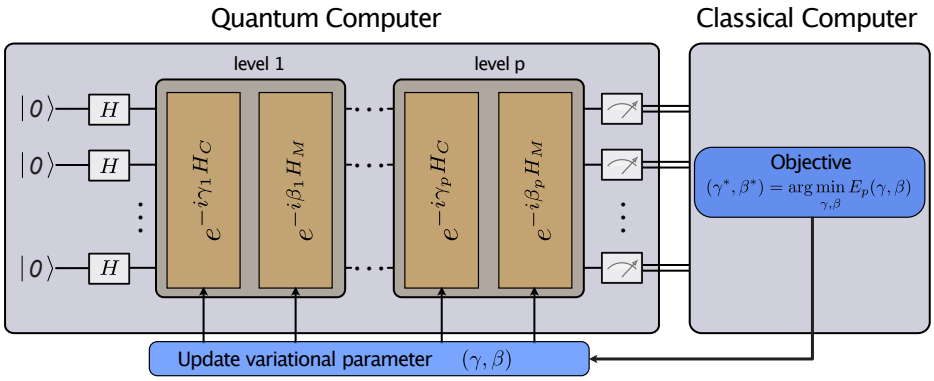


Figure 4.2: A schematic illustration of the QAOA, visualizing the interplay between the quantum device and the classical computer. The quantum computer implements a variational state formed by applying p parameterized layers of operations. Each layer has operations involving the cost Hamiltonian H_C and a mixing Hamiltonian H_M, weighted by the angles γ and β, respectively. Measurements of the variational state and calculations of its resulting energy are used to guide the classical optimizer, which minimizes the energy in a closed-loop optimization.

algorithms [148–153] to machine-learning approaches [154–161] have been suggested.

However, there exists a large class of quantum circuit ansätze where the loss-function landscapes concentrate exponentially towards their mean value as the system size grows. On such landscapes, exponential resources are required for training, prohibiting the successful scaling of variational quantum algorithms. This phenomenon, known as *barren plateaus* [22, 162–170], points to a crucial design principle for variational quantum models; restrict the circuit ansatz to a relevant subspace of the Hilbert space. While this can be difficult, it is essential when barren plateaus hinder the optimization process.

4.1.1 Quantum approximate optimization algorithm

Among VQAs, a prominent algorithm is the quantum approximate optimization algorithm (QAOA), which we studied for a particular optimization problem in Paper B. As a hybrid quantum-classical algorithm, the QAOA combines quantum and classical processing in a closed-loop optimization, illustrated in Fig. 4.2. This figure showcases the algorithm's various components and the building blocks that constitute the quantum circuit. The quantum subroutine, operating on n qubits, consists of a consecutive application of two non-commuting operators de-

fined as

$$U(\gamma) \equiv e^{-i\gamma H_C} \quad \gamma \in [0, 2\pi]\,, \tag{4.2}$$

$$U(\beta) \equiv e^{-i\beta H_M} = \prod_{j=1}^{n} e^{-i\beta X_j} \quad \beta \in [0, \pi]\,. \tag{4.3}$$

The X_j operation is analogous to the classical NOT gate on qubit j. It changes the $|0\rangle$ state to the $|1\rangle$ state, and vice versa. The operator $U(\gamma)$ gives a phase rotation to each computational basis state based on its associated cost, while the mixing term $U(\beta)$ creates superpositions of these states. We call $U(\gamma)$ the phase-separation operator with H_C the cost Hamiltonian (detailed in Section 4.3) and $U(\beta)$ the mixing operator with H_M the mixing Hamiltonian. The bounds for γ and β are valid if H_C has integer eigenvalues [171].

The initial state for the algorithm is a superposition of all possible computational basis states. This superposition can be obtained by first preparing the system in the initial state $|0\rangle^{\otimes n} = |00\ldots0\rangle$ for all qubits and then applying the Hadamard gate on each qubit:

$$\left(\tilde{H}\,|0\rangle\right)^{\otimes n} = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)^{\otimes n} \equiv |+\rangle^{\otimes n}\,, \tag{4.4}$$

where $\otimes$ denotes the tensor product and $\tilde{H}$ the Hadamard gate.

For any integer $p \geq 1$ and $2p$ angles $\gamma_1 \ldots \gamma_p \equiv \boldsymbol{\gamma}$ and $\beta_1 \ldots \beta_p \equiv \boldsymbol{\beta}$, we define the angle-dependent quantum state

$$|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle = U(\beta_p)U(\gamma_p)\ldots U(\beta_1)U(\gamma_1)\,|+\rangle^{\otimes n}\,. \tag{4.5}$$

The quantum circuit parameterized by $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ is then optimized in a closed loop using a classical optimizer. The objective is to minimize the expectation value of the cost Hamiltonian H_C [171], i.e.,

$$(\boldsymbol{\gamma^*}, \boldsymbol{\beta^*}) = \underset{\boldsymbol{\gamma},\boldsymbol{\beta}}{\operatorname{argmin}}\; E(\boldsymbol{\gamma}, \boldsymbol{\beta})\,, \tag{4.6}$$

$$E(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \langle \boldsymbol{\gamma}, \boldsymbol{\beta}|\, H_C\, |\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle\,. \tag{4.7}$$

To provide context for the cost Hamiltonian, Section 4.3 introduces the Ising model and illustrates its application in formulating combinatorial optimization problems. The problem of calculating the energy of $2^{\#q}$ ($\#q$ denotes the number of qubits) possible bitstrings (solutions) is thus reduced to a variational optimization over $2p$ parameters. For a detailed description, see Algorithm 4.

Algorithm 4 Quantum approximate optimization algorithm

Input cost Hamiltonian, H_C, mixer Hamiltonian, H_M, and layers, p.
Construct the circuits $U(\gamma) = e^{-i\gamma H_C}$ and $U(\beta) = e^{-i\beta H_M}$.
Build the state

$$|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle = U(\beta_p)U(\gamma_p)\dots U(\beta_1)U(\gamma_1)\,|+\rangle^{\otimes n}\,.$$

while Optimization (e.g. Nelder-Mead) of $\langle \boldsymbol{\gamma}, \boldsymbol{\beta}|\, H_C\, |\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$ **do**
    for number of shots **do**
        Sample variational quantum state, $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$
        Record measurement outcome
    end for
    Calculate $E(\boldsymbol{\gamma}, \boldsymbol{\beta})$
    Use optimization routine to propose new $\boldsymbol{\gamma}, \boldsymbol{\beta}$ with information $E(\boldsymbol{\gamma}, \boldsymbol{\beta})$
end while
return the final parameters $\boldsymbol{\gamma}, \boldsymbol{\beta}$ and energy of the cost function $E(\boldsymbol{\gamma}, \boldsymbol{\beta})$

4.2 Optimizing variational parameters

The classical computer's role is to iteratively adjust the parameters from an initial guess $\boldsymbol{\theta}_0$ to minimize the cost function. While this process mirrors the optimization of classical NNs discussed in Chapter 2, the quantum nature of VQAs presents unique challenges. Unlike classical NNs, where backpropagation efficiently computes gradients, quantum circuits do not allow for direct differentiation due to their probabilistic nature and the no-cloning theorem.

To address this, various methods have been developed for estimating gradients in quantum circuits. These include finite-difference methods, linear combinations of unitaries [172], and the parameter-shift rule [173, 174]. The latter evaluates the cost function at two shifted parameter positions, using their rescaled difference as an unbiased gradient estimate.

Given these gradient estimates, we can employ gradient descent (GD), applying the update rule $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\nabla\mathcal{L}_k$. Here, $\eta \in \mathbb{R}^+$ is the step size controlling parameter changes per iteration, and $\nabla\mathcal{L}_k \equiv \nabla\mathcal{L}(\boldsymbol{\theta}_k)$ denotes the cost function gradient at the kth iteration. The optimization process terminates when the gradient norm $\|\nabla\mathcal{L}_k\|_2$ approaches zero, indicating a stationary point.

The limitations of these gradient estimation techniques motivate the exploration of advanced optimization strategies for VQAs. In Paper E, we delve into these challenges, proposing novel solutions to enhance the efficiency and effectiveness of VQA optimization.

4.2.1 Metric-informed optimization

VQAs utilize a parameterization of the wave function where the parameters represent the phases of unitary gates acting on an input state. A small change in a parameter $\delta\theta_i$ not only affects the observable of interest, as used by GD, but also impacts the associated metric $\langle\psi(\delta\theta_j)|\psi(\delta\theta_i)\rangle$. This additional information can help determine a more effective direction for the optimization trajectory. Two prominent metric-informed optimization strategies are quantum imaginary time evolution (QITE) and quantum natural gradient (QNG).

QITE [175–178] is based on the Wick-rotated ($\tau = \mathrm{it}$) [179] imaginary-time Schrödinger equation

$$\frac{\partial\,|\Psi(\tau)\rangle}{\partial\tau} = -\hat{H}\,|\Psi(\tau)\rangle \tag{4.8}$$

$$\text{or } |\Psi(\tau+\Delta\tau)\rangle = N(\tau)^{-1}\mathrm{e}^{-\Delta\tau\hat{H}}\,|\Psi(\tau)\rangle\,, \tag{4.9}$$

$$\text{with } N(\tau) = \sqrt{\langle\Psi(\tau)|\,\mathrm{e}^{-2\Delta\tau\hat{H}}\,|\Psi(\tau)\rangle} \tag{4.10}$$

and is a quantum algorithm finding the ground and excited states [180] of a quantum system. It is a variation of the imaginary time evolution algorithm [181–184], a well-established method in classical computational physics to determine the ground state of a system. The iterative application of the exponential operator with sufficiently small time steps $\Delta\tau$ [183] exponentially damps higher energy contributions, leading to convergence to the ground state $|\Psi_0\rangle$, provided the initial state $|\Psi(0)\rangle$ has a non-zero overlap with the ground state [175, 176]. However, since $\mathrm{e}^{-\Delta\tau\hat{H}}$ is not unitary, directly implementing imaginary time evolution (ITE) on quantum hardware is not straightforward.

One approach is to convert QITE into a hybrid quantum-classical variational form (VarQITE) [176, 177] (Fig. 4.1), where the target state $|\Psi(\tau)\rangle$ is encoded by a PQC. This is expressed as $|\psi(\boldsymbol{\theta}(\tau))\rangle = \hat{U}(\boldsymbol{\theta}(\tau))\,|\psi_0\rangle$ and the time evolution is translated to the parameters $\boldsymbol{\theta}(\tau)$ of the variational ansatz. The update rule for the parameters $\boldsymbol{\theta}_k$ for the next iteration $k+1$ at (imaginary) time $\tau+\Delta\tau$ is obtained by applying McLachlan's variational principle [185] to Eq. (4.8), aiming to minimize the discrepancy between the time evolution of the ansatz state $|\psi(\tau)\rangle \equiv |\psi(\boldsymbol{\theta}(\tau))\rangle$ to the exact imaginary time evolution

$$\delta\|\Big(\partial/\partial\tau + \hat{H} - E_\tau\Big)\,|\psi(\tau)\rangle\|_2 = 0, \tag{4.11}$$

where $\||\psi\rangle\|_2 = \sqrt{\langle\psi|\psi\rangle}$ is the 2-norm of a quantum state $|\psi\rangle$ and $E_\tau = \langle\psi(\tau)|\,\hat{H}\,|\psi(\tau)\rangle$ is the expected energy at time τ. By solving Eq. (4.11) we obtain the imaginary-time derivative of the parameters,

$$\frac{\partial\boldsymbol{\theta}}{\partial\tau} = -2\,\mathbf{F}^{-1}\nabla\mathcal{L}, \tag{4.12}$$

where $\mathbf{F}$ is the quantum Fisher information matrix (QFIM) and $\nabla\mathcal{L}$ the cost gradient. Eq. (4.12) enables the parameter updates for the next iteration. With a fixed time-step $\Delta\tau$ and the Euler method, this becomes

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \Delta\tau \frac{\partial \boldsymbol{\theta}}{\partial \tau} = \boldsymbol{\theta}_k - \frac{\Delta\tau}{2}\mathbf{F}_k^{-1}\nabla\mathcal{L}_k, \tag{4.13}$$

or higher-order methods [186]. Here, $\Delta\tau$ is equivalent to the step size, η, in the GD update rule (see Chapter 2). The elements of the QFIM are given by

$$\mathbf{F}_{ij} = 4\mathrm{Re}\left[\left\langle \partial_{\theta_i}\psi|\partial_{\theta_j}\psi\right\rangle - \left\langle \partial_{\theta_i}\psi|\psi\right\rangle\left\langle \psi|\partial_{\theta_j}\psi\right\rangle\right], \tag{4.14}$$

where, $\partial_{\theta_i} \equiv \frac{\partial}{\partial_{\theta_i}}$. The QFIM $\mathbf{F}$ represents the complex geometry of the parameter space [187, 188] and acts as the quantum equivalent of the classical Fisher information matrix. This matrix is the unique Riemannian metric tied to a probability density function [189–191].

QNG [192] is another metric-informed optimization technique based on the principles of natural gradient descent by Amari *et al.* [189, 193–196], originally designed for optimizing NNs. Similar to VarQITE, the natural gradient takes into account the geometry of the function's parameter space and is calculated using the inverse of the QFIM [197, 198]. Therefore, employing QNG leads to steps that are better aligned with the geometry of the parameter space, enabling faster convergence, the ability to cross local minima, and helping the algorithm escape regions with vanishing gradients [163, 192, 199–203].

The primary disadvantage of QFIM and QNG is that computing the entire QFIM for an ansatz with $n_{\boldsymbol{\theta}}$ parameters is computationally intensive and requires measuring $\mathcal{O}(n_{\boldsymbol{\theta}}^2)$ terms in every iteration.

Approximating the metric

Due to the expensive computation of the QFIM, several approximations have been suggested, such as the (block-) diagonal approximation proposed by Stokes *et al.* [192], reduce the scaling to linear in the number of parameters by discarding the off-diagonal elements [199].

Another approximation is based on the assumption that the QFIM varies slowly as the parameter space is traversed. This allows to update the metric $\mathbf{B}_k$ between steps using a rank-1 perturbation based on the current gradient:

$$\mathbf{B}_{k+1} = (1-\varepsilon_k)\mathbf{B}_k + \varepsilon_k \nabla\mathcal{L}_k \nabla\mathcal{L}_k^\top, \tag{4.15}$$

where ε_k is a learning rate that decays over time and $\nabla\mathcal{L}_k \equiv \nabla\mathcal{L}(\boldsymbol{\theta_k})$.

To avoid inverting $\mathbf{B}_{k+1}$ at each step, using the Sherman-Morrison formula to update its inverse directly:

$$\mathbf{B}_{k+1}^{-1} = \left[\mathbb{1} - \frac{\varepsilon_k \mathbf{B}_k^{-1}\nabla\mathcal{L}_k\nabla\mathcal{L}_k^\top}{1-\varepsilon_k(1-\nabla\mathcal{L}_k^\top\mathbf{B}_k^{-1}\nabla\mathcal{L}_k)}\right]\frac{\mathbf{B}_k^{-1}}{1-\varepsilon_k} \tag{4.16}$$

and therefore maintaining positive semi-definiteness of the metric, reduces the number of necessary circuit evaluations.

