

INGENIOS

Benchmarking Quantum Supremacy for Combinatorial Optimization & Machine Learning Problems

Deliverable 1.1

A Comparative Analysis of State-of-the-Art Classical
Combinatorial Optimization, Machine Learning Methods, and
Quantum Computing

Spindox S.p.A.

Contents

1	Introduction	4
2	Methodology	6
2.1	End-to-End Workflow Analysis	6
2.1.1	Problem Formulation	7
2.1.2	Mathematical Reformulation	7
2.1.3	Quantum Circuit Creation	8
2.1.4	Quantum Computing Simulation on a Classical Platform	9
2.1.5	Quantum Hardware Resource Choice	10
2.1.6	Transpilation	10
2.1.7	Job Execution	11
2.1.8	Comparison with Classical	11
2.1.9	Post-Processing and Visualization	13
2.2	Methodology for Evaluating Algorithm Efficiency	13
2.3	Evaluation metrics	14
3	State of the Art: Combinatorial Optimization Algorithms	15
3.1	Basic Facts & Notation	15
3.2	Optimizing under Uncertain Conditions	16
3.3	Real-World Problems	17
3.4	Combinatorial Problems and $\mathcal{NP}$ -Hardness	18
3.5	Classical Solving Algorithms	19
3.5.1	Exact Algorithms	19
3.5.2	Metaheuristics	20
3.5.3	Heuristics	22
3.6	Pros and Cons of Classical Solving Algorithms	22
3.7	Practical Applications: VRP	23
3.7.1	Literature	23
3.7.2	Spindox Approach: A Metaheuristic-based VRP Solver	24
3.8	Practical Applications: Scheduling Problems	25
3.8.1	Literature	25

3.8.2	Spindox Approach: A Multi-Resource Flexible Job-Shop Scheduling problem with Non-Linear Routing and Sequence-Dependent Setup Times	27
4	State of the Art: Machine Learning Algorithms	29
4.1	Supervised Learning	29
4.1.1	Regression Methods	30
4.1.2	Neural Network (NN)	33
4.1.3	Support Vector Machines (SVMs)	37
4.1.4	Other Supervised Learning Models	43
4.2	Unsupervised Learning	44
4.2.1	K-means Clustering	45
4.2.2	Principal Component Analysis (PCA)	46
4.3	Limitations of ML Algorithms	47
5	State of the Art: Quantum Algorithms	49
5.1	Fundamentals of Quantum Information	49
5.1.1	Core Concepts of Quantum Mechanics	50
5.1.2	Core Concepts of Quantum Computing	53
5.2	Quantum Optimization	60
5.2.1	Mathematical Reformulation: Objective Function . . .	60
5.2.2	Mathematical Reformulation: Constraints	61
5.2.3	Hamiltonian Representation	63
5.3	Quantum Annealing	64
5.4	Quantum Approximate Optimization Algorithm (QAOA) . . .	66
5.4.1	Key Algorithmic Steps	67
5.5	Quantum Machine Learning	70
5.5.1	Quantum Encoding	72
5.5.2	HHL Algorithm	75
5.5.3	Quantum Support Vector Machine (QSVM)	76
5.5.4	Variational Quantum Circuit (VQC)	78
5.5.5	Quantum Neural Network (QNN)	82
5.6	Hardware Limitations	83
5.6.1	Introduction to IBM Quantum Hardware	84
5.6.2	Requirements of Quantum Algorithms	85
5.7	Mitigation Strategies for Hardware Limitations	90
5.7.1	Quantum Error Correction	90
5.7.2	Strategies for Reducing Qubit Requirements	91
5.7.3	Strategies for Minimizing Circuit Depth	91
5.7.4	Noise Reduction Strategies	92
5.7.5	Noise-Resilient Algorithms	93

5.7.6	Post-Processing Error Mitigation	93
6	Comparison between Classical and Quantum methods	95
6.1	Advantages of Quantum Approaches in Optimization	95
6.1.1	Quantum Supremacy	95
6.1.2	Quantum Advantage	96
6.2	Advantages of Quantum Approaches in Machine Learning . . .	97
6.3	Disadvantages of Quantum Approaches	99
6.3.1	Hardware Limitations	99
6.3.2	Non-Hardware Limitations	100
6.3.3	Disadvantages of Quantum Approaches in Optimization	101
6.3.4	Disadvantages of Quantum Approaches in Machine Learning	103
7	Conclusions and Recommendations	106
7.1	Key Findings	106
7.2	Strategic Roadmap for Future Phases	107
7.3	Conclusion	109

Chapter 1

Introduction

The INGENIOS project, titled “Benchmarking Quantum Supremacy for Combinatorial Optimization and Machine Learning Problems”, is driven by the growing need to explore the practical capabilities and limitations of quantum computing in real-world industrial contexts. Quantum computing has the potential to transform how we approach optimization and machine learning tasks, offering a new computational paradigm that could surpass classical methods, particularly in problems where the complexity grows exponentially with the size of the instance.

Optimization problems such as Vehicle Routing, Production Planning, and Fleet Management are at the core of many industrial decision-making processes. These problems are often modelled as combinatorial optimization challenges, which, due to their inherent complexity, are difficult to solve using exact methods within a reasonable time frame. Machine learning, another key domain for INGENIOS, faces similar challenges, particularly in clustering, classification, and regression tasks where data complexity and volume can overwhelm traditional algorithms.

This deliverable represents the first step in the INGENIOS project’s roadmap. It provides a comprehensive comparative analysis of classical and quantum algorithms, examining their performance, strengths, and limitations when applied to combinatorial optimization and machine learning problems. The analysis aims to address key questions such as:

- What are the current state-of-the-art classical algorithms for optimization and machine learning, and what are their performance bottlenecks?
- How do quantum algorithms, including hybrid approaches, compare in terms of efficiency, scalability, and accuracy?

- What specific problem characteristics lend themselves to quantum solutions, and how do hardware limitations impact their applicability?

By conducting this comparative study, the INGENIOS project seeks to lay a solid foundation for the future phases of research, which will involve the practical application and benchmarking of quantum algorithms on industrial problems. Furthermore, this deliverable will offer guidance on selecting the most promising quantum methods and frameworks, assisting collaborators in aligning their efforts toward solving real-world use cases effectively.

The structure of this document is designed to provide a clear, methodical overview of the current landscape in classical and quantum computing. It will begin by outlining the methodologies used for analysis, followed by in-depth explorations of the state-of-the-art in combinatorial optimization, machine learning, and quantum algorithms. The document will then present a comparison between classical and quantum approaches, concluding with recommendations for future research directions and development within the project.

Ultimately, this deliverable serves as a critical milestone in the INGENIOS project, bridging theoretical advancements with practical applications. It not only evaluates where quantum computing stands today but also points to where it could lead us in the coming years, potentially revolutionizing how industries approach computational challenges in optimization and machine learning.

Chapter 2

Methodology

The goal of the INGENIOS project is to rigorously assess the feasibility of quantum computing in “real” conditions, exploring its potential to solve practical and complex problems relevant to our business needs and those of our clients. This chapter outlines the methodological approach taken in this project, focusing on the full workflow, from problem formulation to execution on quantum hardware. This analysis aims to understand if and how quantum computing can offer meaningful computational advantages in real-world applications, such as the *Vehicle Routing Problem* (VRP), where constraints and variability mirror actual business scenarios.

Our approach includes analyzing quantum algorithms not just in theoretical terms but through direct benchmarking on real quantum computers, addressing performance limitations, cost implications, and process complexities.