While this approximation method can improve computational efficiency, it may lose some of the theoretical convergence guarantees associated with exact QITE. This approximation is a key component in the optimizer introduced in Paper C.

4.3 The Ising model

Our earlier discussion of VQAs highlighted the role of the cost Hamiltonian H_c in capturing the details of the optimization problem. To bridge the gap between classical combinatorial optimization problems and quantum algorithms, we must represent these problems in a manner that quantum devices can interpret and manipulate. This section introduces the Ising model as a tool for recasting optimization problems into quantum variables, which is central to the work in Paper B.

4.3.1 A model to describe magnetism

The Ising model, introduced in the mid-1920s by Ernst Ising and Wilhelm Lenz, is one of the most influential models in physics. Originally conceived to explain magnetic materials, the method has found applications far beyond its initial scope [38–41, 154, 204–207]. At its core, the Ising model represents a magnetic material as an assembly of interacting atomic spins.

In this model, each atom possesses a spin $s_i \in \{-1, +1\}$, which can align or anti-align with an applied magnetic field. The spins interact with each other, creating a complex network of interactions. The energy of a spin configuration $\boldsymbol{s}$ is given by

$$E(\boldsymbol{s}) = \sum_{i,j'} J_{ij} s_i s_j + \sum_i h_i s_i \equiv \langle \boldsymbol{s}, \boldsymbol{J}\boldsymbol{s} \rangle + \langle \boldsymbol{h}, \boldsymbol{s} \rangle. \tag{4.17}$$

Here, h_i represents the strength of the applied field at atom i, while J_{ij} denotes the interaction strength between spins i and j. At low temperatures, the system favors low-energy states. A positive J_{ij} promotes anti-alignment of neighboring spins ($s_i s_j = -1$), whereas a negative J_{ij} encourages alignment ($s_i s_j = 1$).

Its versatility has led to its adoption in modeling a wide array of physical systems, which extends far beyond its original domain. The model's framework can describe any system composed of pairwise-interacting independent elements. This adaptability makes it an ideal candidate for representing various computational problems, including NP-complete ones, which we will explore in the next section.

4.3.2 Ising model for some NP-complete problems

As discussed, the versatility of the Ising model extends to representing complex computational challenges, particularly NP-complete problems. These problems are characterized by the absence of known polynomial-time algorithms for their solution, despite the ability to quickly verify a proposed solution [208, 209]. The ability to map NP-complete problems to the Ising model opens up possibilities for utilizing quantum annealing and other quantum optimization techniques. Next, we will explore how some NP-complete problems, including the heterogeneous vehicle routing problem (HVRP) considered in Paper B, can be formulated within the Ising framework.

Knapsack problem

The knapsack problem, a classic optimization challenge with applications spanning logistics and finance [210, 211], exemplifies the complexity of resource allocation. Consider a set of N objects, each with an integer weight w_i and value c_i. Given a knapsack with maximum capacity W, the goal is to select items that maximize total value $\mathcal{C}$ while keeping total weight $\mathcal{W}$ within capacity. Formally, we seek to optimize:

$$\mathcal{C} = \sum_{i=1}^{N} c_i x_i \quad \text{subject to} \quad \mathcal{W} = \sum_{i=1}^{N} w_i x_i \leq W, \tag{4.18}$$

where $x_i \in 0, 1$ indicates item inclusion. This NP-hard problem [212, 213] elegantly captures the essence of constrained optimization.

The knapsack problem can be mapped onto an Ising model. Following Lucas [213], we introduce binary variables z_n for $1 \leq n \leq W$, where $z_n = 1$ indicates that the final weight of the knapsack is n. The problem is then formulated as minimizing a Hamiltonian $H = H_A + H_B$, where:

$$H_A = A\left(1 - \sum_{n=1}^{W} z_n\right)^2 + A\left(\sum_{n=1}^{W} n z_n - \sum_{i=1}^{N} w_i x_i\right)^2, \tag{4.19}$$

$$H_B = -B\sum_{i=1}^{N} c_i x_i\,. \tag{4.20}$$

Here, H_A enforces the problem constraints, while H_B represents the optimization objective. To make sure that the hard constraint is not violated, we require $0 < \max(|H_B|) < A$ [214].

Traveling salesperson problem

The travelling salesperson problem (TSP) seeks the shortest path between a series of cities. Formally, given a graph $G = (V, E)$, where vertices represent cities and

edges denote routes, with associated weights W_{ij}, the goal is to find a minimum-weight Hamiltonian cycle [213], i.e., a cycle that visits each vertex once. The Ising formulation encodes this problem as a Hamiltonian $H = H_A + H_B$, where

$$H_A = A\sum_{i=1}^{N}\left(1-\sum_{\alpha=1}^{N} y_{i\alpha}\right)^2 + A\sum_{\alpha=1}^{N}\left(1-\sum_{i=1}^{N} y_{i\alpha}\right)^2 \tag{4.21}$$

$$+ A\sum_{(i,j)\notin E}\sum_{\alpha=1}^{N} y_{i\alpha}y_{j\alpha+1}, \tag{4.22}$$

$$H_B = B\sum_{(i,j)\in E} W_{ij}\sum_{\alpha=1}^{N} y_{i\alpha}y_{j\alpha+1}. \tag{4.23}$$

Here, $N = |V|$ is the number of nodes, A and B are positive constants, satisfying $0 < \max(|H_B|) < A$, and W encodes the distances between the nodes. The index i represents the nodes and α the order in a prospective cycle. The binary variables $y_{i\alpha}$ can be referred to as "routing variables" indicating in which order of the cycle node i is visited. There are N^2 variables, with $y_{i,N+1} \equiv y_{i,1}$ for all i, such that the route ends where it starts [213].

The heterogeneous vehicle routing problem

The vehicle routing problem (VRP), an extensively studied combinatorial optimization problem, focuses on determining the optimal route design for a fleet of vehicles serving multiple customers. Since its formalization by Dantzig and Ramser in 1959 [215], the VRP has spawned hundreds of research papers exploring exact and approximate solutions for its numerous variants. Among these variants is the HVRP, introduced by Golden et al. [216], which considers a fleet of vehicles with varying capacities and costs for distribution activities.

The HVRP can be formulated as follows [217]. Consider a complete graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where node 0 represents a single depot housing a fleet of vehicles. The set of nodes $\mathcal{N} = \{0, \ldots, n\}$ includes the depot and n customers, with the customer set defined as $\mathcal{N}_0 = \mathcal{N} \setminus \{0\}$. The set of edges $\mathcal{E} = \{(i,j) : 0 \leq i,j \leq n, i \neq j\}$ connects all nodes. Each customer i has a positive demand q_i. The vehicle fleet consists of k types, represented by the set $\mathcal{V} = \{1, ..., k\}$, with m_v vehicles available of each type $v \in \mathcal{V}$.

Delivering goods to meet customer demands involves various costs and constraints. These include the fixed vehicle cost t^v, which is independent of the distance traveled by a vehicle of type v and depends on factors such as powertrain, trailer, and engine type. Another consideration is the vehicle capacity Q^v, noting that different vehicle types may have the same capacities but vary in other aspects, such as powertrain [218]. Additionally, there's the distance-dependent cost c_{ij}^v, representing the cost of traveling on edge (i,j) with a vehicle of type v, including fuel and powertrain-related expenses. Binary variables x_{ij}^v are used,

where 1 indicates that a vehicle of type v travels on edge (i, j), and 0 otherwise. Furthermore, we denote by f_{ij}^v the amount of goods that are leaving node i to go to node j using truck of type v, while the amount of goods entering the node is denoted f_{ji}^v.

Using this notation, the HVRP is to minimize the cost

$$C_{\text{tot}} = \sum_{v\in\mathcal{V}} \sum_{j\in\mathcal{N}_0} t^v x_{0j}^v + \sum_{v\in\mathcal{V}} \sum_{(i,j)\in\mathcal{E}} c_{ij}^v x_{ij}^v \,, \tag{4.24}$$

subject to the constraints

$$\sum_{j\in\mathcal{N}_0} x_{0j}^v \leq m_v \quad v \in \mathcal{V} \,, \tag{4.25}$$

$$\sum_{v\in\mathcal{V}} \sum_{j\in\mathcal{N}} x_{ij}^v = 1 \quad i \in \mathcal{N}_0 \,, \tag{4.26}$$

$$\sum_{v\in\mathcal{V}} \sum_{i\in\mathcal{N}} x_{ij}^v = 1 \quad j \in \mathcal{N}_0 \,, \tag{4.27}$$

$$\sum_{j\in\mathcal{N}_0} x_{j0}^v = \sum_{j\in\mathcal{N}_0} x_{0j}^v \quad v \in \mathcal{V} \,, \tag{4.28}$$

$$\sum_{v\in\mathcal{V}} \sum_{j\in\mathcal{N}} f_{ji}^v - \sum_{v\in\mathcal{V}} \sum_{j\in\mathcal{N}} f_{ij}^v = q_i \quad i \in \mathcal{N}_0 \,, \tag{4.29}$$

$$q_j x_{ij}^v \leq f_{ij}^v \leq (Q^v - q_i) x_{ij}^v \quad (i,j) \in \mathcal{E}, v \in \mathcal{V} \,, \tag{4.30}$$

$$x_{ij}^v \in \{0, 1\} \quad (i,j) \in \mathcal{E}, v \in \mathcal{V} \,, \tag{4.31}$$

$$f_{ij}^v \geq 0 \quad (i,j) \in \mathcal{E}, v \in \mathcal{V} \,. \tag{4.32}$$

The objective function in Eq. (4.24) is the sum of the fixed vehicle cost for the vehicles used to deliver goods and the total (variable) travel cost for those vehicles. Constraint Eq. (4.25) ensures adherence to the availability limits of vehicle types. Equations (4.26) and (4.27) guarantee that each customer is served exactly once, while Eq. (4.28) ensures all vehicles return to the depot after completing deliveries. The proper flow of goods to meet the demands of the customers is enforced by constraints (4.29) and (4.30). Lastly, equations (4.31) and (4.32) impose binary and non-negativity restrictions on the variables, respectively.

By integrating the Ising models presented in Section 4.3.2 and Section 4.3.2 with our mathematical description of the HVRP, we can construct an Ising model for the HVRP. This approach, similar to formulations described in [219, 220], involves two steps. First, we extend the TSP formulation to encompass the VRP. Subsequently, we incorporate a capacity constraint inspired by the Knapsack Ising formulation to complete the HVRP model.

The original Ising formulation of the TSP employs binary variables $y_{i\alpha}$ to denote the order in which city i is visited in the cycle. In contrast, the HVRP mathematical formulation uses decision variables x_{ij}, where 1 indicates a vehicle

traverses edge ij, and 0 otherwise. To reconcile these approaches and integrate the TSP Ising model with the HVRP formulation presented in Eqs. (4.24)–(4.32), we need to establish a mapping from the y variables to the x variables. The map we use is

$$x_{ij}^{v} = \sum_{\alpha=1}^{N_0-1} y_{i\alpha}^{v} y_{j\alpha+1}^{v} \,, \tag{4.33}$$

$$x_{0i}^{v} = y_{i1}^{v} + \sum_{\alpha=2}^{N_0} \left(1 - \sum_{\substack{j=1 \\ j\neq i}}^{N_0} y_{j\alpha-1}^{v} \right) y_{i\alpha}^{v} \,, \tag{4.34}$$

$$x_{i0}^{v} = y_{iN_0}^{v} + \sum_{\alpha=1}^{N_0-1} y_{i\alpha}^{v} \left(1 - \sum_{\substack{j=1 \\ j\neq i}}^{N_0} y_{j\alpha+1}^{v} \right) \,. \tag{4.35}$$

The summation in Eq. (4.33) is non-zero only if nodes i and j are consecutive stops on the same route. Equations (4.34) and (4.35) guarantee that the initial and final stops are automatically linked to the depot (assuming there is only one depot). It is important to note that index 0 refers to the depot, while index 1 represents the first city (node) in the sequence of cities (nodes).

We can now express the Ising formulation for the routing problem. Let $V = |\mathcal{V}|$ represent the total number of trucks, where $\mathcal{V}$ refers to the set of vehicles selected for optimization (as opposed to vehicle *types*). Additionally, let $N_0 = |\mathcal{N}_0|$ denote the number of customers to be visited. In this formulation, the indices v correspond to a specific truck of a particular type, rather than just a vehicle type.

The Ising Hamiltonian we arrive at is then

$$H = H_A + H_B + H_C + H_D\,, \tag{4.36}$$

$$H_A = A\sum_{v=1}^{V}\sum_{i=1}^{N_0}\sum_{j=1}^{N_0} c_{ij}^{v}\sum_{\alpha=1}^{N_0-1} y_{i\alpha}^{v} y_{j\alpha+1}^{v} \tag{4.37}$$

$$+ A\sum_{v=1}^{V}\sum_{i=1}^{N_0} c_{0i}^{v}\left[y_{i1}^{v} + \sum_{\alpha=2}^{N_0}\left(1 - \sum_{\substack{j=1\\ j\neq i}}^{N_0} y_{j\alpha-1}^{v}\right) y_{i\alpha}^{v}\right] \tag{4.38}$$

$$+ A\sum_{v=1}^{V}\sum_{i=1}^{N_0} c_{i0}^{v}\left[y_{iN_0}^{v} + \sum_{\alpha=1}^{N_0-1} y_{i\alpha}^{v}\left(1 - \sum_{\substack{j=1\\ j\neq i}}^{N_0} y_{j\alpha+1}^{v}\right)\right]\,, \tag{4.39}$$

$$H_B = B\sum_{j=1}^{N_0}\sum_{v=1}^{V} t^{v}\sum_{\alpha=2}^{N_0}\left(1 - \sum_{i=1}^{N_0} y_{i\alpha-1}^{v}\right) y_{j\alpha}^{v}\,, \tag{4.40}$$

$$H_C = C\sum_{i=1}^{N_0}\left(1 - \sum_{\alpha=1}^{N_0}\sum_{v=1}^{V} y_{i\alpha}^{v}\right)^2\,, \tag{4.41}$$

$$H_D = D\sum_{\alpha=1}^{N_0}\left(1 - \sum_{i=1}^{N_0}\sum_{v=1}^{V} y_{i\alpha}^{v}\right)^2\,. \tag{4.42}$$

The Hamiltonian H in Eq. (4.36) is composed of different parts. Here, H_A in Eq. (4.39) captures the first part of the original mathematical formulation, i.e., the minimization of the variable cost. The first term estimates the variable cost for traveling between the different customers/cities, while the second and third terms measure the cost of leaving and arriving at the depot. For this particular mapping, it is necessary to define the set of vehicles used for the optimization beforehand. Therefore, we can neglect the inequality constraint defined in Eq. (4.25) from the original formulation, which ensures that the number of vehicles of a specific type does not exceed the number of available vehicles. Similarly, H_B in Eq. (4.40) estimates the fixed costs of each vehicle leaving the depot [see Eq. (4.24)]. Note that the prefactors A and B must be equal in order not to rescale the relative fixed versus variable costs. The constraint given by H_C in Eq. (4.41) ensures that each city is visited exactly once. Furthermore, H_D in Eq. (4.42) guarantees that each city has a unique position in the cycle and that not more than one city can be traveled to at the same time. We require $0 < \max(H_A + H_B) < C, D$, to satisfy the constraints.