To achieve the objectives of the INGENIOS project, we will leverage IBM®’s state-of-the-art quantum computing infrastructure, including its quantum machines and the *Qiskit* framework. IBM®’s quantum systems provide the necessary hardware capabilities to implement and test our algorithms in real-world conditions, while Qiskit offers a robust suite of tools for developing, simulating, and optimizing quantum circuits. This combination allows us to translate theoretical models into executable workflows, facilitating a detailed evaluation of quantum computing’s feasibility and performance in addressing practical problems.

2.1 End-to-End Workflow Analysis

Our methodology includes a complete analysis of the workflow required to translate a classical problem into a quantum solution. This end-to-end work-

flow encompasses each stage in the computational pipeline.

2.1.1 Problem Formulation

The first step involves identifying and clearly defining the problem we intend to solve. In this phase, we choose a real-world problem and translate it into mathematical terms, creating an abstract version that can be worked with on a quantum computer. This stage sets the foundation for all the following steps by ensuring that we have a well-understood problem that fits into the combinatorial optimization or machine learning category.

In particular, our objective is to model a VRP, which is highly suitable for quantum implementation due to its nature. We will examine the problem’s dual complexity from two perspectives: constraint modeling and scalability. First, we will analyze how effectively we can model and include multiple constraints, such as time windows and pickup-and-delivery requirements. Second, we will explore the dimensional complexity, specifically the number of nodes (representing locations or customers) and the number of vehicles.

In addition to the VRP, classification problems in machine learning offer an excellent framework for exploring quantum computing’s potential. Classification will be investigated by modeling the problem in terms of the hybrid Classical-Quantum machine learning algorithms available, specifically *Quantum Support Vector Machines* (QSVM), *Quantum k-Nearest Neighbors* (QkNN), and *Quantum Neural Networks* (QNN). The objective is to examine accuracy and scalability to large datasets, particularly by testing the reliability of *Quantum machine learning* (QML) results on complex problems with large data sizes and high-dimensional feature spaces.

Both problems will be evaluated with respect to the available quantum hardware, focusing on key metrics such as the number of qubits and circuit depth required. By addressing these facets, we aim to determine the feasibility and efficiency of implementing a quantum solution for VRP and classification, highlighting potential advantages and limitations posed by current quantum processing capabilities.

2.1.2 Mathematical Reformulation

For what concerns optimization problems, once the problem is defined, we reformulate it into a *Quadratic Unconstrained Binary Optimization* (QUBO) problem. QUBO is a popular form that is well-suited for quantum optimization since it represents the problem in a binary, quadratic framework that quantum devices can efficiently work with. Reformulating in this way

transforms the original problem into a form that allows quantum systems to handle binary variables, a fundamental step for preparing combinatorial optimization problems for quantum algorithms. To solve our problem on a quantum computer, we then create a Hamiltonian, which is a mathematical expression that represents the energy landscape of the problem. The minimum energy objective guides the quantum computer in finding the lowest possible energy state of this Hamiltonian, which corresponds to the best solution for our optimization problem.

For QML, the reformulation step involves encoding classical data, in terms of features, into quantum states. Various encoding methods can be employed, such as *amplitude encoding*, where the data is encoded into the amplitudes of quantum states, or *angle encoding*, where the data is represented in the angles of quantum gates (typically rotation gates) acting on qubits. This enables the quantum system to process data in a high-dimensional Hilbert space, which is a key advantage in solving complex problems. As with optimization problems, the encoding process must be carefully considered in terms of circuit depth and qubit count. Encoding methods that require fewer qubits or shallower circuits may offer advantages in terms of computational efficiency, while more complex encodings may be more powerful but computationally intensive.

2.1.3 Quantum Circuit Creation

We next move to designing a quantum circuit. This circuit is a sequence of quantum gates arranged to solve the problem, with each gate representing an operation on the quantum bits (*qubits*). The structure of the circuit defines how the quantum computer will process and manipulate information to arrive at a solution.

The design process begins with identifying the appropriate algorithms that inform the circuit’s architecture. For instance, in optimization problems, algorithms like the *Quantum Approximate Optimization Algorithm* (QAOA) provide a framework for constructing the circuit, defining its layers and parameterization. Similarly, in machine learning applications, approaches such as the *Quantum Support Vector Machine* (QSVM) or *Variational Quantum Classifiers* (VQC) guide the selection and arrangement of quantum gates to represent data and perform computations effectively. These algorithms dictate the circuit’s logical flow, aligning it with the problem’s specific requirements.

These kind of circuits are called variational quantum circuits, and designing them is a crucial step in most of optimization and machine learning quantum application. A key consideration is the number of *layers* (or *p* lay-

ers), which corresponds to the depth of the circuit. While adding more layers can potentially increase the accuracy of the solution, it also raises the risk of introducing noise. In addition, increasing the layers makes the problem more computationally intensive, leading to longer execution times. In some cases, adding too many layers can render the problem practically intractable for current quantum hardware. Therefore, there is a trade-off between the depth of the circuit and both the accuracy and feasibility of executing the circuit reliably on a quantum computer.

Furthermore, the number of qubits available is a limiting factor, as many problems require a large qubit count to represent complex computations effectively. Techniques such as *circuit cutting* allow for decomposing large circuits into smaller, more manageable parts, enabling the execution of circuits that otherwise would exceed qubit limitations. Circuit cutting also helps in addressing noise by isolating parts of the computation and reducing the error propagation across the entire circuit.

2.1.4 Quantum Computing Simulation on a Classical Platform

Classical simulations play a pivotal role in the quantum computing workflow, acting as a critical bridge between theoretical algorithm design and practical hardware implementation. They enable:

- **Algorithm Validation:** By simulating quantum circuits classically, researchers can verify the correctness of quantum algorithms and detect errors in the design phase, ensuring that they function as intended before deployment on actual hardware.
- **Performance Benchmarking:** Simulations provide valuable insights into the expected behavior of quantum circuits, such as gate fidelity, circuit depth, and qubit utilization, helping to predict potential bottlenecks or failures on quantum machines.
- **Feasibility Testing:** Classical simulations allow testing of quantum circuits under realistic constraints, such as limited qubit connectivity or noise, ensuring that the circuits are tailored to the specific requirements of the target hardware.

Before running the quantum circuit on an actual quantum device, we simulate it on a classical platform. These simulators allow researchers to execute quantum circuits as they would run on actual quantum devices, providing both idealized (noiseless) and realistic (noisy) simulation modes. By

incorporating noise models that mimic the imperfections found in physical qubits, such as decoherence and gate errors, the simulator enables a more accurate assessment of algorithm performance under conditions similar to those on real hardware. One powerful feature in this context is the simulator’s ability to handle *stabilizer circuits*, a specific class of quantum circuits that can be efficiently simulated due to their structure. Stabilizer circuits, based on operations that map Pauli operators to other Pauli operators, are essential for tasks like error correction and certain quantum protocols, where the preservation of certain quantum states is critical. Leveraging this efficient simulation capability allows for the rapid testing and verification of circuits central to practical quantum applications. Additionally, it offers the option to accelerate computations by leveraging GPU resources and parallel computing, which significantly enhances simulation speed, particularly for complex circuits and large-scale systems. This capability is invaluable for prototyping and optimizing quantum algorithms before deploying them on actual quantum processors. Moreover, the simulation helps in assessing how well the circuit might perform and identifying any adjustments that need to be made to ensure accuracy. Classical simulations also allow us to benchmark the performance of the circuit in a controlled environment.

However, classical simulations are inherently limited by the exponential growth in computational resources required to model large quantum systems. This highlights the need for careful prioritization of simulation tasks and a focus on circuits that are scalable within available computational resources.

2.1.5 Quantum Hardware Resource Choice

After verifying the circuit on a classical simulator, we choose the appropriate quantum hardware. Selecting the right quantum device depends on factors like qubit quality, coherence time, and the size of the circuit required for the problem. This step ensures that we are using the most suitable quantum resources available for the task.