The decision variables y_{ij}^{v} are positioned as shown in Fig. 4.3. It helps us understand what the different constraints enforce. Consider Eq. (4.41) and Eq. (4.42), summing over all indices v and i, ensuring that each column and row contains a

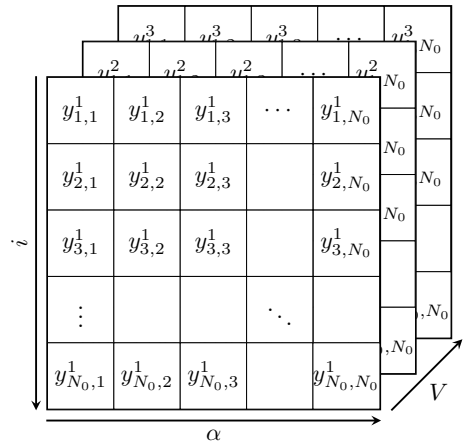


Figure 4.3: Visualization of the decision variables y^v_{ij} in the Ising formulation of the routing problem.

single non-zero element. This is important since a valid solution must visit each city exactly once.

Next, we focus on the capacity constraints described by the HVRP. The capacity constraint is of a similar nature as the constraints for the knapsack problem — both are described by an inequality constraint, which for the knapsack problem is not to add too many items to the knapsack and for the capacity problem not to overload the vehicles. Therefore, we can use the formulation given in Ref. [213] to model the inequality constraint introduced by the capacities.

We can make use of the inequality constraint given in the knapsack formulation [Eq. (4.19)] to encode the capacity constraints for the HVRP. Therefore, we can neglect H_B [see Eq. (4.20)] and only consider H_A [see Eq. (4.19)]. Let Q^v be the maximum capacity of vehicle v. The Hamiltonian then becomes

$$H_E = E \sum_v \left(1 - \sum_{k=0}^{Q^v} z_k^v\right)^2 + E \sum_v \left(\sum_{k=0}^{Q^v} k \cdot z_k^v - \sum_{\alpha,i} q_i y_{i\alpha}^v\right)^2 . \tag{4.43}$$

We can now combine the Ising model for the routing and capacities into one unifying Hamiltonian:

$$H_C = H_A + H_B + H_C + H_D + H_E . \tag{4.44}$$

For the terms H_A to H_E, see Eqs. (4.39)–(4.42) and (4.43) [214].

This concludes our exploration of QML methods. While these algorithms demonstrate the potential of quantum computing, their practical implementation hinges on efficient compilation to target devices. This process of transforming

abstract quantum algorithms into optimized circuits tailored for specific quantum architectures is the focus of QCO, which we will explore in depth in the next chapter.

Chapter 5

Quantum circuit optimization

Up to this point, we have explored various quantum computing concepts, from basic principles to near-term algorithms, such as VQAs. Referring back to the quantum computing stack introduced in Fig. 1.2, our focus has been primarily on the algorithm layer. Now, we descend to the QIR layer, bridging the gap between high-level algorithms and their actual implementation on quantum hardware.

This transition is analogous to the compilation process in classical computing, where high-level programming languages are translated into machine code. While classical compilers efficiently optimize code for specific hardware architectures, QCO performs a similar yet more complex role for quantum circuits. The additional layers of complexity in QCO stem from the unique constraints and characteristics of quantum systems.

These constraints often include limited qubit connectivity and platform-specific native gate sets, which can differ even across quantum processors based on the same technology, such as superconducting qubits. As a result, compiling high-level quantum algorithms into low-level instructions becomes more challenging, requiring the optimizer to account for qubit routing and the gates supported by the hardware. Although various hardware platforms face unique challenges, we focus specifically on the compilation process for superconducting qubits.

In the context of superconducting qubits, QCO involves solving a set of interconnected problems, such as initial qubit allocation, circuit routing (to facilitate interactions between non-adjacent qubits), and gate compression (to minimize the total gate count and circuit depth). Each of these subproblems presents its own set of challenges, often falling into the category of NP-hard optimization tasks [221].

In this chapter, we cover the fundamental components of QCO, including key

circuit representations essential for optimization. We examine each subproblem, with a particular focus on qubit routing, which is the primary challenge addressed in the final research project. Finally, we discuss how these elements come together in the overall QCO process.

5.1 Quantum circuit optimization pipeline

The process of QCO involves several interconnected steps, transforming an abstract quantum algorithm into an optimized circuit tailored for a specific quantum hardware.

- **Abstract algorithm representation**: The QCO process begins with an abstract quantum algorithm, often expressed as a series of unitary operations. For instance, the QAOA variational state can be represented as $U(\beta_p)U(\gamma_p)\dots U(\beta_1)U(\gamma_1)\,|+\rangle$, as displayed in Fig. 4.2.

- **Initial circuit decomposition**: The abstract representation is decomposed into a sequence of quantum gates. At this stage, we assume ideal conditions with full qubit connectivity, allowing any qubit to interact with any other without restrictions, thus violating some of the hardware constraints. We refer to the qubits in this representation as virtual ones.

- **Native gate set adaptation**: The circuit is then translated into the native gate set of the target quantum hardware. A native gate set typically consists of a universal set of quantum gates, often composed of just single- and two-qubit gates, which are sufficient to perform any quantum computation [7]. This step ensures that all operations can be physically implemented on the specific hardware device.

- **Qubit allocation**: Virtual qubits from the circuit are mapped to physical qubits on the hardware. This step takes into account factors like qubit quality and connectivity. Qubit quality refers to several characteristics, such as coherence time (the duration a qubit can maintain its quantum state), or gate fidelity (the accuracy of operations performed on the qubit) [222].

- **Qubit routing**: Given the hardware's connectivity constraints, additional operations (often SWAP gates) are inserted to enable interactions between non-adjacent qubits. This problem is related to token swapping, known to be NP-hard [221].

- **Circuit compression**: The final step involves reducing the circuit depth and gate count while preserving the algorithm's functionality. This often includes applying circuit identities and cancelling redundant operations.

It is important to note that these steps are not always executed in a strictly linear fashion. Different approaches to solving the QCO problem may vary in the order of execution, often combining or repeating steps in unique ways.

The complexity of QCO stems from the interplay between these steps. For instance, the initial qubit allocation significantly impacts the subsequent routing problem, while the choice of native gates can affect the possibilities for circuit compression. Moreover, many of these subproblems, such as qubit allocation and optimal routing, are computationally hard, often falling into NP-hard complexity classes [221].

5.2 Quantum circuit representation

In order to tackle the problem of QCO, it is important to work with an effective encoding for quantum circuits. Finding such an encoding that enables QCO is a non-trivial task. The challenge is compounded by hardware-specific constraints, such as limited qubit connectivity, which requires mapping from virtual to physical qubits. This mapping increases both the complexity of the problem and the amount of information needed to find a suitable state representation.

5.2.1 Standard circuit diagram

A quantum circuit is a model for quantum computation, analogous to classical circuits. The circuit model contains several key elements as depicted in Fig. 5.1. First are the qubits, represented by horizontal lines. Next, the state preparation or initial operations to set up the desired quantum state followed by gate operations are represented by boxes on qubit lines for single-qubit gates, or vertical lines connecting boxes for multi-qubit gates. Finally, the measurements are, depicted at the end of qubit lines; these operations read out the final quantum state.

The standard circuit diagram provides an intuitive visualization of the quantum algorithm, showing the sequence of operations from left to right. It illustrates which qubits are involved in each operation and the order of gate applications. While this representation is widely used, it has limitations for optimization tasks, particularly in capturing gate commutations and hardware constraints. As circuits become more complex, this representation can become difficult to analyze algorithmically. These limitations motivate the need for alternative representations, such as the directed acyclic graph (DAG) representation, which we discuss next.

5.2.2 Directed acyclic graph (DAG)

The circuit representation of a quantum computation (a large unitary matrix decomposed into single and multi-qubit interactions allowed by the target hardware) is usually not unique since various gates may commute. For example, the

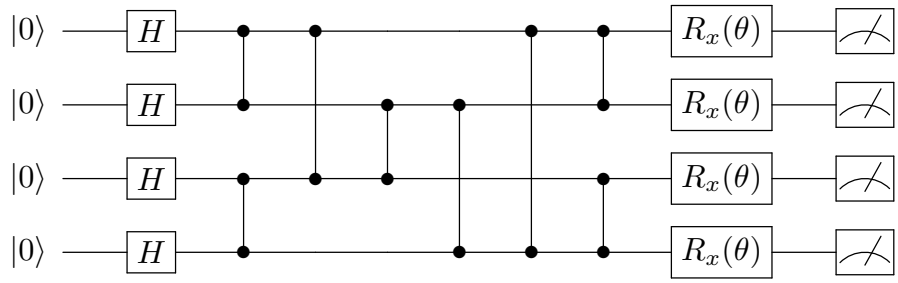
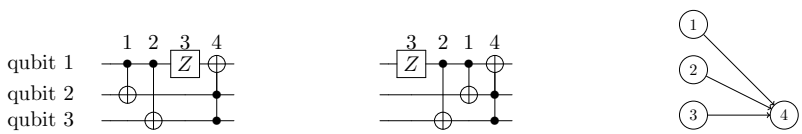


Figure 5.1: Quantum circuit diagram. Qubits, depicted as horizontal lines, undergo a series of operations from left to right. The circuit structure includes initial state preparation, followed by quantum gates (boxes for single-qubit operations, vertical lines for multi-qubit interactions), and concludes with measurement operations to extract the final quantum state.

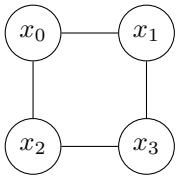
two circuits represented in Fig. 5.2(a) and Fig. 5.2(b) implement the same unitary operation. For some applications it is desirable to work with a representation of quantum circuits that does not change under pairwise commutation of gates, as shown in Fig. 5.2(c) [223, 224].

This form of a quantum circuit is a directed acyclic graph (DAG) with the following properties. Vertices in the graph correspond to individual gates in the circuit. The graph has an edge $i \to j$ from vertex i to vertex j if, by repeatedly interchanging commuting gates, one can bring gate i immediately to the left of gate j, but gates i and j themselves do not commute. Figure 5.2 illustrates this concept with an example circuit, its commuted equivalent, and the resulting canonical form. For any circuit its canonical form can be efficiently computed with an algorithm whose worst-case complexity scales quadratically in the number of gates [224].



(a) Circuit C (b) Commuted circuit C (c) Canonical form of C

Figure 5.2: Illustration of quantum circuit equivalence. (a) Initial circuit design, (b) restructured circuit with interchanged commuting gates, and (c) canonical representation, invariant under pairwise gate commutations. From Ref. [223].



(a) Coupling graph

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

(b) Adjacency matrix

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

(c) Degree matrix

$$\begin{pmatrix} 2 & -1 & -1 & 0 \\ -1 & 2 & 0 & -1 \\ -1 & 0 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix}$$

(d) Laplacian matrix

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

(e) Edge incidence matrix

Figure 5.3: Different representations and properties of the coupling map. (a) The coupling graph shows the connectivity between the different physical qubits. Each qubit is connected to two other qubits. (b) The adjacency matrix form A is a commonly used representation for graphs. (c) The degree matrix D indicates the number of edges attached to each vertex. (d) The Laplacian matrix defined as $L = D - A$ is another commonly used representation of a graph. (e) The edge incidence matrix is useful for directed graphs since it encodes the direction of the connection. Each column represents a different edge and each row represents a specific node in the graph.

5.2.3 Hardware representation

The physical constraints of quantum hardware play a crucial role in QCO. These constraints, particularly qubit connectivity for superconducting architectures, are typically represented as graphs. Figure 5.3(a) shows an example of such a coupling graph. Several mathematical representations can capture the structure of these hardware graphs.

A common representation for graphs is as an adjacency matrix A. The adjacency form for the coupling graph is shown in Fig. 5.3(b). This form does not contain any information about the direction of the edges. This can be a problem since certain hardware devices allow only a specific direction and the coupling map is therefore a directed graph. The degree of a node in a graph encodes the number of edges attached to this node. The degree matrix D captures this information for all nodes/vertices; see Fig. 5.3(c). The Laplacian matrix is a combination of the adjacency matrix and the degree matrix and is defined as $L = D - A$, see Fig. 5.3(d). The edge incidence form becomes useful since it can capture the information of a directed graph. This representation is covered in Fig. 5.3(e). Each column represents a different edge and each row represents a specific node in the graph. Thus, for each row there are two non-zero entries encoding the connection.

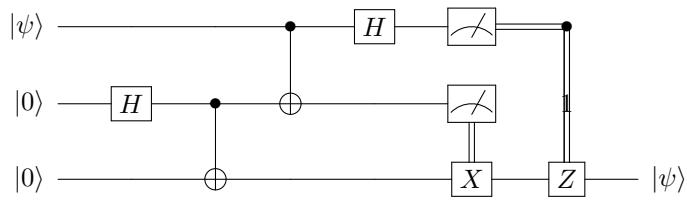


Figure 5.4: The circuit for the quantum teleportation protocol.

These representations provide different perspectives on the hardware's connectivity, each useful for specific aspects of quantum circuit optimization. For instance, the adjacency matrix is helpful for quickly determining direct connections, while the edge incidence matrix can be crucial for optimizing operations on architectures with directional constraints.

5.3 Qubit allocation

Qubit allocation involves mapping virtual qubits to physical qubits while adhering to architectural constraints and optimizing circuit parameters such as depth. In this context, depth refers to the number of computational layers or the longest sequence of dependent operations in the quantum circuit. Consider the four-qubit architecture in Fig. 5.3(a), where vertices and edges represent qubit registers and their connections, respectively. To explore this concept, we implement the quantum teleportation protocol [Fig. 5.4] on this architecture, focusing on the two-qubit gates that are subject to hardware limitations. The protocol's circuit diagram, shown separately, designates qubits as $q0$, $q1$, and $q2$ from top to bottom.