2.1.6 Transpilation

Transpilation is the process of adapting our circuit to run effectively on the chosen quantum hardware. This step includes several sub-processes:

- **Layout:** Mapping logical qubits to physical qubits on the hardware.
- **Routing:** Managing qubit interactions to ensure efficient operation within hardware constraints.

- **Translation:** Converting high-level instructions into specific gate operations that the device can perform.
- **Optimization:** Reducing the number of gates and complexity to improve the circuit’s efficiency and performance on the hardware.

Key metrics for assessing the transpilation process include:

- **Transpilation Time:** The time required to complete circuit optimization.
- **Transpilation Gains:** Improvements in circuit feasibility (lower error rates, reduced gate count) and performance due to transpilation.

The Qiskit framework by IBM[®] provides a comprehensive suite of tools for quantum circuit transpilation, particularly focused on optimizing circuit performance for specific hardware constraints. This includes advanced optimization techniques aimed at minimizing circuit depth and reducing gate count, which are essential for improving execution fidelity on noisy quantum processors.

One of the standout features of Qiskit’s transpilation toolkit is its flexibility in customization: users can define tailored optimization passes that adapt the circuit to the unique needs of their application. By creating custom transpilation steps, developers can apply highly targeted optimizations that go beyond generic methods, which often lack the specificity needed for complex, application-driven requirements. This customized approach can lead to significant performance gains, making circuits not only more efficient but also more resilient to the limitations of current quantum hardware.

2.1.7 Job Execution

With the optimized circuit ready, we proceed to the job execution phase, where the circuit runs on the quantum hardware. This step involves scheduling, where we determine the best time to execute the job to reduce potential errors and make the most of the hardware’s availability. When possible, we also use parallelization techniques, running multiple instances of the circuit to gather more data and improve the reliability of the results.

2.1.8 Comparison with Classical

In classical computing, performance is often measured in terms of *Floating-point Operations Per Second* (FLOPS), indicating the number of floating-point arithmetic operations a system can perform per second. FLOPS is

a standard metric for evaluating the computational power of classical algorithms, especially in applications that require intensive numerical calculations, such as simulations, machine learning, and scientific computing.

In quantum computing, an analogous metric called *Circuit Layer Operations Per Second* (CLOPS) is used to measure performance. CLOPS quantifies the rate at which quantum operations, or layers of quantum gates, can be executed on a quantum processor. While CLOPS and FLOPS both measure computational speed, they apply to fundamentally different types of computation. CLOPS focuses on the rate of circuit layers executed on qubits, which exploit quantum phenomena such as superposition and entanglement, offering the potential for exponential speedup in specific problem domains.

Another critical metric in quantum computing is the *Quantum Volume* (QV), which provides a holistic measure of a quantum computer’s capability. Quantum Volume considers several factors, including the number of qubits, the connectivity between them, gate fidelity, and circuit depth. It reflects the system’s ability to solve increasingly complex quantum problems, making it a key metric for evaluating the practical utility of quantum hardware. Unlike CLOPS, which measures computational throughput, Quantum Volume assesses the quality and scalability of quantum computations, indicating the types of problems a system can handle effectively. A high QV suggests that a quantum processor can execute more sophisticated algorithms with greater accuracy, which is essential for real-world applications.

Although CLOPS and FLOPS both indicate computational throughput, they are not directly comparable. FLOPS measures classical arithmetic operations, while CLOPS reflects quantum gate operations on a circuit layer. Similarly, Quantum Volume has no direct classical equivalent, as it encapsulates unique attributes of quantum systems. Therefore, evaluating the efficiency of classical versus quantum algorithms requires not just a comparison of FLOPS and CLOPS but also consideration of Quantum Volume, as well as the distinct computational paradigms and types of problems each is suited to address.

Unlike classical algorithms, which are typically deterministic and produce the same output for a given input, **quantum algorithms are inherently non-deterministic**. Quantum computations leverage principles such as superposition and entanglement, which lead to a range of possible outcomes upon measurement. As a result, running the same quantum algorithm multiple times can yield different results, even when using identical inputs. To address this, our analysis will also consider the concept of **repeatability**. This involves evaluating how consistently the algorithm produces correct or optimal results across multiple executions. We will investigate the statistical properties of repeated runs, examining both the distribution of outcomes and

the likelihood of obtaining the desired solution. The analysis will help us understand the trade-off between execution fidelity and the probabilistic nature of quantum computing, providing insights into the feasibility and reliability of quantum algorithms for practical applications.

2.1.9 Post-Processing and Visualization

Finally, after executing the quantum circuit, we analyze the output. Post-processing includes interpreting the results, filtering out errors, and potentially refining the data. Visualization techniques help us to better understand the outcomes, especially for non-expert audiences, as graphical representations often provide clearer insights into the performance and efficacy of the quantum computation.

2.2 Methodology for Evaluating Algorithm Efficiency

The efficiency of an algorithm is often evaluated using **asymptotic notation**, which provides a way to classify algorithms based on their growth rate relative to the input size n . Common notations include:

- **Big-O** ($\mathcal{O}$) for the upper bound,
- **Big-Theta** (Θ) for a tight bound,
- **Big-Omega** (Ω) for the lower bound.

These notations (see Cormen et al. [36]) describe different bounds on an algorithm's runtime and are formally defined as follows.

Big-O Notation The Big-O notation $f(n) = \mathcal{O}(g(n))$ describes an *upper bound* on $f(n)$:

$$f(n) = \mathcal{O}(g(n)) \Leftrightarrow \exists c > 0, n_0 \in \mathbb{N} : f(n) \leq c \cdot g(n), \forall n \geq n_0.$$

Big-Theta Notation The Big-Theta notation $f(n) = \Theta(g(n))$ represents a *tight bound*:

$$f(n) = \Theta(g(n))$$

if and only if:

$$\exists c_1, c_2 > 0, n_0 \in \mathbb{N} : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n), \forall n \geq n_0.$$

Big-Omega Notation The Big-Omega notation $f(n) = \Omega(g(n))$ defines a *lower bound*:

$$f(n) = \Omega(g(n)) \Leftrightarrow \exists c > 0, n_0 \in \mathbb{N} : f(n) \geq c \cdot g(n), \forall n \geq n_0.$$

These notations are essential in understanding the scalability and performance of algorithms as input sizes increase, serving as a foundation for algorithm analysis in computational theory.

2.3 Evaluation metrics

To comprehensively assess the performance of quantum algorithms and hardware, and to evaluate the real-world practicality of quantum computing we employ the following metrics:

- **Classical Computation Time for Simulation:** The time required for classical simulation of quantum circuits, which allows us to estimate resource demands before deploying on quantum hardware.
- **Transpilation and Optimization Time:** The time taken to convert high-level circuits into hardware-compatible formats. Transpilation impacts both circuit depth and error rates, making it a key step in the workflow.
- **Circuit Feasibility Gains:** These gains quantify the improvements achieved through optimization, such as reduced circuit depth, minimized gate count, and improved connectivity handling.
- **Quantum Computation Time on Hardware:** The actual execution time on quantum processors, essential for determining the practicality of quantum computation in real-time applications.
- **Monetary Costs:** the financial cost associated with each computational step, including simulation, transpilation, and quantum execution. This data provides an economic assessment of quantum computing's current and potential value.

These metrics are integrated into the workflow to ensure a holistic analysis of quantum performance, from algorithm design to execution and post-processing. By tracking these metrics, we can identify bottlenecks, quantify improvements, and build a roadmap for achieving quantum advantage.