Examining Fig. 5.4, we observe two-qubit interactions between $q1 - q2$ and $q0 - q1$. With these interactions identified, we can proceed to map the qubits onto the hardware registers. Figure 5.5 illustrates two potential qubit allocations for this three-qubit circuit. The configuration in Fig. 5.5(a) requires additional SWAP gates to execute the $q1 - q2$ interaction, whereas Fig. 5.5(b) presents an optimal assignment, allowing all interactions without supplementary gates. This example underscores the impact of initial qubit allocation on circuit performance. However, finding an ideal allocation that satisfies all connectivity requirements is often infeasible, necessitating additional routing procedures. Furthermore, qubit mapping algorithms must account for device noise in current quantum systems. The problem of determining optimal qubit allocation with routing is NP-hard [225, 226], requiring heuristic algorithms [227].

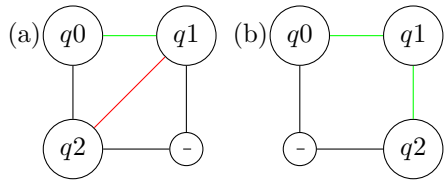


Figure 5.5: Two possible qubit allocations. (a) The assignment of qubit registers does not allow the qubit interaction between $q1$ and $q2$ (red). (b) The allocated qubits allow all the relevant interactions (green). This assignment is optimal in the sense that no additional routing is necessary.

5.4 Qubit routing

In the following, we assume that there exists a non-optimal quantum circuit implementing the unitary of choice without taking the connectivity of the hardware into consideration. The goal is to find a sequence of gates that implements the same unitary while satisfying these hard constraints. Formally, given a coupling graph and a quantum circuit C, the goal is to compile a circuit C' that is functionally equivalent to C and compatible with the coupling graph [226].

This problem is particularly challenging due to its vast search space and the need to minimize the circuit depth ratio (CDR), defined as the ratio of the compiled circuit depth to the initial circuit depth. Ideally, the CDR should be as close to one as possible, though this is often unattainable due to hardware constraints.

Both qubit allocation [225, 226, 228–231] and routing [221, 228, 232–236] have been extensively studied in recent years. Siraichi *et al.* [237] proposed a bounded mapping tree algorithm leveraging advancements in graph theory, decomposing the problem into subgraph isomorphism and token swapping. Li *et al.* [231] introduced SABRE, a SWAP-based BidiRectional heuristic search algorithm. While the qubit allocation problem is intricately linked to routing, the latter can be examined independently. In addition, ML techniques are used to improve the routing procedure [225, 238–241]. In our work, we explore this further by leveraging AlphaZero, an RL algorithm, to optimize the qubit routing process. It has also been suggested to use temporal planners for the routing and allocation problem [230, 242–244].

Having examined methods to optimize quantum circuits for current hardware constraints, we now confront a fundamental challenge in quantum computing: the inevitability of errors. The following chapter introduces QEC, a vital component in the development of scalable and reliable quantum computers. These techniques are essential for overcoming the limitations imposed by quantum decoherence and imperfect operations, paving the way for fault-tolerant quantum computation and play a crucial role in Paper A.

Chapter 6

Quantum error correction

Quantum computers promise to solve certain tasks faster than any classical computer can [15–18]. However, these results assume a "perfect" quantum computer, meaning that the computer is free of errors. Since quantum computers are inherently noisy, we need to understand if they can deal with these errors. The natural way to tackle these errors is by using error correction.

The concept of error correction is nothing new. In classical computing, there is a long history of using error-correcting codes [245, 246]. These codes typically rely on the idea of redundancy, meaning that we encode the relevant information using several bits such that an error on one of the latter does not change the encoded information. The simplest example would be to copy the classical information represented by one bit into n bits. This ensemble of bits represents one logical bit, and the logical state is defined by the majority of its physical bits. Lets say due to some errors $\lfloor \frac{n-1}{2} \rfloor$ bits flip. Then the majority voting system would still allow us to conserve the original state of the bit. Only if more than $\lfloor \frac{n-1}{2} \rfloor$ errors occur does the majority voting system fail. The number of errors leading to a change of the logical state is what is known as the code distance, d.

There exist many different quantum error correction codes and these are commonly describe by a triplet,

$$[[n, k, d]],$$

where n denotes the number of physical qubits, k the number of logical qubits, and d the code distance [247]. Ideally, we want error-correction schemes that require few physical qubits n to encode a large amount of logical qubits k while having a large code distance d.

Contrary to classical error correction, correcting errors in quantum mechanical systems poses some additional challenges. First, we cannot clone an arbitrary quantum state due to the no-cloning theorem [7, 248]. Quantum mechanics rules out the presence of a unitary operation U that changes a known state $|s\rangle$ into

a copy of an unknown arbitrary quantum state $|\psi\rangle$ without altering the latter. Formally, we require the unitary U to satisfy the following relation: $U(|\psi\rangle \otimes |s\rangle) = |\psi\rangle \otimes |\psi\rangle$, where U cannot exist [7, 248]. Second, measuring a quantum state causes its collapse, raising the question how QEC is at all possible if corrections disturb the quantum state. Luckily, we can obtain information about some global property, without causing too much back-action. And finally, there is a continuous set of possible errors due to the analog nature of quantum computers, while digital computers only have bit-flip error sources. We refer the reader to the literature for more details [249–252].

This chapter introduces key concepts in QEC and describes various error-correction schemes, providing essential background for Paper A. In that work, we develop an RL agent to act as a decoder for the toric code.

6.1 The three-qubit bit-flip code

As the name suggests, the three qubit bit-flip code protects the quantum state from bit flips. Assume we want to protect an arbitrary quantum state,

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,. \tag{6.1}$$

We encode the quantum state using three qubits as

$$|\psi_3\rangle = \alpha\,|000\rangle + \beta\,|111\rangle\,. \tag{6.2}$$

This state can be created by applying CNOT gates between the auxilliary qubits and the registers holding the quantum state, as shown in Fig. 6.1. Now assume a bit-flip error occured on qubit 2 such that

$$|\psi_{3,err}\rangle = \alpha\,|010\rangle + \beta\,|101\rangle\,. \tag{6.3}$$

Next, we perform parity measurements on adjacent qubits, meaning Z_1Z_2 (M_{12}) and Z_2Z_3 (M_{23}). These measurements do not affect the quantum state $|\psi\rangle$, since $Z_1Z_2\,|\psi_3\rangle = |\psi_3\rangle = Z_2Z_3\,|\psi_3\rangle$, up to a global phase. The information from the parity measurement helps us to conclude that qubit 2 flipped, and we can then correct that with an X gate, returning to the state $|\psi_3\rangle$.

For an arbitrary X rotation $R_x(\theta)$, we know that $R_x(\theta) = \cos(\theta/2)I - i\sin(\theta/2)X$. Thus, the state we obtain is now a superposition of the correct state and faulty state:

$$\cos(\theta/2)\,|\psi_3\rangle - i\sin(\theta/2)\,|\psi_{3,err}\rangle\,. \tag{6.4}$$

Measuring again the observables Z_1Z_2 and Z_2Z_3 will collapse the superposition and project us either in the state $|\psi_3\rangle$ or $|\psi_{3,err}\rangle$. In both scenarios, whether the state collapses to $|\psi_3\rangle$ or $|\psi_{3,err}\rangle$, the parity measurements M_{12} and M_{23} provide relevant information. If $|\psi_3\rangle$ is obtained, no correction is needed. If $|\psi_{3,err}\rangle$ results, the measurement outcomes pinpoint the flipped qubit, allowing for a

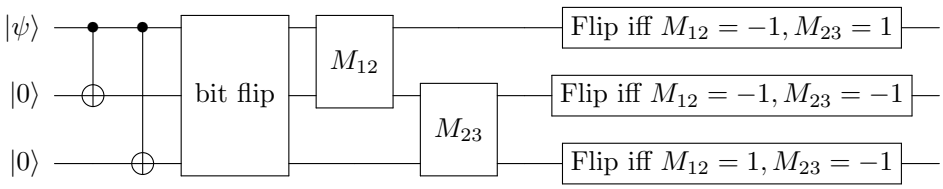


Figure 6.1: The three-qubit bit-flip error-correction code. The CNOT gates generate the state in Eq. (6.2). Once a bit flip occurred, parity measurements guide the error-correction procedure.

targeted X gate correction to restore the original state. This measurement-based feedback ensures we can always recover the error-corrected state [247].

The error-correction protocol introduced can be extended to capture phase flips. The idea is to switch from the computational basis to the Hadamard basis. The protocol is then the same. Thus, we encode our quantum state into the following state

$$|\psi_3\rangle = \alpha\,|+++\rangle + \beta\,|---\rangle\,. \tag{6.5}$$

6.2 Stabilisers

Many QEC codes can be understood in terms of stabilisers. Extending this framework, we define S to be a group of N-qubit operators. If a state is unchanged under the action of a unitary operator U, i.e., $U\,|\psi\rangle = |\psi\rangle$, we say $|\psi\rangle$ is stabilized by U. This concept extends to operators and sets of states. For a group S of N-qubit operators, we define V_S as the set of N-qubit states stabilized by every element in S. Then, S is called the stabilizer of V_S. For V_S to be non-trivial, the elements of S must commute, and $-I$ cannot be in S.

Returning to the three-qubit bit-flip example introduced in Section 6.1, we measured Z_1Z_2 and Z_2Z_3, which retained the encoded state. In this case, V_S is the subspace spanned by $|000\rangle$ and $|111\rangle$, while S is the group $I, Z_1Z_2, Z_2Z_3, Z_1Z_3$ generated by Z_1Z_2 and Z_2Z_3. The additional elements I and Z_1Z_3 result from combining the generators Z_1Z_2 and Z_2Z_3.

Measuring the generators of S enables us to correct specific errors on the stabilized states. This principle can be extended to the design of other error-correction codes by identifying stabilizers, their generators, and the corresponding stabilized states. To resolve which errors a code can protect against, we consider a stabilizer S and error sets E_j that are subgroups of the N-qubit Pauli group G_N, consisting of products of single-qubit Pauli matrices and factors $\pm 1, \pm i$. These errors are correctable if $E_j^\dagger E_k \notin N(S) - S$ for all j, k. Here, $N(S)$, the normalizer of S, is the set of elements $g \in G_N$ that satisfy $gsg^\dagger \in S$ for all $s \in S$, implying that $N(S)$ commutes with S as a set (see Ref. [7] for the proof). In the three-qubit bit-flip code example, it easy to see that any product of two elements from the

error set I, X_1, X_2, X_3 anti-commutes with at least one element in the stabilizer group generated by Z_1Z_2 and Z_2Z_3 (except I, but $I \in S$, so $I \notin N(S) - S$). Consequently, all errors in this set are correctable [247]. There is much more to say about the stabiliser formalism; for more details we refer the reader to the literature [7, 247, 252, 253].

6.3 Surface codes

Surface codes are a promising error-correction scheme to achieve fault-tolerant quantum computation. Recent experiments have shown that these codes can effectively increase the lifetime of quantum states [254], underscoring their relevance. Fundamentally, these codes rely on the concept of repetition codes on a $2d$ lattice. The toric code was the first of these, introduced by Kitaev [249], and played a key role in Paper A. In this work, we build an RL agent acting as a decoder for the toric code.

6.3.1 Toric code

The toric code consists of a two-dimensional quadratic grid of physical qubits with periodic boundary conditions. The $d \times d$ grid consists of $2d^2$ qubits corresponding to a Hilbert space of 2^{2d^2} states, of which four will form the logical code space. This configuration encodes a 4-fold qudit, equivalent to two logical qubits, though we conventionally refer to this as a single logical qubit. The code's architecture is based on the stabilizer formalism. A large set of commuting local parity-check operators (the stabilizers) split the state space into distinct sectors.

The toric code stabilizers are categorized into two types, namely plaquette and vertex operators,

$$A_s = \prod_{j \in \text{star}(s)} X_j, \quad B_p = \prod_{j \in \text{boundary}(p)} Z_j, \tag{6.6}$$

consisting of products of Pauli Z or X operators acting on the four qubits of a plaquette or vertex (see Fig. 6.2), respectively. These operators mutually commute as the star (s) and boundary (p) share either no edges or exactly two edges. The operators A_s and B_p are Hermitian with eigenvalues of 1 and -1, respectively. The logical qubit corresponds to the sector with eigenvalue $+1$ on all stabilizers. We denote a stabilizer with eigenvalue -1 as a plaquette or vertex defect. A single bit flip X or phase flip Z on a state in the qubit sector will produce a pair of defects on neighboring plaquettes or vertices, with Pauli $Y \sim XZ$ giving both pairs of defects, as shown in Fig. 6.2. In Paper A, we consider a depolarizing noise model ($p_x = p_y = p_z$).

The syndrome, a collection of stabilizer defects associated with specific X, Y, or Z operations on the logical state, does not uniquely determine the underlying errors. This ambiguity makes QEC challenging and motivated our exploration

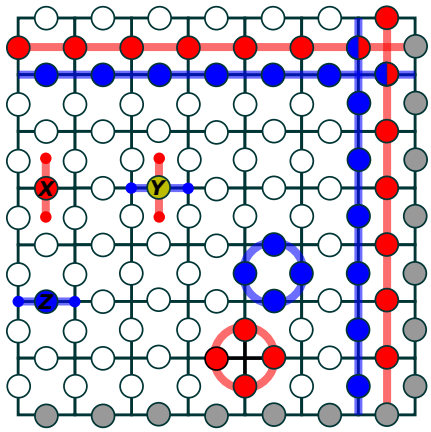


Figure 6.2: A $d = 9$ toric code showing the basic operations. Physical qubits are shown as circles, with shaded areas indicating periodic boundaries. Error types are color-coded: bit flip X (red), phase flip Z (blue), and combined $Y \sim XZ$ (yellow), each generating plaquette and vertex "defects" at error-chain endpoints. These defects are detected using plaquette and vertex parity-check operators, employing $\otimes Z$ and $\otimes X$ respectively. Also shown are logical bit- and phase-flip operators corresponding to closed loops spanning the torus. From Paper A.

of RL techniques in Paper A. Logical operations, which transition between different states in the logical sector, are defined by sequences of X or Z operators that loop around the torus, representing logical bit-flip and phase-flip operations, respectively (see Fig. 6.2). The shortest loop that can enclose the torus has a length of d. The toric code corrects $\lfloor \frac{d-1}{2} \rfloor$ errors. This is easy to see as an error chain longer than $\lfloor \frac{d-1}{2} \rfloor$ and assuming that the shortest correction chain will be employed, will introduce an additional non-trivial loop and change the logic state of the code. The toric code is an example of a topological code, as the logical operations correspond to 'non-contractible' loops on the torus, whereas products of stabilizers can only generate 'contractible' loops.

Chapter 7

Paper overview

This chapter provides a summary of the four papers that form the foundation of this thesis. Research is done best in teams, and each of these papers represents a joint effort. As part of the overview, I will clarify my own contribution to each of the appended papers.

7.1 Paper A: Deep Q-learning decoder for depolarizing noise on the toric code

In Paper A, we address a critical challenge of QEC by developing an RL-based decoding agent for the toric code. Our approach utilizes deep Q-learning, an RL technique (introduced in Chapter 3), to train an NN that encodes state-action Q-values for error-correcting Pauli operations. This methodology allows the decoder to learn and exploit complex correlations between bit-flip and phase-flip errors, a capability that traditional decoders often lack. The theoretical foundations for this work are detailed in Chapters 2, 3, and 6, which cover ML, RL, and QEC, respectively.

We demonstrate that our deep reinforcement learning (DRL)-based decoder outperforms the widely-used minimum-weight-perfect-matching (MWPM) [255, 256] algorithm for code distances up to $d = 9$. Specifically, our decoder achieves higher success rates and error thresholds for depolarizing noise, a common noise model in quantum systems.

Moreover, our decoder exhibits versatility. Not only does it excel in its trained domain of depolarizing noise, but it also shows robust performance across various noise models, including uncorrelated and biased noise (see Chapter 6). This adaptability is a crucial feature for practical QEC, where the exact nature of the noise can vary or be imperfectly characterized.

This work represents one step forward in the application of ML techniques to QEC. By demonstrating that a DRL-based approach can outperform traditional methods, we have opened new avenues for research in this area of quantum computing. This project highlights the potential of RL in tackling complex quantum information processing tasks, suggesting that further exploration of these techniques is warranted. This is exemplified by recent works [257, 258], which combine several concepts related to this thesis, such as transformers or GNNs as decoders for the surface code.

My contributions to this project included implementing the algorithm, training the RL agent, and benchmarking its performance against the graph-based MWPM algorithm, showcasing the practical application of RL in quantum computing.

7.2 Paper B: Applying quantum approximate optimization to the heterogeneous vehicle routing problem

Paper B focuses on the application of hybrid quantum-classical algorithms to optimization problems, as introduced in Chapter 4, where we describe the Ising model and QAOA, the key components of our work. We focus on a specific logistics problem known as the HVRP and develop a mapping for the HVRP to an Ising model as shown in Section 4.3.2, allowing us to apply the framework of hybrid quantum-classical computation to find approximate solutions. Further, we simulate QAOA and determine its performance, i.e., its ability to find the global minimum of the optimization problem. We investigate problem instances consisting of 11, 19, and 21 qubits and use well-established optimization techniques to optimize the quantum circuit.

We analyze the optimization landscape for $p = 1$, where p is the number of layers in the PQC, and find distinct minima for each problem instance. We also simulate QAOA with higher p and see that with increasing circuit depth, the performance increases. However, there is a trade-off that with a growing number of variational parameters: the optimization of these variational parameters becomes increasingly difficult. This can even lead to a shallower circuit reaching a lower energy state than a highly parameterized quantum circuit. We analyze the running time of the four considered optimizers and can observe an exponential scaling. This is in accordance with Ref. [81] considering that the optimization of the energy landscape is NP-hard. Both differential evolution and basinhopping give the best performance, i.e., they find the optimal solution with the highest probability. We investigate whether the variational states generated via the hybrid quantum-classical routine generate states with a high probability of sampling the optimal bitstring. This is desired since we are looking for the best solution to the problem.

We find that constrained optimization problems are particularly challenging

for the QAOA since much work is spent on staying in the subspace of physically viable solutions. The quantum alternating operator ansatz suggests some novel mixers and initial states, but as shown in [38, 40], also these methods are sensitive to the inherent noise of NISQ devices.

In this work, I was involved in all stages, from ideation to implementation and manuscript writing, and conducted all simulations.

7.3 Paper C: Optimizing variational quantum algorithms with qBang

Paper C addresses a critical challenge in the realm of VQAs: the optimization of variational parameters. We introduce the quantum Broyden adaptive natural gradient (qBang) approach, a novel optimization technique designed to enhance VQA performance while minimizing quantum resource requirements. qBang combines the Broyden method [Section 4.2.1] for approximating Fisher information matrix updates with a momentum-based algorithm [Chapter 2]. This interplay allows qBang to navigate difficult optimization landscapes more efficiently than existing methods. The algorithm strikes a balance between computational efficiency and convergence speed, making it particularly well-suited for the noisy intermediate-scale quantum (NISQ) era where quantum resources are limited.

To validate qBang's effectiveness, we conducted extensive benchmarks. These include the barren-plateau problem [202], which tests an optimizer's ability to navigate flat energy landscapes; quantum chemistry simulations, which probe the algorithm's performance in real-world scientific applications; and the max-cut problem, a classic optimization task with relevance to both classical and quantum computing. Across these varied scenarios, qBang demonstrated stable performance and advantages over existing techniques. Notably, it showed particular strength in scenarios featuring flat (but not exponentially flat) optimization landscapes due to faster converges compared to its competitors. This work not only provides a powerful new tool for optimizing VQAs, including potential applications to our Ising formulation from Paper B, but also establishes a new development strategy for gradient-based VQAs with potential for further improvements. The theoretical foundations underlying this study are elaborated in detail in Chapters 2 and 4.

My role in this project was comprehensive, encompassing all stages from initial concept development to implementation and manuscript preparation, including the design and execution of all simulations.

7.4 Paper D: RydbergGPT

In Paper D, we introduce RydbergGPT, a generative pretrained transformer designed to predict measurement outcomes of neutral-atom-array quantum com-

puters. This work applies ML techniques from Chapter 2 to the emerging field of Rydberg atom arrays, which are a promising platform for quantum information processing.

In these systems, qubits are encoded using the electronic ground state $|g\rangle$ and excited (Rydberg) state $|r\rangle$ of individual atoms arranged in a lattice structure [259–263]. In our work, we consider a system of $N = L \times L$ atoms arranged on a square lattice. The governing Hamiltonian defining the Rydberg atom array interactions has the following form:

$$\hat{H} = \frac{\Omega}{2} \sum_{i=1}^{N} \hat{\sigma}_i^x - \delta \sum_{i=1}^{N} \hat{n}_i + \sum_{i,j} V_{ij} \hat{n}_i \hat{n}_j, \tag{7.1}$$

where $\hat{\sigma}_i^x = |g\rangle_i\langle r|_i + |r\rangle_i\langle g|_i$, the occupation number operator $\hat{n}_i = \frac{1}{2}(\hat{\sigma}_i + \mathbb{1}) = |r\rangle_i\langle r|_i$, $\hat{\sigma}_i = |r\rangle_i\langle r|_i - |g\rangle_i\langle g|_i$, and $V_{ij} = \Omega R_b^6/|\vec{r}_i - \vec{r}_j|^6$. The experimental settings of a Rydberg atom array are controlled by the detuning from resonance δ, the Rabi frequency Ω, the lattice length scale a, and the positions of the atoms $\{\mathbf{r}_i\}_i^N$. We obtain a symmetric matrix $\mathbf{V}$, which encapsulates the relevant information about the lattice geometry, and derive the Rydberg blockade radius R_b, within which simultaneous excitations are penalized [96, 259, 260].

Our encoder-decoder architecture, based on a transformer model, as introduced in Section 2.2.2, takes the interacting Hamiltonian as input and generates an autoregressive sequence of qubit measurement probabilities, as detailed in Section 2.3. We demonstrate the effectiveness of the architecture by studying its performance near a quantum phase transition in Rydberg atoms arranged in a square lattice [264]. The model's generalization capabilities are explored by predicting ground-state measurements for Hamiltonian parameters not included in the training set. Notably, we found that our model accurately reproduces physical estimators for the ground state, but shows limitations when applied to thermally mixed states at higher temperatures. This work not only provides benchmarks for scaling larger RydbergGPT models, but also introduces an open-source framework to facilitate the development of foundation models for various quantum computer interactions and datasets.

My contributions to this project included designing and building the transformer architecture, training the model, and collaboratively writing the manuscript.

Chapter 8

Conclusion

The intersection of quantum computing and machine learning has been the focus of this thesis. We followed a hybrid path, examining both hybrid classical-quantum algorithms and the application of machine learning to the quantum computing stack. The hybrid classical-quantum algorithms and their optimization methods are fundamental to our work in Papers B and C.

In Paper B, we investigate the application of the QAOA to the HVRP, an industrially relevant logistics problem introduced in Chapter 4. We present a novel Ising mapping for the HVRP and simulate it for simplified instances. Our results reveal that obtaining the optimal solution with high probability is challenging. A key issue is that the Hamiltonian's energetics are dominated by constraints, causing the algorithm to prioritize constraint satisfaction over cost optimization. This leads to a scenario where the algorithm expends most of its computational resources maintaining feasible solutions, rather than optimizing within the subspace of physically viable solutions.

Furthermore, we note that modern high-performance optimizers (classical computers) for the HVRP can solve problem instances with more than 1,000 customers [265, 266]. For a quantum computer to solve problem instances of this size, it would need at least millions of controllable qubits with millions of multi-qubit interactions.

The qBang optimizer, introduced in Paper C, represents an advance in the optimization of VQAs. By combining the Broyden approach to approximate the QFIM updates with momentum-based techniques, qBang effectively navigates flat energy landscapes while reducing quantum resource requirements. The success of qBang opens up new avenues for gradient-based VQA optimization, suggesting numerous potential improvements and applications. Future research could explore its adaptability to different quantum computing architectures, its scalability with increasing qubit numbers, and its integration with error-mitigation

techniques. Furthermore, the principles underlying qBang may inspire further innovations in quantum-classical hybrid algorithms.

Looking ahead, future research should build upon these findings by focusing on benchmarking quantum devices with larger optimization problems. As quantum hardware advances, tackling more complex tasks becomes feasible. A critical aspect of this progression will be comparing the time-to-solution of quantum approaches against classical optimizers. This benchmarking process will provide valuable insights into the computational advantages of quantum devices and help identify specific areas where quantum computing offers significant benefits. Such comparisons will not only guide the development of future quantum algorithms and hardware, but also offer a more nuanced understanding of the relative strengths of quantum and classical computing paradigms.

In Paper D, we introduced RydbergGPT, a generative pretrained transformer that learns and reproduces measurement outcomes from neutral atom array quantum computers. This model demonstrates the potential of machine learning in quantum state prediction, effectively generalizing to unseen Hamiltonian parameters near quantum phase transitions.

Looking ahead, machine learning approaches offer promising solutions to the challenge of exponentially increasing Hilbert-space dimensions in quantum systems. In this context, NQSs, introduced in Section 2.3, present an interesting avenue to represent complex quantum systems efficiently. However, the interpretability of these models remains a challenge. Future research could explore symbolic machine learning for more interpretable quantum state representations. Furthermore, leveraging machine learning to optimize measurement strategies in quantum experiments could significantly reduce complexity and cost. These developments may uncover new synergies between quantum physics and machine learning, potentially leading to breakthroughs in both fields and inspiring novel quantum algorithms and experimental designs.

RL holds immense promise for automating and optimizing various aspects of the quantum computing stack. As demonstrated in Paper A with quantum error correction, RL can be extended to address other critical challenges in quantum computing. Future applications could include automated calibration of quantum devices, where RL agents learn to fine-tune system parameters for optimal performance. In quantum circuit optimization, RL algorithms could discover more efficient circuit designs, potentially reducing gate counts and execution times. Moreover, RL could play a crucial role in the adaptive control of quantum systems, real-time error mitigation, and even in the design optimization of quantum hardware components. By leveraging RL's ability to learn complex strategies through interaction, we can envision a more self-optimizing and robust quantum computing ecosystem. This approach not only has the potential to enhance the performance and reliability of quantum systems but also to accelerate the development of practical, large-scale quantum computers.

Bibliography

[1] R. F. W. Bader, "A quantum theory of molecular structure and its applications", Chemical Reviews **91**, 893-928 (1991).

[2] E. Morse, *Nuclear Fusion*, Graduate Texts in Physics (Springer International Publishing, Cham, 2018).

[3] R. Combescot, *Superconductivity: An Introduction* (Cambridge University Press, Cambridge, England, UK, 2022).

[4] A. Sudbery, *Quantum Mechanics and the Particles of Nature: An Outline for Mathematicians* (Cambridge University Press, Cambridge, England, UK, 1986).

[5] P. 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**, 563–591 (1980).

[6] D. Deutsch, "Quantum theory, the Church–Turing principle and the universal quantum computer", Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences **400**, 97–117 (1985).

[7] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Vol. 66 (Cambridge University Press, Cambridge, 2010).

[8] R. P. Feynman, "Simulating physics with computers", International Journal of Theoretical Physics **21**, 467–488 (1982).

[9] I. M. Georgescu, S. Ashhab, and F. Nori, "Quantum simulation", Reviews of Modern Physics **86**, 153–185 (2014).

[10] S. Lloyd, "Universal Quantum Simulators", Science **273**, 1073–1078 (1996).

[11] R. Wiersema, C. Zhou, Y. de Sereville, J. F. Carrasquilla, Y. B. Kim, and H. Yuen, "Exploring Entanglement and Optimization within the Hamiltonian Variational Ansatz", PRX Quantum **1**, 020319 (2020).

[12] J. Tilly, H. Chen, S. Cao, D. Picozzi, K. Setia, *et al.*, "The Variational Quantum Eigensolver: A review of methods and best practices", Physics Reports **986**, 1–128 (2022).

[13] C. Cade, L. Mineh, A. Montanaro, and S. Stanisic, "Strategies for solving the Fermi-Hubbard model on near-term quantum computers", Physical Review B **102**, 235122 (2020).

[14] C.-Y. Park, "Efficient ground state preparation in variational quantum eigensolver with symmetry-breaking layers", APL Quantum **1**, 10.1063/5.0186205 (2024).

[15] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke, W.-K. Mok, S. Sim, L.-C. Kwek, and A. Aspuru-Guzik, "Noisy intermediate-scale quantum algorithms", Reviews of Modern Physics **94**, 015004 (2022).

[16] A. Montanaro, "Quantum algorithms: An overview", npj Quantum Information **2**, 15023 (2016).

[17] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum Algorithm for Linear Systems of Equations", Physical Review Letters **103**, 150502 (2009).

[18] A. M. Dalzell, S. McArdle, M. Berta, P. Bienias, C.-F. Chen, A. Gilyén, C. T. Hann, M. J. Kastoryano, E. T. Khabiboulline, A. Kubica, G. Salton, S. Wang, and F. G. S. L. Brandão, "Quantum algorithms: A survey of applications and end-to-end complexities", arXiv (2023), 2310.03011 .

[19] P. Shor, in *Proceedings 35th Annual Symposium on Foundations of Computer Science* (IEEE Comput. Soc. Press, 1994) pp. 124–134.

[20] V. Mavroeidis, K. Vishi, M. D. Zych, and A. Jøsang, "The Impact of Quantum Computing on Present Cryptography", International Journal of Advanced Computer Science and Applications **9**, 405–414 (2018).