Chapter 3

State of the Art: Combinatorial Optimization Algorithms

3.1 Basic Facts & Notation

Combinatorial optimization is a branch of optimization that focuses on finding an optimal object from a finite set of objects (Korte and Vygen [101]). Unlike continuous optimization, where solutions can vary smoothly, combinatorial optimization deals with discrete structures. A general combinatorial optimization problem can be formulated as:

$$\min_{x \in \mathcal{X}} f(x) \tag{3.1}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$ is a finite or countable set of feasible solutions (which could represent, for example, all possible combinations or permutations of elements) and $f : \mathcal{X} \rightarrow \mathbb{R}$ is an objective function that assigns a cost to each solution $x \in \mathbb{R}^n$. The goal is to find the *optimal solution* $x^* \in \mathcal{X}$ such that:

$$f(x^*) \leq f(x) \quad \forall x \in \mathcal{X}.$$

Definition 1 A solution $\bar{x} \in \mathcal{X}$ is said to be *locally optimal* if $\varepsilon > 0$ exists such that:

$$f(\bar{x}) \leq f(x) \quad \forall x \in \mathcal{X} : \|x - \bar{x}\| \leq \varepsilon.$$

Definition 2 A problem for which there is no feasible solution ($\mathcal{X} = \emptyset$) is said to be *infeasible*; in this case we agree to write:

$$\min_{x \in \mathcal{X}} f(x) = +\infty.$$

Definition 3 A problem for which f is not bounded from below in $\mathcal{X}$ is said to be unbounded; in this case we write:

$$\min_{x \in \mathcal{X}} f(x) = -\infty,$$

meaning that $M > 0$, $\exists x \in \mathcal{X}$ such that $f(x) < -M$.

The reader is referred to Fischetti [53] and references therein for any further detail and a general overview on classes of mathematical optimization models.

3.2 Optimizing under Uncertain Conditions

An underlying assumption of classical combinatorial models is that the input data are not corrupted with noise and, therefore, known exactly at the moment of optimizing. This assumption, however, is not always practical. Indeed, real-world observations are often plagued by uncertainty (*e.g.*, due to limited precision of collecting instruments, measurement mistakes in data gathering, sampling errors, etc.) and disregarding it might lead to solutions that are far from optimal. Therefore, the problem of formulating models not facing deterioration when there are some perturbations in the data set is an interesting problem that has gained considerable attention from the scientific community.

Below are reported the main paradigms to deal with problems affected with uncertain data.

Definition 4 The risk neutral stochastic version of (3.1) is given as follows (Shapiro et al. [146]):

$$\min_{x \in \mathcal{X}} \mathbb{E}_{\Pi}[f(x, \xi)], \quad (3.2)$$

where the random vector ξ is defined on a probability space $(\Xi, \mathfrak{F}, \Pi)$ with support $\Xi \subseteq \mathbb{R}^d$ and probability distribution Π on the σ -algebra $\mathfrak{F}$; f is a real-valued function on $\mathcal{X} \times \Xi$ and the expectation $\mathbb{E}_{\Pi}$ is taken with respect to the probability distribution of ξ .

Definition 5 The risk averse stochastic version of (3.1) is given as follows (Artzner et al. [13]):

$$\min_{x \in \mathcal{X}} \mathcal{R}_{\Pi}[f(x, \xi)], \quad (3.3)$$

with $\mathcal{R}_{\Pi}$ coherent measure of risk on ξ . By virtue of example, a common choice of $\mathcal{R}_{\Pi}$ is the mean-CVaR.

Definition 6 *The robust version of (3.1) is given as follows (Ben-Tal et al. [17]):*

$$\min_{x \in \mathcal{X}} \max_{\xi \in \Xi} f(x, \xi), \quad (3.4)$$

where the goal is to minimize the worst-case cost $f(x, \xi)$ over all potential realizations of the uncertain parameters ξ within a predefined uncertainty set Ξ .

Definition 7 *The distributionally robust version of (3.1) is given as follows (Gao et al. [57]):*

$$\min_{x \in \mathcal{X}} \max_{\Pi \in \mathcal{P}} \mathbb{E}_{\Pi}[f(x, \xi)], \quad (3.5)$$

where the expectation is taken over a set of probability distributions¹ $\mathcal{P}$ that capture the ambiguity in the true distribution of the uncertain parameters ξ . The two most widely used types of ambiguity sets are moment-based and statistical distance-based sets, with popular choices to measure the dissimilarity between two probability distributions being Wasserstein distance and φ -divergences (such as Kullback-Leibler divergence, Burg entropy, Modified- χ^2 distance, Variation distance, etc.). See Bayraksan and Love [15], Xie and Wei [168].

3.3 Real-World Problems

Combinatorial optimization plays a crucial role in solving a wide range of real-world problems. A few notable applications include (notice that the list is not exhaustive):

- *Knapsack Problem:* In this problem, one must select items with given weights and values to maximize the total value without exceeding a weight limit. It is used in resource allocation, finance, and project selection. See Salkin and De Kluyver [141].
- *Assignment Problems:* These involve assigning a set of agents (workers, resources) to tasks, where each assignment has a cost. The goal is to minimize the total cost of the assignments. This applies to areas like staff scheduling, airline crew assignments, and school timetabling. See Kuhn [105].
- *Traveling Salesman Problem (TSP):* Given a set of cities and distances between them, find the shortest route that visits each city once and returns to the starting depot.

¹The so-called ambiguity set.

- *Vehicle Routing Problem (VRP)*: This generalizes the TSP by involving multiple vehicles. The goal is to find the optimal set of routes for a fleet of vehicles to service a set of customers while minimizing travel costs. VRP has broad applications in logistics, supply chain management, and last-mile delivery systems. See Kim et al. [99].
- *Job Scheduling*: It involves assigning tasks to machines or processors to optimize objectives such as minimizing completion time or balancing workload. Applications can be found in production planning and cloud computing.
- *Nurse Rostering*: The objective is to create an optimal schedule for nurses while satisfying a variety of constraints, such as ensuring adequate coverage for each shift, adhering to labor regulations (*e.g.*, maximum working hours, mandatory rest periods), accommodating nurses' preferences (*e.g.*, days off, shift preferences), balancing the workload evenly among nurses, etc. See Burke et al. [27].
- *Network Design*: In telecommunications or transportation networks, optimizing the layout of networks to minimize cost while satisfying bandwidth or connectivity requirements is a typical combinatorial optimization task (Guihaire and Hao [69]).

3.4 Combinatorial Problems and $\mathcal{NP}$ -Hardness

Many combinatorial optimization problems are classified as $\mathcal{NP}$ -hard, indicating that finding an optimal solution for these problems is computationally challenging, with no known polynomial-time algorithm that can solve all instances efficiently (Du and Ko [45]). Understanding $\mathcal{NP}$ -hardness is crucial in assessing the computational difficulty of such problems and in choosing the right approaches to solve them.

Example 1 *TSP is one of the best-known $\mathcal{NP}$ -hard problems in combinatorial optimization. Given n cities, the number of possible routes grows factorially, $\mathcal{O}(n!)$, making it computationally infeasible to solve for large n .*

Example 2 *The Knapsack Problem is $\mathcal{NP}$ -hard because the number of subsets of items grows exponentially as 2^n , where n is the number of items, leading to a combinatorial explosion in the number of possible solutions.*

For these reasons, for large problem instances, *exact algorithms* that guarantee optimal solutions may take an exponential amount of time, making them impractical for real-world use. Therefore, practitioners often turn

to *approximation algorithms*, *heuristics*, or *metaheuristics* that can provide near-optimal solutions within reasonable time frames, even if these methods do not guarantee optimality.

3.5 Classical Solving Algorithms

Several classical algorithms have been developed to solve combinatorial optimization problems. These algorithms can be broadly classified into exact algorithms, heuristics, and metaheuristics.

3.5.1 Exact Algorithms

Exact algorithms are methods designed to find the optimal solution to a problem by exhaustively exploring the entire solution space or using advanced mathematical techniques to guarantee optimality.