[21] M. Kaplan, G. Leurent, A. Leverrier, and M. Naya-Plasencia, in *Advances in Cryptology – CRYPTO 2016*, edited by M. Robshaw and J. Katz (Springer Berlin Heidelberg, Berlin, Heidelberg, 2016) pp. 207–237.

[22] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles, "Variational quantum algorithms", Nature Reviews Physics **3**, 625–644 (2021).

[23] M. Cerezo, G. Verdon, H.-Y. Huang, L. Cincio, and P. J. Coles, "Challenges and opportunities in quantum machine learning", Nature Computational Science **2**, 567–576 (2022).

[24] M. Schuld and F. Petruccione, "Quantum ensembles of quantum classifiers", Scientific Reports **8**, 2772 (2018).

[25] V. Saggio, B. Asenbeck, A. Hamann, T. Strömberg, P. Schiansky, V. Dunjko, N. Friis, N. Harris, M. Hochberg, D. Englund, S. Wölk, H. Briegel, and P. Walther, in *Quantum Nanophotonic Materials, Devices, and Systems 2021*, edited by M. Agio, C. Soci, and M. T. Sheldon (SPIE, San Diego, United States, 2021) p. 18.

[26] C. Blank, D. K. Park, J. K. K. Rhee, and F. Petruccione, "Quantum classifier with tailored quantum kernel", npj Quantum Information **6**, 41 (2020).

[27] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, "Supervised learning with quantum-enhanced feature spaces", Nature **567**, 209–212 (2019).

[28] M. Schuld and N. Killoran, "Quantum Machine Learning in Feature Hilbert Spaces", Physical Review Letters **122**, 040504 (2019).

[29] V. Dunjko, J. M. Taylor, and H. J. Briegel, in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (IEEE, Banff, AB, 2017) pp. 282–287.

[30] M. Schuld, A. Bocharov, K. M. Svore, and N. Wiebe, "Circuit-centric quantum classifiers", Physical Review A **101**, 032308 (2020).

[31] Y. Sun, J.-Y. Zhang, M. S. Byrd, and L.-A. Wu, "Adiabatic Quantum Simulation Using Trotterization", arXiv:1805.11568 (2018).

[32] Q. Gao, J. M. Garcia, N. Yamamoto, H. Nakamura, T. P. Gujarati, G. O. Jones, J. E. Rice, S. P. Wood, M. Pistoia, and N. Yamamoto, "Computational investigations of the lithium superoxide dimer rearrangement on noisy quantum devices", Journal of Physical Chemistry A **125**, 1827–1836 (2021).

[33] K. Meichanetzidis, S. Gogioso, G. de Felice, N. Chiappori, A. Toumi, and B. Coecke, "Quantum Natural Language Processing on Near-Term Quantum Computers", Electronic Proceedings in Theoretical Computer Science **340**, 213–229 (2021).

[34] E. R. Miranda, R. Yeung, A. Pearson, K. Meichanetzidis, and B. Coecke, in *Quantum Computer Music: Foundations, Methods and Advanced Concepts* (Springer, Cham, Switzerland, 2022) pp. 313–356.

[35] B. Coecke, G. de Felice, K. Meichanetzidis, and A. Toumi, "Foundations for Near-Term Quantum Natural Language Processing", arXiv:2012.03755 (2020).

[36] D. Kartsaklis, I. Fan, R. Yeung, A. Pearson, R. Lorenz, A. Toumi, G. de Felice, K. Meichanetzidis, S. Clark, and B. Coecke, "Lambeq: An Efficient High-Level Python Library for Quantum NLP", arXiv:2110.04236 (2021).

[37] C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits", Quantum **5**, 433 (2021).

[38] Z. Wang, N. C. Rubin, J. M. Dominy, and E. G. Rieffel, "XY mixers: Analytical and numerical results for the quantum alternating operator ansatz", Physical Review A **101**, 012320 (2020).

[39] S. Hadfield, Z. Wang, B. O'Gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, "From the quantum approximate optimization algorithm to a quantum alternating operator ansatz", Algorithms **12**, 34 (2019).

[40] M. Streif, M. Leib, F. Wudarski, E. Rieffel, and Z. Wang, "Quantum algorithms with local particle-number conservation: Noise effects and error correction", Physical Review A **103**, 042412 (2021).

[41] S. Yarkoni, A. Alekseyenko, M. Streif, D. Von Dollen, F. Neukart, *et al.*, *Multi-car paint shop optimization with quantum annealing* (IEEE Computer Society, 2021).

[42] D. Amaro, M. Rosenkranz, N. Fitzpatrick, K. Hirano, and M. Fiorentini, "A case study of variational quantum algorithms for a job shop scheduling problem", EPJ Quantum Technology **9**, 1–20 (2022).

[43] L. K. Grover, in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing - STOC '96* (ACM Press, Philadelphia, Pennsylvania, United States, 1996) pp. 212–219.

[44] A. K. Ekert, "Quantum cryptography based on Bell's theorem", Physical Review Letters **67**, 661–663 (1991).

[45] D. Bouwmeester, J.-W. Pan, K. Mattle, M. Eibl, H. Weinfurter, and A. Zeilinger, "Experimental quantum teleportation", Nature **390**, 575–579 (1997).

[46] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O'Brien, "Quantum computers", Nature **464**, 45–53 (2010).

[47] I. Buluta, S. Ashhab, and F. Nori, "Natural and artificial atoms for quantum computation", Reports on Progress in Physics **74**, 104401 (2011).

[48] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play", Science **362**, 1140-1144 (2018).

[49] M. Alansari, O. A. Hay, S. Javed, A. Shoufan, Y. Zweiri, and N. Werghi, "GhostFaceNets: Lightweight Face Recognition Model From Cheap Operations", IEEE Access **11**, 35429–35446 (2023).

[50] T. Mare, G. Duta, M.-I. Georgescu, A. Sandru, B. Alexe, M. Popescu, and R. T. Ionescu, "A realistic approach to generate masked faces applied on two novel masked face recognition data sets", arXiv (2021), 2109.01745 .

[51] W. Huang, X. Wu, Q. Zhang, N. Wu, and Z. Song, in *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)* (IEEE) pp. 10–12.

[52] M. Reda, A. Onsy, A. Y. Haikal, and A. Ghanbari, "Path planning algorithms in the autonomous driving system: A comprehensive review", Robotics and Autonomous Systems **174**, 104630 (2024).

[53] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners" (2019).

[54] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", ArXiv e-prints (2018), 1810.04805 .

[55] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning", Nature **518**, 529–533 (2015).

[56] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search", Nature **529**, 484–489 (2016).

[57] M. Innes, A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V. B. Shah, and W. Tebbutt, "A Differentiable Programming System to Bridge Machine Learning and Scientific Computing", arXiv (2019), 1907.07587 .

[58] F. Sapienza, J. Bolibar, F. Schäfer, B. Groenke, A. Pal, V. Boussange, P. Heimbach, G. Hooker, F. Pérez, P.-O. Persson, and C. Rackauckas, "Differentiable Programming for Differential Equations: A Review", arXiv (2024), 2406.09699 .

[59] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural Ordinary Differential Equations", arXiv (2018), 1806.07366 .

[60] A. Dawid, J. Arnold, B. Requena, A. Gresch, M. Płodzień, K. Donatella, K. A. Nicoli, P. Stornati, R. Koch, M. Büttner, R. Okuła, G. Muñoz-Gil, R. A. Vargas-Hernández, A. Cervera-Lierta, J. Carrasquilla, V. Dunjko, M. Gabrié, P. Huembeli, E. van Nieuwenburg, F. Vicentini, L. Wang, S. J. Wetzel, G. Carleo, E. Greplová, R. Krems, F. Marquardt, M. Tomza, M. Lewenstein, and A. Dauphin, "Modern applications of machine learning in quantum sciences", arXiv (2022), 2204.04198 .

[61] H.-J. Liao, J.-G. Liu, L. Wang, and T. Xiang, "Differentiable Programming Tensor Networks", Phys. Rev. X **9**, 031041 (2019).

[62] B.-B. Chen, Y. Gao, Y.-B. Guo, Y. Liu, H.-H. Zhao, H.-J. Liao, L. Wang, T. Xiang, W. Li, and Z. Y. Xie, "Automatic differentiation for second renormalization of tensor networks", Phys. Rev. B **101**, 220409 (2020).

[63] J. Ingraham, A. Riesselman, C. Sander, and D. Marks, "Learning Protein Structure with a Differentiable Simulator", OpenReview (2018).

[64] S. S. Schoenholz and E. D. Cubuk, "JAX, M.D.: A Framework for Differentiable Physics", arXiv (2019), 1912.04232 .

[65] T. Tamayo-Mendoza, C. Kreisbeck, R. Lindh, and A. Aspuru-Guzik, "Automatic Differentiation in Quantum Chemistry with Applications to Fully Variational Hartree–Fock", ACS Cent. Sci. **4**, 559–566 (2018).

[66] A. S. Abbott, B. Z. Abbott, J. M. Turney, and H. F. I. Schaefer, "Arbitrary-Order Derivatives of Quantum Chemical Methods via Automatic Differentiation", J. Phys. Chem. Lett. **12**, 3232–3239 (2021).

[67] M. F. Kasim, S. Lehtola, and S. M. Vinko, "DQC: A Python program package for differentiable quantum chemistry", The Journal of Chemical Physics **156**, 084801 (2022).

[68] X. Zhang and G. K.-L. Chan, "Differentiable quantum chemistry with PySCF for molecules and materials at the mean-field level and beyond", The Journal of Chemical Physics **157**, 204801 (2022).

[69] M. Bukov, A. G. Day, D. Sels, P. Weinberg, A. Polkovnikov, and P. Mehta, "Reinforcement Learning in Different Phases of Quantum Control", Physical Review X **8**, 031086 (2018), arXiv:1705.00565 .

[70] M. Dalgaard, F. Motzoi, J. J. Sørensen, and J. Sherson, "Global optimization of quantum dynamics with AlphaZero deep exploration", npj Quantum Information **6**, 6 (2020), 1907.05672 .

[71] D. Beloborodov, A. E. Ulanov, J. N. Foerster, S. Whiteson, and A. I. Lvovsky, "Reinforcement learning enhanced quantum-inspired algorithm for combinatorial optimization", Machine Learning: Science and Technology **2**, 025009 (2021).

[72] S. Giordano and M. A. Martin-Delgado, "Reinforcement-learning generation of four-qubit entangled states", Physical Review Research **4**, 043056 (2022).

[73] N. Leung, M. Abdelhafez, J. Koch, and D. Schuster, "Speedup for quantum optimal control from automatic differentiation based on graphics processing units", Phys. Rev. A **95**, 042318 (2017).

[74] F. Schäfer, M. Kloc, C. Bruder, and N. Lörch, "A differentiable programming method for quantum control", Machine Learning: Science and Technology **1**, 035009 (2020).

[75] X.-Z. Luo, J.-G. Liu, P. Zhang, and L. Wang, "Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design", Quantum **4**, 341 (2020), 1912.10877v3 .

[76] O. Kyriienko, A. E. Paine, and V. E. Elfving, "Solving nonlinear differential equations with differentiable quantum circuits", Physical Review A **103**, 052416 (2021).

[77] V. E. Elfving, B. W. Broer, M. Webber, J. Gavartin, M. D. Halls, K. P. Lorton, and A. Bochevarov, "How will quantum computers provide an industrially relevant computational advantage in quantum chemistry?" (2020).

[78] S. Mensa, E. Sahin, F. Tacchino, P. K. Barkoutsos, and I. Tavernelli, "Quantum machine learning framework for virtual screening in drug discovery: a prospective quantum advantage", Machine Learning: Science and Technology **4**, 015023 (2023).

[79] S. McArdle, T. Jones, S. Endo, Y. Li, S. C. Benjamin, and X. Yuan, "Variational ansatz-based quantum simulation of imaginary time evolution", npj Quantum Information **5**, 75 (2019).

[80] J. Yao, M. Bukov, and L. Lin, in *Mathematical and Scientific Machine Learning* (PMLR, 2020) pp. 605–634.

[81] L. Bittel and M. Kliesch, "Training Variational Quantum Algorithms Is NP-Hard", Physical Review Letters **127**, 120502 (2021).

[82] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", Nature **323**, 533–536 (1986).

[83] J. Carrasquilla and R. G. Melko, "Machine learning phases of matter", Nature Physics **13**, 431–434 (2017).

[84] S. J. Wetzel, "Unsupervised learning of phase transitions: From principal component analysis to variational autoencoders", Physical Review E **96**, 022140 (2017).

[85] K. Kottmann, P. Huembeli, M. Lewenstein, and A. Acín, "Unsupervised Phase Discovery with Deep Anomaly Detection", Physical Review Letters **125**, 170603 (2020).

[86] E. P. L. Van Nieuwenburg, Y.-H. Liu, and S. D. Huber, "Learning phase transitions by confusion", Nature Physics **13**, 435–439 (2017).

[87] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]", IEEE Signal Process. Mag. **29**, 141–142 (2012).

[88] S. Ahmed, *Machine Learning for Quantum Information and Computing*, Phd thesis, Chalmers University of Technology, Göteborg (2023).

[89] Bernard. W. Silverman, *Density Estimation for Statistics and Data Analysis* (Taylor & Francis, Andover, England, UK, 2017).

[90] S. Ahmed, C. Sánchez Muñoz, F. Nori, and A. F. Kockum, "Quantum State Tomography with Conditional Generative Adversarial Networks", Physical Review Letters **127**, 140502 (2021).

[91] S. Ahmed, F. Quijandría, and A. F. Kockum, "Gradient-Descent Quantum Process Tomography by Learning Kraus Operators", Physical Review Letters **130**, 150402 (2023).

[92] H. Wang, M. Weber, J. Izaac, and C. Y.-Y. Lin, "Predicting Properties of Quantum Systems with Conditional Generative Models", arxiv:2211.16943 [quant-ph] (2022).

[93] Y.-H. Zhang and M. Di Ventra, "Transformer quantum state: A multipurpose model for quantum many-body problems", Physical Review B **107** (2023).

[94] M. Hibat-Allah, M. Ganahl, L. E. Hayward, R. G. Melko, and J. Carrasquilla, "Recurrent neural network wave functions", Physical Review Research **2**, 023358 (2020).

[95] S. Czischek, M. S. Moss, M. Radzihovsky, E. Merali, and R. G. Melko, "Data-Enhanced Variational Monte Carlo Simulations for Rydberg Atom Arrays", Physical Review B **105**, 205108 (2022), arxiv:2203.04988 .

[96] M. S. Moss, S. Ebadi, T. T. Wang, G. Semeghini, A. Bohrdt, M. D. Lukin, and R. G. Melko, "Enhancing variational monte carlo simulations using a programmable quantum simulator", Physical Review A **109**, 032410 (2024).

[97] K. Sprague and S. Czischek, "Variational Monte Carlo with large patched transformers", Communications Physics **7**, 90 (2024).