- *Branch and Bound*: This is a tree-based algorithm that systematically explores the solution space by partitioning it into smaller subproblems (Fischetti [53]). It guarantees finding the optimal solution but can be computationally expensive for large instances due to the exponential growth of the search tree. Commercial solvers like CPLEX and Gurobi are highly effective for small- to medium-sized problems but can struggle with large-scale instances due to the combinatorial explosion of integer solutions.

The complexity of the branch and bound algorithm can vary significantly depending on the specific problem being solved and the structure of the search space. In general, this algorithm does not have a fixed complexity. However, in the worst case, its complexity can be exponential, particularly for combinatorial optimization problems like the TSP or the knapsack problem. This is often expressed as $\mathcal{O}(b^d)$ where b is the branching factor (the number of child nodes generated from each node) and d is the depth of the tree (the number of decision variables).

- *Cutting Planes Method*: This iterative algorithm starts with the linear relaxation of the problem, which ignores the integer constraints, and adds cutting planes (linear inequalities) to exclude non-integer solutions until an integer optimal solution is found (Fischetti [53]). The cutting plane method may require a large number of cuts to converge. It has exponential complexity, indeed, in the worst case, the number of cutting planes needed to solve an integer programming problem is

exponential: $\mathcal{O}(n^m)$, where n is the number of decision variables and m is the number of constraints.

- *Branch and Cut*: It is an extension of the branch and bound method, augmented with cutting planes, to efficiently prune the search space and improve convergence to an optimal solution. See Kazachkov et al. [96].
Similar to branch and bound, the worst-case complexity can be exponential, often represented as $\mathcal{O}(b^d)$.
- *Benders Decomposition*: This is an algorithm designed to solve large-scale mixed-integer optimization problems by breaking them into two subproblems: a master problem (involving integer variables) and a subproblem (often continuous). By solving the subproblem iteratively and passing information (cuts) back to the master problem, Benders decomposition reduces the complexity of solving large-scale problems (Rahmaniani et al. [131]). It is especially useful in applications like network design and facility location.
The overall complexity is not easily characterized in big-O notation because it depends on the number of iterations needed, the complexity of the master and subproblem, and the effectiveness of the cuts.
- *Dynamic Programming*: This approach breaks a problem into smaller subproblems and solves them recursively (Hubert et al. [82]). It is used when a problem has an optimal substructure, where the solution can be built from solutions to subproblems. Problems like the Knapsack problem and the Shortest Path problem are commonly solved using dynamic programming. However, its applicability is limited to problems that exhibit optimal substructure and overlapping subproblems.

Remark 1 *In addition to general exact algorithms, there are also specific exact algorithms devised for particular problems. Examples include the Hungarian Method (Kuhn [105]) for the assignment problem and Dijkstra’s Algorithm (Javaid [89]) for shortest path problems. These specialized algorithms are often more efficient and tailored to the unique structure of the problems they address.*

3.5.2 Metaheuristics

Metaheuristics are high-level frameworks designed to provide sufficiently good solutions to complex optimization problems and explore the solution space

more thoroughly. It is worth noticing that metaheuristics are problem-independent, meaning that they can be applied to a wide variety of optimization problems without significant modification. Common metaheuristics include:

- *Simulated Annealing*: This method simulates the physical process of annealing, where a material is slowly cooled to find a low-energy state (Bertsimas and Tsitsiklis [20]). In optimization, it allows occasional uphill moves to escape local optima. It is simple and effective for various problems but may be slow for large search spaces.
Average Time Complexity: $\mathcal{O}(n \cdot \log(n))$ to $\mathcal{O}(n^2)$, depending on the cooling schedule and the problem.
- *Tabu Search*: Tabu search enhances local search by maintaining a list of recently visited solutions (tabu list) to prevent cycling back (Glover [62]). It is particularly effective for problems with complex solution landscapes but requires careful management of the tabu list to avoid excessively restricting the search.
Average Time Complexity: $\mathcal{O}(n^2)$ to $\mathcal{O}(n^3)$, depending on the implementation and neighborhood structure.
- *Genetic Algorithms*: These are inspired by natural selection and evolution. A population of solutions evolves over generations through selection, crossover, and mutation operators. Genetic algorithms are versatile and can solve complex problems, but their performance depends heavily on parameter tuning and may converge slowly. See Kramer [102].
Average Time Complexity: $\mathcal{O}(g \cdot n \cdot m)$, where g is the number of generations, n is the population size, and m is the number of fitness evaluations per generation.
- *Ant Colony Optimization (ACO)*: Inspired by the behavior of ants finding paths to food, ACO uses a population of agents (ants) to explore the solution space, depositing pheromones on good solutions (Dorigo et al. [43]). Over time, more ants follow the paths with higher pheromone levels. ACO works well for routing and network design problems but can be slow to converge.
Average Time Complexity: $\mathcal{O}(m \cdot n^2)$, where m is the number of ants and n is the number of nodes in the graph.

3.5.3 Heuristics

Unlike metaheuristics, a heuristic is typically tailored to a specific type of problem or class of problems. See Sánchez et al. [142] for a systematic review.

Example 3 *The Clarke-Wright Savings heuristic (Doyuran and Çatay [44]) is a well-known heuristic for solving the Capacitated Vehicle Routing Problem (CVRP). This heuristic starts with a solution where each customer is served by a separate vehicle, then iteratively merges routes to reduce costs based on the savings obtained from serving two customers with the same route instead of separate ones.*

Example 4 *Greedy Randomized Adaptive Search Procedure (GRASP): It is a multi-start heuristic that combines greedy and randomized approaches. In the construction phase, a solution is built using a greedy function, but a randomized element is introduced to explore different regions of the solution space. In the local search phase, the solution is refined by searching its neighborhood. GRASP is particularly useful in combinatorial optimization problems like graph coloring and network design, where both greedy and local search approaches can complement each other.*

It is essential to acknowledge that the solutions generated by heuristics can vary in quality, and for some problems, the gap between the heuristic solution and the optimal solution may be significant.

3.6 Pros and Cons of Classical Solving Algorithms

The application of classical algorithms to real-world combinatorial problems has several advantages:

- Exact algorithms provide exact solutions, ensuring that the global optimum is found, which is critical in scenarios where optimal decisions have significant implications (*e.g.*, safety, engineering, cost savings, etc.).
- Unlike exact algorithms, which may become intractable for large problems, heuristics and metaheuristics can handle large instances efficiently, providing good (if not optimal) solutions in a reasonable amount of time.

- Metaheuristics like genetic algorithms and SA are problem-agnostic and can be applied to a wide range of optimization problems, requiring minimal problem-specific adjustments.

Nonetheless, they also encompass a wide list of limitations:

- While exact algorithms are effective for small to medium-sized problems, their computational complexity often grows exponentially with the problem size, making them impractical for real-world large-scale problems.
- Heuristics may quickly find feasible solutions, but they offer no guarantees about how close the solution is to the optimum. In critical applications, relying on heuristics can be risky.
- The performance of metaheuristics heavily depends on choosing the right parameters (*e.g.*, population size in genetic algorithms, cooling schedule in simulated annealing). Poor tuning can lead to suboptimal solutions or excessive computation times.

In conclusion, while combinatorial optimization is a powerful tool for solving complex problems across various domains, it also presents significant challenges due to its inherent computational difficulty. Traditional exact and heuristic methods can struggle to deliver optimal solutions within reasonable time frames, particularly for large-scale instances. This calls for the exploration of innovative approaches to enhance solution efficiency and effectiveness. One promising avenue lies in the realm of quantum computing, which has the potential to revolutionize how we tackle combinatorial problems. By harnessing the principles of quantum mechanics, we may unlock new algorithms that can process vast solution spaces more rapidly than classical counterparts, paving the way for breakthroughs in optimization that were previously thought to be out of reach.