[98] J. Gui, T. Chen, J. Zhang, Q. Cao, Z. Sun, H. Luo, and D. Tao, "A survey on self-supervised learning: Algorithms, applications, and future trends", IEEE Transactions on Pattern Analysis and Machine Intelligence , 1-20 (2024).

[99] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", bulletin of mathematical biophysics **5**, 115–133 (1943).

[100] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)", arXiv (2016), 1606.08415 .

[101] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, Cambridge, MA, USA, 2016) `http://www.deeplearningbook.org`.

[102] O. Triebe, N. Laptev, and R. Rajagopal, "AR-Net: A simple Auto-Regressive Neural Network for time-series", arXiv (2019), 1911.12436 .

[103] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate", arXiv:1409.0473 [cs.CL] (2016).

[104] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation", arXiv:1406.1078 [cs.CL] (2014).

[105] J. Carrasquilla and G. Torlai, "How to use neural networks to investigate quantum many-body physics", PRX Quantum **2**, 040201 (2021).

[106] S. Morawetz, I. J. S. De Vlugt, J. Carrasquilla, and R. G. Melko, "U(1)-symmetric recurrent neural networks for quantum state reconstruction", Physical Review A **104**, 012401 (2021).

[107] M. Hibat-Allah, R. G. Melko, and J. Carrasquilla, "Investigating topological order using recurrent neural networks", Physical Review B **108**, 075152 (2023).

[108] M. Hibat-Allah, E. M. Inack, R. Wiersema, R. G. Melko, and J. Carrasquilla, "Variational neural annealing", Nature Machine Intelligence **3**, 952–961 (2021).

[109] M. Hibat-Allah, R. G. Melko, and J. Carrasquilla, "Supplementing recurrent neural network wave functions with symmetry and annealing to improve accuracy", arXiv:2207.14314 [cond-mat.dis-nn] (2024).

[110] J. Carrasquilla, D. Luo, F. Pérez, A. Milsted, B. K. Clark, M. Volkovs, and L. Aolita, "Probabilistic simulation of quantum circuits using a deep-learning architecture", Physical Review A **104**, 032610 (2021).

[111] P. Cha, P. Ginsparg, F. Wu, J. Carrasquilla, P. L. McMahon, and E.-A. Kim, "Attention-based Quantum Tomography", Machine Learning: Science and Technology **3**, 01LT01 (2022).

[112] L. L. Viteritti, R. Rende, and F. Becca, "Transformer variational wave functions for frustrated quantum spin systems", Physical Review Letters **130**, 236401 (2023).

[113] L. L. Viteritti, R. Rende, A. Parola, S. Goldt, and F. Becca, "Transformer wave function for the shastry-sutherland model: emergence of a spin-liquid phase", arXiv:2311.16889 [cond-mat.str-el] (2024).

[114] H. Lange, G. Bornet, G. Emperauger, C. Chen, T. Lahaye, S. Kienle, A. Browaeys, and A. Bohrdt, "Transformer neural networks and quantum simulators: a hybrid approach for simulating strongly correlated systems", arXiv:2406.00091 [cond-mat.dis-nn] (2024).

[115] B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle, "Neural Autoregressive Distribution Estimation", Journal of Machine Learning Research **17**, 1–37 (2016).

[116] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, in *Advances in Neural Information Processing Systems*, Vol. 2017-Decem (2017) pp. 5999–6009, arxiv:1706.03762 .

[117] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", arXiv:1502.03044 [cs] (2016).

[118] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation", arXiv:1508.04025 [cs] (2015).

[119] K. He, X. Zhang, S. Ren, and J. Sun, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016) pp. 770–778.

[120] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization", arXiv:1607.06450 [stat.ML] (2016).

[121] L. Wu, P. Cui, J. Pei, and L. Zhao, eds., *Graph Neural Networks: Foundations, Frontiers, and Applications* (Springer Nature Singapore, Singapore, 2022).

[122] A. Daigavane, B. Ravindran, and G. Aggarwal, "Understanding Convolutions on Graphs", Distill **6**, e32 (2021).

[123] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko, "A Gentle Introduction to Graph Neural Networks", Distill **6**, e33 (2021).

[124] T. N. Kipf and M. Welling, in *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (2017) arxiv:1609.02907 .

[125] G. Carleo and M. Troyer, "Solving the quantum many-body problem with artificial neural networks", Science **355**, 602-606 (2017).

[126] H. Lange, A. Van de Walle, A. Abedinnia, and A. Bohrdt, "From Architectures to Applications: A Review of Neural Quantum States", arXiv (2024), 2402.09402 .

[127] E. Merali, I. J. S. De Vlugt, and R. G. Melko, "Stochastic Series Expansion Quantum Monte Carlo for Rydberg Arrays", arxiv:2107.00766 (2023).

[128] "Understanding the Metropolis-Hastings Algorithm on JSTOR", [Online; accessed 6. Sep. 2024] (1995).

[129] X. He, F. Xue, X. Ren, and Y. You, "Large-scale deep learning optimizations: A comprehensive survey", arXiv:2111.00856 [cs.LG] (2021).

[130] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", arXiv (2014), 1412.6980 .

[131] C. J. Li, A. Yuan, G. Gidel, Q. Gu, and M. I. Jordan, in *International Conference on Machine Learning* (2022).

[132] Y. Nesterov, "A method for solving the convex programming problem with convergence rate O(1/kˆ2)", Proceedings of the USSR Academy of Sciences (1983).

[133] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, in *Proceedings of the 30th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 28, edited by S. Dasgupta and D. McAllester (PMLR, Atlanta, Georgia, USA, 2013) pp. 1139–1147.

[134] S. Ruder, "An overview of gradient descent optimization algorithms", arXiv:1609.04747 [cs.LG] (2017).

[135] J. Domke, "Automatic Differentiation and Neural Networks" (2011).

[136] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, second edition ed., Adaptive Computation and Machine Learning Series (The MIT Press).

[137] F. J. R. Ruiz, T. Laakkonen, J. Bausch, M. Balog, M. Barekatain, F. J. H. Heras, A. Novikov, N. Fitzpatrick, B. Romera-Paredes, J. van de Wetering, A. Fawzi, K. Meichanetzidis, and P. Kohli, "Quantum Circuit Optimization with AlphaTensor", arXiv (2024), 2402.14396 .

[138] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli, "Discovering faster matrix multiplication algorithms with reinforcement learning", Nature **610**, 47–53 (2022).

[139] D. J. Mankowitz, A. Michi, A. Zhernov, M. Gelmi, M. Selvi, C. Paduraru, E. Leurent, S. Iqbal, J.-B. Lespiau, A. Ahern, T. Köppe, K. Millikin, S. Gaffney, S. Elster, J. Broshear, C. Gamble, K. Milan, R. Tung, M. Hwang, T. Cemgil, M. Barekatain, Y. Li, A. Mandhane, T. Hubert, J. Schrittwieser, D. Hassabis, P. Kohli, M. Riedmiller, O. Vinyals, and D. Silver, "Faster sorting algorithms discovered using deep reinforcement learning", Nature **618**, 257–263 (2023).

[140] C. J. C. H. Watkins and P. Dayan, "Q-learning", Machine Learning **8**, 279–292 (1992).

[141] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay", arXiv:1511.05952 [cs] (2016).

[142] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge", Nature **550**, 354–359 (2017).

[143] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering Atari, Go, chess and shogi by planning with a learned model", Nature **588**, 604–609 (2020).

[144] T.-R. Wu, H. Guei, P.-C. Peng, P.-W. Huang, T. H. Wei, C.-C. Shih, and Y.-J. Tsai, "MiniZero: Comparative Analysis of AlphaZero and MuZero on Go, Othello, and Atari Games", arXiv (2023), 2310.11305 .

[145] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, in *International Conference on Machine Learning* (PMLR, 2013) pp. 1058–1066.

[146] A. Krogh and J. Hertz, "A Simple Weight Decay Can Improve Generalization", Advances in Neural Information Processing Systems **4** (1991).

[147] J. Preskill, "Quantum computing in the NISQ era and beyond", Quantum **2**, 79 (2018).

[148] W. Lavrijsen, A. Tudor, J. Muller, C. Iancu, and W. De Jong, "Classical Optimizers for Noisy Intermediate-Scale Quantum Devices", Proceedings - IEEE International Conference on Quantum Computing and Engineering, QCE 2020 , 267–277 (2020).

[149] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization", The Computer Journal **7**, 308–313 (1965).

[150] M. J. D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives", The Computer Journal **7**, 155–162 (1964).

[151] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces", Journal of Global Optimization **11**, 341–359 (1996).

[152] D. Wales and J. Doye, "Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms", The Journal of Physical Chemistry A **101**, 5111–5116 (1998).

[153] M. Wahde, "Biologically inspired optimization methods: An introduction", Choice Reviews Online **46**, 46-3899-46-3899 (2009).

[154] M. Alam, A. Ash-Saki, and S. Ghosh, in *Proceedings of the 2020 Design, Automation and Test in Europe Conference and Exhibition, DATE 2020* (2020) pp. 686–689.

[155] G. Verdon, M. Broughton, J. R. McClean, K. J. Sung, R. Babbush, Z. Jiang, H. Neven, and M. Mohseni, "Learning to learn with quantum neural networks via classical neural networks", arXiv:1907.05415 (2019).

[156] M. M. Wauters, E. Panizon, G. B. Mbeng, and G. E. Santoro, "Reinforcement-learning-assisted quantum optimization", Physical Review Research **2**, 033446 (2020).

[157] A. Garcia-Saez and J. Riu, "Quantum Observables for continuous control of the Quantum Approximate Optimization Algorithm via Reinforcement Learning", arXiv:1911.09682 (2019).

[158] S. Khairy, R. Shaydulin, L. Cincio, Y. Alexeev, and P. Balaprakash, "Learning to Optimize Variational Quantum Circuits to Solve Combinatorial Problems", Proceedings of the AAAI Conference on Artificial Intelligence **34**, 2367–2375 (2020).

[159] D. Beloborodov, A. E. Ulanov, J. N. Foerster, S. Whiteson, and A. I. Lvovsky, "Reinforcement learning enhanced quantum-inspired algorithm for combinatorial optimization", Machine Learning: Science and Technology **2**, 025009 (2021).

[160] M. Y. Niu, S. Boixo, V. N. Smelyanskiy, and H. Neven, "Universal quantum control through deep reinforcement learning", npj Quantum Information **5**, 33 (2019).

[161] C. Cao, Z. An, S.-Y. Hou, D. L. Zhou, and B. Zeng, "Quantum imaginary time evolution steered by reinforcement learning", Communications Physics **5**, 57 (2022).

[162] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, "Barren plateaus in quantum neural network training landscapes", Nature Communications **9**, 4812 (2018).

[163] M. Cerezo, A. Sone, T. Volkoff, L. Cincio, and P. J. Coles, "Cost function dependent barren plateaus in shallow parametrized quantum circuits", Nature Communications **12**, 1791 (2021).

[164] Z. Holmes, A. Arrasmith, B. Yan, P. J. Coles, A. Albrecht, and A. T. Sornborger, "Barren plateaus preclude learning scramblers", Physical Review Letters **126**, 190501 (2021).

[165] S. Thanasilp, S. Wang, N. A. Nghiem, P. J. Coles, and M. Cerezo, "Subtleties in the trainability of quantum machine learning models", arXiv:2110.14753 (2021).

[166] M. Schuld and F. Petruccione, *Machine Learning with Quantum Computers*, Quantum Science and Technology (Springer International Publishing, Cham, 2021).

[167] C. Ortiz Marrero, M. Kieferová, and N. Wiebe, "Entanglement-Induced Barren Plateaus", PRX Quantum **2**, 040316 (2021).

[168] M. Cerezo, M. Larocca, D. García-Martín, N. L. Diaz, P. Braccia, E. Fontana, M. S. Rudolph, P. Bermejo, A. Ijaz, S. Thanasilp, E. R. Anschuetz, and Z. Holmes, "Does provable absence of barren plateaus imply classical simulability? Or, why we need to rethink variational quantum computing", arXiv:2312.09121 [quant-ph, stat] (2024).

[169] M. Ragone, B. N. Bakalov, F. Sauvage, A. F. Kemper, C. O. Marrero, M. Larocca, and M. Cerezo, "A Lie Algebraic Theory of Barren Plateaus for Deep Parameterized Quantum Circuits", arXiv (2023), 2309.09342 .

[170] S. Wang, E. Fontana, M. Cerezo, K. Sharma, A. Sone, *et al.*, "Noise-induced barren plateaus in variational quantum algorithms", Nature Communications **12**, 1–11 (2021).

[171] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm" (2014).

[172] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, "Evaluating analytic gradients on quantum hardware", Physical Review A **99**, 032331 (2019).

[173] G. E. Crooks, "Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition", arxiv: 1905.13311 (2019), 1905.13311 [quant-ph] .

[174] D. Wierichs, J. Izaac, C. Wang, and C. Y.-Y. Lin, "General parameter-shift rules for quantum gradients", Quantum **6**, 677 (2022).

[175] M. Motta, C. Sun, A. T. K. Tan, M. J. O. Rourke, E. Ye, A. J. Minnich, F. G. S. L. Brandao, and G. K.-L. Chan, "Determining eigenstates and thermal states on a quantum computer using quantum imaginary time evolution", Nature Physics **16**, 205–210 (2020).

[176] S. McArdle, T. Jones, S. Endo, Y. Li, S. C. Benjamin, and X. Yuan, "Variational ansatz-based quantum simulation of imaginary time evolution", npj Quantum Information **5**, 75 (2019).

[177] X. Yuan, S. Endo, Q. Zhao, Y. Li, and S. Benjamin, "Theory of variational quantum simulation", Quantum **3**, 191 (2019).

[178] C. Cao, Z. An, S.-Y. Hou, D. L. Zhou, and B. Zeng, "Quantum imaginary time evolution steered by reinforcement learning", Communications Physics **5**, 57 (2022).

[179] G. C. Wick, "Properties of Bethe-Salpeter wave functions", Physical Review **96**, 1124–1134 (1954).

[180] T. Tsuchimochi, Y. Ryo, S. L. Ten-no, and K. Sasasako, "Improved Algorithms of Quantum Imaginary Time Evolution for Ground and Excited States of Molecular Systems", Journal of Chemical Theory and Computation **19**, 503–513 (2023).

[181] W. von der Linden, "A quantum Monte Carlo approach to many-body physics", Physics Reports **220**, 53–162 (1992).

[182] D. M. Ceperley, "Path integrals in the theory of condensed helium", Reviews of Modern Physics **67**, 279–355 (1995).

[183] N. Trivedi and D. M. Ceperley, "Ground-state correlations of quantum antiferromagnets: A Green-function Monte Carlo study", Physical Review B **41**, 4552–4569 (1990).