3.7 Practical Applications: VRP

3.7.1 Literature

The *Vehicle Routing Problem* (VRP) is an optimization problem that involves determining the optimal set of routes for a fleet of m vehicles (each with same capacity) to deliver goods to a given set of n customers.

The objective is to minimize the total distance (or cost) traveled by all vehicles. The VRP becomes computationally intractable as the number of

customers (or nodes n) and vehicles (m) increases due to the exponential growth in the number of possible routes.

Several studies have examined when VRP instances become too large to solve optimally with exact algorithms. These studies indicate that problem instances with 50 to 100 customers already challenge exact solvers, while larger instances typically require heuristic or metaheuristic methods. For example, Solomon [149] suggests that exact methods become intractable when the number of customer nodes reaches around 100. Golden et al. [63] further indicate that instances with more than 200 customers are generally solved using heuristics. As the number of customers and vehicles increases, the use of approximate methods, such as genetic algorithms, tabu search, or simulated annealing, becomes necessary to find approximate solutions within a reasonable time frame (within minutes to hours).

The computational complexity increases with the number of vehicles (n) as well. Instances with 5 to 10 vehicles typically require more sophisticated solvers. The number of vehicles affects the complexity of coordinating multiple routes. Cordeau et al. [35] note that solving VRP instances with more than 5 vehicles and around 100 customers becomes intractable for exact methods, further supporting the shift to heuristic approaches.

In summary, VRP becomes intractable for exact solvers around $n = 100$ customers and $5 < m < 10$ vehicles. For larger instances ($n > 200$), heuristic and metaheuristic methods are typically required.

3.7.2 Spindox Approach: A Metaheuristic-based VRP Solver

To address the challenges of solving large-scale VRP instances, we employ a custom metaheuristic-based solver. Our approach is designed to handle not only the classical VRP constraints but also additional practical constraints often encountered in real-world logistics scenarios. These include:

- *Maximum distance between two consecutive stops:* the solver enforces a user-defined maximum allowable distance between two consecutively visited customers, ensuring routes remain geographically compact.
- *Maximum time between two points:* besides distance, the solver also supports time-based constraints, limiting the maximum time a vehicle can travel between two consecutive stops.
- *Visit sequence for priority customers:* For certain products (*e.g.*, medical supplies), the solver prioritizes specific customer types (*e.g.*, hos-

pitals) before allowing visits to other locations, following predefined sequences.

- *Maximum waiting time at locations*: if a vehicle arrives before the opening of a customer’s time window, the solver ensures that the maximum waiting time does not exceed a user-defined threshold, optimizing vehicle schedules and reducing idle time.
- *Maximum number of stops per route*: the number of stops on each route is capped, which helps distribute the workload among vehicles evenly and ensures operational feasibility.
- *Other constraints*: our approach is flexible and can incorporate additional constraints such as vehicle capacity, delivery priority, and time windows.

Thanks to these advanced features, our solution can efficiently handle instances involving up to 1000 customers and approximately 100 vehicles, with solving times ranging from minutes to a few hours depending on the complexity of the instance and constraints.

3.8 Practical Applications: Scheduling Problems

3.8.1 Literature

A *scheduling* problem involves the allocation of tasks (referred to as *jobs*) to resources (typically referred to as *machines*) with the aim of optimizing one or more objectives, such as minimizing completion time or maximizing resource utilization. These problems arise in numerous industrial sectors, such as manufacturing, logistics, and computing systems, and represent a significant challenge for companies seeking to enhance productivity and reduce costs.

The main variants of scheduling problems include:

- *Parallel Machine Scheduling Problem (PMSP)*: in this simplified scenario, all machines are identical, and each job consists of a single task. The goal is to assign jobs to machines and define their execution order. Despite its simplicity, this problem becomes *NP-hard* when there are at least three machines, making it difficult to solve efficiently.
- *Job Shop Scheduling Problem (JSSP)*: a more complex case where each job consists of multiple operations, each of which must be executed on

different machines in a predefined order. This model is representative of modern factories that produce diverse products, each requiring a specific workflow.

- *Flow Shop Scheduling Problem (FSSP)*: a variant of JSSP where the order of operations is the same for all jobs. This type of problem is common in industries like baking, where the production flow follows the same path for all products, even if the processing times vary.
- *Open Shop Scheduling Problem (OSSP)*: in this variant, the order of operations is not fixed, providing more flexibility than JSSP. However, the absence of a predefined sequence complicates optimal scheduling.

Many of these variants can be further complicated by factors such as sequence-dependent setup times (Flexible JSSP with sequence-dependent setup) or machines capable of performing multiple operation types. See Da Col and Teppan [39].

Most scheduling problems, even in their simplified forms, are $\mathcal{NP}$ -hard, meaning that no efficient algorithm is known to solve them in polynomial time for large instances. In academic settings, scheduling problems typically involve between 100 and 2,000 operations, but in real-world industrial applications, instances can easily exceed 100,000 operations within a planning horizon, such as a week. Solving such large-scale problems requires a balance between solution quality and computational time.

Exact methods, such as *Mixed-Integer Linear Programming (MILP)* and *Branch and Bound*, are capable of providing optimal solutions by exhaustively exploring the solution space. However, their computational complexity grows exponentially with the problem size. This limits their practical applicability to small or medium-sized instances (typically a few hundred operations at most). For real-world scenarios involving thousands or more operations, exact methods become infeasible due to excessive computational time and memory requirements.

To tackle larger instances, heuristic and metaheuristic methods are often employed. Techniques like *Genetic Algorithms (GA)*, *Simulated Annealing (SA)*, and *Tabu Search* offer near-optimal solutions by exploring the solution space more efficiently than exact methods. These methods do not guarantee finding the global optimum but can handle much larger problem sizes (up to tens or hundreds of thousands of operations).

Their limitations, however, lie in the quality of the solutions, which may vary depending on the problem structure and the specific parameters used in the algorithm. Additionally, while heuristics can provide good solutions in a reasonable time, they often require significant fine-tuning and may not

perform equally well across different types of scheduling problems. Moreover, they may still struggle with capturing specific constraints relevant to real-world industrial systems, such as setup times or machine-specific processing restrictions.

In summary, while exact methods offer precision at the cost of scalability, heuristic approaches provide scalability at the expense of optimality and sometimes solution consistency. For industrial-scale scheduling problems, the choice of method often involves a trade-off between computational feasibility and solution quality.

3.8.2 Spindox Approach: A Multi-Resource Flexible Job-Shop Scheduling problem with Non-Linear Routing and Sequence-Dependent Setup Times

To bridge the gap between the problems studied in academic literature and large-scale practical applications, we have developed a scheduling variant that incorporates all the challenges faced by our customers.

The *Multi-Resource Flexible Job-Shop Scheduling problem with Non-Linear Routing and Sequence-Dependent Setup Times* aims to include the following complex real-world features:

- *Multi-Resource* – Each task (or operation) of a job may require more than one resource (or machine) to be completed;
- *Flexibility* – More than one resource (machine) can perform the same task;
- *Skill Factors* – Resources have different processing times for tasks (*e.g.*, to model seniority or expertise levels);
- *Resource Availability* – Resources are subject to availability constraints, such as maintenance, shifts, or operational downtimes;
- *Resource Priority* – When multiple resources are eligible for a task, the one with higher priority is chosen;
- *Release and Due Dates* – Jobs must be processed within a specified time window;
- *Job Priority* – Higher-priority jobs should be scheduled before lower-priority jobs;

- *Non-Linear Routing* – The sequence of tasks within a job may follow a non-linear structure (*e.g.*, parallel sequences that eventually converge or logistic flows separate from the main production line);
- *Sequence-Dependent Setup Times* – Machines incur a waiting period when switching between tasks from different jobs, modeling setup or cleaning times.

The approach we’ve chosen is heuristic and based on constraint programming, allowing us to strike an optimal balance between performance, solution quality, and the method’s expressiveness and scalability.

Considering the average case study of a customer, this method can tackle problems with hundreds of jobs and hundreds of machines, providing a good-quality solution with a wall time ranging from minutes to hours, depending on the active features and their correlations.

Chapter 4

State of the Art: Machine Learning Algorithms

Machine Learning (ML) is a subset of artificial intelligence that focuses on the development of algorithms that allow computers to learn from and make predictions or decisions based on data. See Goodfellow et al. [65]. In ML the general goal is to learn a model from a system with known inputs, and sometimes outputs, so that the model can predict or enhance understanding of the underlying data. In the most general case, there are two types of machine learning algorithms¹: supervised and unsupervised.

In the unsupervised case, only the inputs to a system are known and the goal is generally to better describe the data itself. In the supervised case, both the inputs and outputs of the system are known, and the goal is to accurately predict the systems outputs given the inputs.

4.1 Supervised Learning

Supervised learning is a type of machine learning where the algorithm learns from labeled data (Vapnik [158]). Given a training set T of N example input-output pairs:

$$T = (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n) \quad (4.1)$$

where each input x_i is associated with a corresponding target output y_i . The goal is to find a **hypothesis function** $h(x)$ that best approximates the

¹In addition to supervised and unsupervised learning, there are also other specialized paradigms such as reinforcement learning, semi-supervised learning, and self-supervised learning, which address more specific tasks or scenarios.

unknown true function $f(x)$, which generated the data. The relationship between input and output is often modeled as:

$$y_i = h(x_i) + \epsilon_i \quad (4.2)$$

where ϵ_i is the error term for each observation i , assumed to be independently and identically distributed with a mean of zero (Bishop [21]). Supervised learning problems are broadly divided into **regression** and **classification**:

Regression In a regression task, the target output y is a continuous variable. The objective of the algorithm is to predict values within a continuous range, such as predicting house prices, temperature, or stock prices. In this case, the hypothesis function $h(x)$ is designed to predict numerical values that approximate the actual output.

Classification In a classification task, the target output y is a discrete label, often representing distinct classes or categories. The goal is to assign each input x to one of the predefined classes, such as classifying emails as spam or not spam, or identifying types of animals in images. The hypothesis function $h(x)$ outputs a probability or a class label, representing the predicted category for each input.

By distinguishing between regression and classification, supervised learning algorithms can be tailored to the type of output variable, allowing for effective solutions across diverse applications.

4.1.1 Regression Methods

The goal is to identify a hypothesis function $h(x)$ able to predict a continuous output variable, y . The objective is typically expressed as minimizing a **loss function**, often the *Mean Squared Error* (MSE, see Wallach and Goffinet [159]):

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - h(x_i))^2. \quad (4.3)$$

The choice of $h(x)$ varies depending on the type of regression method, as described below.

Linear Regression

The most basic form of regression is linear regression (Maulud and Abdulazeed [116]), where the relationship between the input features x and the

output y is modeled by a linear equation:

$$h(x) = \theta_0 + \sum_{j=1}^p \theta_j x_j = x^\top \theta \quad (4.4)$$

where $x = [1, x_1, x_2, \dots, x_p]^\top$ (including an intercept term), and $\theta = [\theta_0, \theta_1, \theta_2, \dots, \theta_p]^\top$ represents the vector of model parameters. The goal of linear regression is to find the parameters θ that minimize the MSE over all examples.

$$\text{Cost function: } J(\theta) = \sum_{i=1}^m (y_i - h(x_i))^2. \quad (4.5)$$

The optimal parameters θ in linear regression can be obtained by solving the **normal equation**:

$$\theta = (X^\top X)^{-1} X^\top y \quad (4.6)$$

where X is the matrix of input features and y is the vector of target values. This closed-form solution is computationally efficient for small datasets but can become computationally intensive with high-dimensional or large datasets due to matrix inversion.

Polynomial Regression

Polynomial regression Roziqin et al. [138] extend linear regression by introducing non-linear relationships between input features and output variables through polynomial terms. Although polynomial regression remains linear in the parameters, it models complex patterns by fitting a polynomial of degree d to the data:

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_d x^d. \quad (4.7)$$

In vector notation, the input x is transformed into a feature vector that includes polynomial terms, and the hypothesis function becomes:

$$h(x) = \sum_{j=0}^d \theta_j x^j = \phi(x)^\top \theta \quad (4.8)$$

where $\phi(x) = [1, x, x^2, \dots, x^d]^\top$ is the vector of transformed features. Polynomial regression is useful when the data exhibits non-linear relationships, but choosing an appropriate degree d is essential to avoid **overfitting** (fitting too closely to the training data) or **underfitting** (failing to capture the true pattern).

Logistic Regression (for Classification)

Although **logistic regression** (Stoltzfus [151]) is a classification technique, it is often discussed in the context of regression because it models the probability (which is a continuous value) of class membership as a function of the input features. Logistic regression uses a **sigmoid** (or logistic) function to map the linear combination of inputs to a probability between 0 and 1. For binary classification, the hypothesis function, which correspond to the probability that an instance $\mathbf{x}$ belong to the positive class, is:

$$h(x) = \sigma(x^\top \theta) = \frac{1}{1 + e^{-x^\top \theta}} = P(y = 1 | \mathbf{x}). \quad (4.9)$$

The decision boundary is defined by choosing a threshold (commonly 0.5), such that:

$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1 | \mathbf{X}) \geq 0.5 \\ 0 & \text{if } P(y = 1 | \mathbf{X}) < 0.5. \end{cases} \quad (4.10)$$

The objective of training is to find the parameters set, θ , so that the model estimates high probabilities for positive instances ($y = 1$) and low probabilities for negative instances ($y = 0$). This idea is pictured by the so called **log loss** (or **binary cross-entropy loss**) cost function:

$$\text{Log Loss} = -\frac{1}{N} \sum_{i=1}^N (y_i \log(h(x_i)) + (1 - y_i) \log(1 - h(x_i))). \quad (4.11)$$

Unlike linear regression, logistic regression lacks a closed-form solution due to the non-linear nature of the sigmoid function. However, it retains linearity in the model parameters, and the convexity of the log-loss cost function allows for the use of convex optimization techniques to find the optimal parameters. For multi-class classification, extensions like **softmax regression** are used, where the output is the probability distribution over multiple classes.

Optimization Methods for Regression Models

To train these regression models, a variety of optimization methods can be used, particularly when the closed-form solution (normal equation) is not feasible or the dataset is too large. Below are common techniques:

1. **Gradient Descent**: An iterative algorithm that adjusts model parameters by moving in the direction that minimizes the cost function.

- **Batch Gradient Descent:** Computes the gradient over the entire dataset. This approach is computationally intensive but provides stable convergence as showed in Tschöpe [155].
 - **Stochastic Gradient Descent (SGD):** Computes the gradient based on a single training example per iteration, which introduces noise in parameter updates but speeds up convergence, see Tschöpe [155].
 - **Mini-Batch Gradient Descent:** Computes the gradient over a small batch of training examples, balancing the stability of batch gradient descent with the speed of SGD, see Khirirat et al. [98].
2. **Singular Value Decomposition (SVD):** For models sensitive to multicollinearity (such as polynomial regression), SVD can provide a stable solution by decomposing the matrix X into singular values, allowing for effective parameter estimation even when X is near-singular (Mandel [115]).

These optimization methods allow regression models to efficiently learn from data, even in high-dimensional or large-scale scenarios.