[184] K. Guther, R. J. Anderson, N. S. Blunt, N. A. Bogdanov, D. Cleland, N. Dattani, W. Dobrautz, K. Ghanem, P. Jeszenszki, N. Liebermann, *et al.*,

"NECI: N-Electron Configuration Interaction with an emphasis on state-of-the-art stochastic methods", The Journal of Chemical Physics **153**, 034107 (2020).

[185] A. McLachlan, "A variational solution of the time-dependent Schrodinger equation", Molecular Physics **8**, 39–44 (1964).

[186] C. Zoufal, D. Sutter, and S. Woerner, "Error bounds for variational quantum time evolution", Physical Review Applied **20**, 044059 (2023).

[187] J. P. Provost and G. Vallee, "Riemannian structure on manifolds of quantum states", Communications in Mathematical Physics **76**, 289–301 (1980).

[188] D. Petz and C. Sudár, "Geometries of quantum states", Journal of Mathematical Physics **37**, 2662–2673 (1996).

[189] C.-Y. Park and M. J. Kastoryano, "Geometry of learning neural quantum states", Physical Review Research **2**, 023232 (2020).

[190] S. L. Braunstein and C. M. Caves, "Statistical distance and the geometry of quantum states", Physical Review Letters **72**, 3439–3443 (1994).

[191] P. Facchi, R. Kulkarni, V. Man'ko, G. Marmo, E. Sudarshan, and F. Ventriglia, "Classical and quantum Fisher information in the geometrical formulation of quantum mechanics", Physics Letters A **374**, 4801–4803 (2010).

[192] J. Stokes, J. Izaac, N. Killoran, and G. Carleo, "Quantum Natural Gradient", Quantum **4**, 269 (2020).

[193] S.-I. Amari, in *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96 (MIT Press, 1996) pp. 127—133.

[194] S.-i. Amari, "Natural gradient works efficiently in learning", Neural Computation **10**, 251–276 (1998).

[195] S.-i. Amari and S. Douglas, in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*, Vol. 2 (1998) pp. 1213–1216.

[196] S.-i. Amari, H. Park, and K. Fukumizu, "Adaptive method of realizing natural gradient learning for multilayer perceptrons", Neural Computation **12**, 1399–1409 (2000).

[197] J. Liu, H. Yuan, X.-M. Lu, and X. Wang, "Quantum Fisher information matrix and multiparater estimation", Journal of Physics A: Mathematical and Theoretical **53**, 023001 (2020).

[198] J. J. Meyer, "Fisher information in noisy intermediate-scale quantum applications", Quantum **5**, 539 (2021).

[199] D. Wierichs, C. Gogolin, and M. Kastoryano, "Avoiding local minima in variational quantum eigensolvers with the natural gradient optimizer", Physical Review Research **2**, 043246 (2020).

[200] J. Gacon, C. Zoufal, G. Carleo, and S. Woerner, "Simultaneous perturbation stochastic approximation of the quantum Fisher information", Quantum **5**, 567 (2021).

[201] P. Huembeli and A. Dauphin, "Characterizing the loss landscape of variational quantum circuits", Quantum Science and Technology **6**, 025011 (2021).

[202] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, "Barren plateaus in quantum neural network training landscapes", Nature Communications **9**, 4812 (2018).

[203] E. Grant, L. Wossnig, M. Ostaszewski, and M. Benedetti, "An initialization strategy for addressing barren plateaus in parametrized quantum circuits", Quantum **3**, 214 (2019).

[204] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the theory of neural computation* (Addison-Wesley Longman Publishing Co., Inc., 1991).

[205] P. Mehta and D. J. Schwab, "An exact mapping between the Variational Renormalization Group and Deep Learning", arXiv:1410.3831 (2014).

[206] H. Matsuda, "The Ising Model for Population Biology", Progress of Theoretical Physics **66**, 1078–1080 (1981).

[207] Z. Bian, F. A. Chudak, W. Macready, and G. Rose, "The Ising model : teaching an old problem new tricks", [Online; accessed 25. Sep. 2024] (2010).

[208] S. A. Cook, in *Proceedings of the Third Annual ACM Symposium on Theory of Computing - STOC '71* (ACM Press, Shaker Heights, Ohio, United States, 1971) pp. 151–158.

[209] L. A. Levin, "Universal sequential search problems", Problemy peredachi informatsii **9**, 115–116 (1973).

[210] D. J. Egger, C. Gambella, J. Marecek, S. McFaddin, M. Mevissen, R. Raymond, A. Simonetto, S. Woerner, and E. Yndurain, "Quantum Computing for Finance: State of the Art and Future Prospects", IEEE Transactions on Quantum Engineering **1**, 3101724 (2020).

[211] H. Al-Khazraji, S. Khlil, and Z. Alabacy, "Industrial Picking and Packing Problem: Logistic Management for Products Expedition", Journal of Mechanical Engineering Research and Developments **43**, 74–80 (2020).

[212] R. Andonov, V. Poirriez, and S. Rajopadhye, "Unbounded knapsack problem: Dynamic programming revisited", European Journal of Operational Research **123**, 394–407 (2000).

[213] A. Lucas, "Ising formulations of many NP problems", Frontiers in Physics **2**, 5 (2014).

[214] D. Fitzek, *Applying quantum approximate optimization to the heterogeneous vehicle routing problem*, Tech. Rep. (2022).

[215] G. B. Dantzig and J. H. Ramser, "The Truck Dispatching Problem", Management Science **6**, 80–91 (1959).

[216] B. Golden, A. Assad, L. Levy, and F. Gheysens, "The fleet size and mix vehicle routing problem", Computers & Operations Research **11**, 49–66 (1984).

[217] Ç. Koç, T. Bektaş, O. Jabali, and G. Laporte, "Thirty years of heterogeneous vehicle routing", European Journal of Operational Research **249**, 1–21 (2016).

[218] T. Ghandriz, B. Jacobson, M. Islam, J. Hellgren, and L. Laine, "Transportation-Mission-Based Optimization of Heterogeneous Heavy-Vehicle Fleet Including Electrified Propulsion", Energies **14**, 3221 (2021).

[219] S. Feld, C. Roch, T. Gabor, C. Seidel, F. Neukart, I. Galter, W. Mauerer, and C. Linnhoff-Popien, "A hybrid solution method for the capacitated vehicle routing problem using a quantum annealer", Frontiers in ICT **6** (2019).

[220] R. Harikrishnakumar, S. Nannapaneni, N. H. Nguyen, J. E. Steck, and E. C. Behrman, "A Quantum Annealing Approach for Dynamic Multi-Depot Capacitated Vehicle Routing Problem", arXiv:2005.12478 (2020).

[221] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 135, edited by W. van Dam and L. Mančinska (Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2019) pp. 5:1–5:32.

[222] 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**, 021318 (2019).

[223] R. Iten, R. Moyard, T. Metger, D. Sutter, and S. Woerner, "Exact and Practical Pattern Matching for Quantum Circuit Optimization", ACM Transactions on Quantum Computing **3**, 1–41 (2022).

[224] Md. M. Rahman, G. W. Dueck, and J. D. Horton, "An Algorithm for Quantum Template Matching", ACM Journal on Emerging Technologies in Computing Systems (JETC) **11**, 1–20 (2014).

[225] M. G. Pozzi, S. J. Herbert, A. Sengupta, and R. D. Mullins, "Using Reinforcement Learning to Perform Qubit Routing in Quantum Compilers", ACM Transactions on Quantum Computing **3**, 1–25 (2022).

[226] A. Paler, A. Zulehner, and R. Wille, "Nisq circuit compilation is the travelling salesman problem on a torus", Quantum Science and Technology **6**, 025016 (2021).

[227] M. Y. Siraichi, V. F. dos Santos, S. Collange, and F. M. Q. Pereira, in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, Vol. 2018-Febru (ACM, New York, NY, USA, 2018) pp. 113–125.

[228] P. Murali, J. M. Baker, A. J. Abhari, F. T. Chong, and M. Martonosi, "Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers", Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems , 1015–1029 (2019), arXiv:1901.11054 .

[229] A. Paler, in *Quantum Technology and Optimization Problems* (Springer, Cham, Switzerland, 2019) pp. 207–217.

[230] K. Booth, M. do, J. Beck, E. Rieffel, D. Venturelli, and J. Frank, "Comparing and integrating constraint programming and temporal planning for quantum circuit compilation", Proceedings of the International Conference on Automated Planning and Scheduling **28**, 366-374 (2018).

[231] G. Li, Y. Ding, and Y. Xie, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19 (Association for Computing Machinery, New York, NY, USA, 2019) p. 1001–1014.

[232] S. Martiel and T. G. d. Brugière, "Architecture aware compilation of quantum circuits via lazy synthesis", Quantum **6**, 729 (2022).

[233] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the ibm qx architectures", arXiv:1712.04722 [quant-ph] (2018).

[234] A. Zulehner and R. Wille, in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ASPDAC '19 (Association for Computing Machinery, New York, NY, USA, 2019) p. 185–190.

[235] B. Nash, V. Gheorghiu, and M. Mosca, "Quantum circuit optimizations for NISQ architectures", Quantum Science and Technology **5**, 025010 (2020).

[236] A. Kissinger and A. M.-v. de Griend, "CNOT circuit extraction for topologically-constrained quantum memories", arXiv (2019), 1904.00633 .

[237] M. Y. Siraichi, V. F. dos Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation as a combination of subgraph isomorphism and token swapping", Proceedings of the ACM on Programming Languages **3**, 1–29 (2019).

[238] X. Zhou, S. Li, and Y. Feng, "Quantum Circuit Transformation Based on Simulated Annealing and Heuristic Search", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **39**, 4683–4694 (2020), arXiv:1908.08853 .

[239] X. Zhou, Y. Feng, and S. Li, "Supervised learning enhanced quantum circuit transformation", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **42**, 437-447 (2023).

[240] X. Zhou, Y. Feng, and S. Li, in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)* (2020) pp. 1–7.

[241] S. Herbert and A. Sengupta, "Using Reinforcement Learning to find Efficient Qubit Routing Policies for Deployment in Near-term Quantum Computers", arXiv (2018), arXiv:1812.11619 .

[242] M. Do, Z. Wang, B. O'Gorman, D. Venturelli, E. Rieffel, and J. Frank, "Planning for Compilation of a Quantum Algorithm for Graph Coloring", Frontiers in Artificial Intelligence and Applications **325**, 2338–2345 (2020), arXiv:2002.10917 .

[243] D. Venturelli, M. Do, E. Rieffel, and J. Frank, "Compiling quantum circuits to realistic hardware architectures using temporal planners", Quantum Science and Technology **3**, 025004 (2017), arXiv:1705.08927 .

[244] D. Venturelli, M. Do, E. Rieffel, and J. Frank, in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence* (International Joint Conferences on Artificial Intelligence Organization, California, 2017) pp. 4440–4446.

[245] T. K. Moon, *Error Correction Coding* (2005).

[246] A. Shokrollahi, in *Coding, Cryptography and Combinatorics*, edited by K. Feng, H. Niederreiter, and C. Xing (Birkhäuser Basel, Basel, 2004) pp. 85–110.

[247] A. F. Kockum, A. Soro, L. García-Álvarez, P. Vikstål, T. Douce, G. Johansson, and G. Ferrini, "Lecture notes on quantum computing", arXiv:2311.08445 [quant-ph] (2024).

[248] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned", Nature **299**, 802–803 (1982).

[249] A.Yu. Kitaev, "Fault-tolerant quantum computation by anyons", Annals of Physics **303**, 2–30 (2003).

[250] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory", Journal of Mathematical Physics **43**, 4452–4505 (2002).

[251] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation", Phys. Rev. A **86**, 032324 (2012).

[252] B. M. Terhal, "Quantum Error Correction for Quantum Memories", Reviews of Modern Physics **87**, 307–346 (2015), arXiv:1302.3428 [quant-ph] .

[253] D. Gottesman, *Stabilizer Codes and Quantum Error Correction*, Tech. Rep. (1997) arXiv:quant-ph/9705052 .

[254] R. Acharya, L. Aghababaie-Beni, I. Aleiner, T. I. Andersen, M. Ansmann, *et al.*, "Quantum error correction below the surface code threshold", arXiv (2024), 2408.13687 .

[255] J. Edmonds, "Paths, Trees, and Flowers", Canadian Journal of Mathematics **17**, 449–467 (1965).

[256] A. G. Fowler, "Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $O(1)$ parallel time", arXiv 10.48550/arXiv.1307.1740 (2013), 1307.1740 .

[257] J. Bausch, A. W. Senior, F. J. H. Heras, T. Edlich, A. Davies, M. Newman, C. Jones, K. Satzinger, M. Y. Niu, S. Blackwell, G. Holland, D. Kafri, J. Atalaya, C. Gidney, D. Hassabis, S. Boixo, H. Neven, and P. Kohli, "Learning to Decode the Surface Code with a Recurrent, Transformer-Based Neural Network", arXiv (2023), 2310.05900 .

[258] M. Lange, P. Havström, B. Srivastava, V. Bergentall, K. Hammar, *et al.*, "Data-driven decoding of quantum error correcting codes using graph neural networks", arXiv (2023), 2307.01241 .

[259] X. Wu, X. Liang, Y. Tian, F. Yang, C. Chen, Y.-C. Liu, M. K. Tey, and L. You, "A concise review of Rydberg atom based quantum computation and quantum simulation", Chinese Physics B **30**, 020305 (2021).

[260] M. D. Lukin, M. Fleischhauer, R. Cote, L. M. Duan, D. Jaksch, J. I. Cirac, and P. Zoller, "Dipole Blockade and Quantum Information Processing in Mesoscopic Atomic Ensembles", Physical Review Letters **87**, 037901 (2001).

[261] A. Browaeys and T. Lahaye, "Many-body physics with individually controlled Rydberg atoms", Nature Physics **16**, 132–142 (2020).

[262] D. Jaksch, J. I. Cirac, P. Zoller, S. L. Rolston, R. Côté, and M. D. Lukin, "Fast Quantum Gates for Neutral Atoms", Physical Review Letters **85**, 2208–2211 (2000).

[263] M. Endres, H. Bernien, A. Keesling, H. Levine, E. R. Anschuetz, A. Krajenbrink, C. Senko, V. Vuletic, M. Greiner, and M. D. Lukin, "Atom-by-atom assembly of defect-free one-dimensional cold atom arrays", Science **354**, 1024–1027 (2016).

[264] R. Samajdar, W. W. Ho, H. Pichler, M. D. Lukin, and S. Sachdev, "Complex density wave orders and quantum phase transitions in a model of square-lattice rydberg atom arrays", Physical Review Letters **124**, 103601 (2020).

[265] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian, "New benchmark instances for the Capacitated Vehicle Routing Problem", European Journal of Operational Research **257**, 845–858 (2017).

[266] S. E. Barman, P. Lindroth, and A.-B. Strömberg, in *Optimization, Control, and Applications in the Information Age*, edited by A. Migdalas and A. Karakitsiou (Springer International Publishing, Cham, 2015) pp. 113–138.