4.1.2 Neural Network (NN)

Neural networks are highly flexible models capable of performing both regression and classification tasks by layering multiple neurons. The idea of NN is inspired by biological models, such as the brain, in which signals are processed by individual neurons connected to thousands of other neurons. The *perceptron*, developed by Rosenblatt [137], was the first artificial neural network. Until today versions of this building block, referred to as artificial neurons, are used for NN architectures. Such a neuron functions similarly to the hypothesis functions described for regression. It takes n inputs $x_1, \dots, x_n$ and has a single binary output y , also called *activation* as depicted in Figure 4.1. Every input has an assigned weight $w_i \in \mathbb{R}$. Additionally, the neuron is equipped with a bias $b \in \mathbb{R}$. The neuron's output is computed through:

$$y = g(z) \quad \text{where} \quad z = \sum_i w_i x_i + b \quad (4.12)$$

where $g(x)$ is called activation function. The output will then depend on the choice of this activation function; see Goodfellow et al. [65].

A *neural network* is organized into layers of interconnected neurons, where each layer serves a specific purpose in processing the input data. These layers are the backbone of a neural network's architecture, enabling it to model

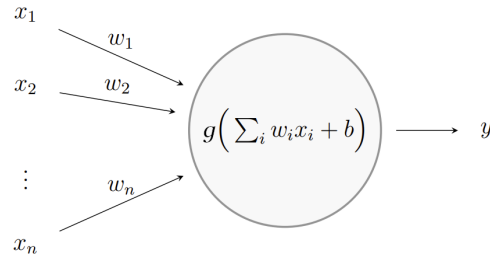


Figure 4.1: Artificial neuron. The building block of NNs.

complex relationships. A neural network with a single linear layer can approximate linear regression, while deeper networks with non-linear activation functions (*e.g.*, sigmoid or ReLU) can model polynomial and logistic regression characteristics, allowing it to handle both linear and non-linear relationships.

The network is typically divided into three main types of layers:

- **Input layer:** This is the first layer of the network, where raw input data is fed into the model. Each neuron in the input layer represents a feature of the input data (*e.g.*, pixels in an image or attributes in a dataset).
- **Hidden layers:** These are intermediate layers between the input and output layers, where most of the computation happens. Each neuron in a hidden layer takes inputs from all neurons in the previous layer. The inputs are combined using weights and biases as described above. The connections between neurons are weighted, meaning the importance of each connection is learned during training. A network can have multiple hidden layers, allowing it to model increasingly complex relationships in the data.
- **Output layer:** This is the final layer of the network, which produces the output. The number of neurons in this layer depends on the task:
 - For **regression**, it typically has a single neuron with a linear activation function to output a continuous value.
 - For **classification**, it has one neuron per class, often with a softmax activation function to produce class probabilities.

The mathematical formulation of a neural network is based on a sequence of computations across layers, with each layer transforming the input data to extract features and ultimately produce a prediction. A neural network can

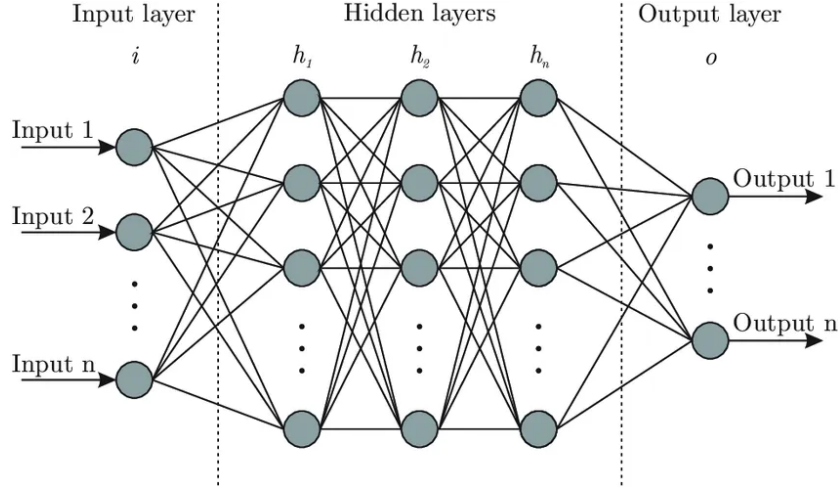


Figure 4.2: A neural network with three hidden layers.

be thought of as a function $f(x; \theta)$ where x is the input data and θ represents the trainable parameters (weights W and biases b).

The simplest NN architecture is the *feed-forward neural networks*, see Bebis and Georgiopoulos [16]. Here the neurons get the output of previous layers neurons as an input, and no loops are built-in. Furthermore, such NNs are often built of fully connected layers, *i.e.*, layers where all the inputs from one layer are connected to every neuron of the next layer. See Figure 4.2. The working of a feedforward neural network involves two phases: the feedforward phase and the backpropagation phase.

Feedforward Phase In this phase, the input data is fed into the network, and it propagates forward through the layers. At each hidden layer, l , each neuron receives inputs $h^{(l-1)}$ (outputs of the previous layer or the raw input for the first layer, $h^{(0)} = x$) and computes a weighted sum of its inputs, $W^{(l)}$, plus a bias term, $b^{(l)}$:

$$z^{(l)} = W^{(l)}h^{(l-1)} + b^{(l)} \quad (4.13)$$

The weighted sum $z^{(l)}$ is then passed through an *activation function* $g(\cdot)$ to introduce non-linearity:

$$h^{(l)} = g(z^{(l)}) \quad (4.14)$$

This step is crucial because it enables the network to model non-linear relationships in the data. Without activation functions, the network would be equivalent to a single-layer linear model, regardless of the number of layers (Hastie et al. [74]). Common activation functions include:

- **ReLU**: $g(z) = \max(0, z)$

- **Sigmoid:** $g(z) = \frac{1}{1+e^{-z}}$
- **Tanh:** $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

This process continues until the output layer is reached, and a prediction is made. The specific design of the output layer depends on the type of task:

- **Regression:** The output is continuous, often with no activation or a linear activation $g(z) = z$.
- **Classification:** The output is discrete, typically using *Sigmoid activation* for binary classification (output between 0 and 1) and *Softmax activation* for multiclass classification (output probabilities for each class).

The complete network function $f(x; \theta)$ is a composition of the functions of each layers $l = (1, 2, \dots, L - 1, L)$:

$$f(x; \theta) = g^{(L)} \left(W^{(L)} g^{(L-1)} \left(\dots g^{(1)} \left(W^{(1)} x + b^{(1)} \right) \dots \right) + b^{(L)} \right). \quad (4.15)$$

The weights $W^{(l)}$ and biases $b^{(l)}$ are learned during training, allowing the network to adapt to the data.

Backpropagation Phase Once a prediction is made in the output layer, the error is calculated by comparing the predicted outputs with the true labels using a chosen loss function. Common loss functions for neural networks include:

- **Mean Squared Error (MSE)** for regression:

$$\frac{1}{N} \sum_{i=1}^N (y_i - h^{(L)}(x_i))^2. \quad (4.16)$$

- **Binary Cross-Entropy Loss** for binary classification:

$$-\frac{1}{N} \sum_{i=1}^N [y_i \log(h^{(L)}(x_i)) + (1 - y_i) \log(1 - h^{(L)}(x_i))]. \quad (4.17)$$

- **Categorical Cross-Entropy Loss** for multiclass classification:

$$-\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^K y_{ij} \log(h_j^{(L)}(x_i)). \quad (4.18)$$

where $h^{(L)}$ is the prediction come out from the output layer.

Training a neural network involves adjusting the weights $W^{(l)}$ and biases $b^{(l)}$ across all layers to minimize the loss function. This is typically done through **backpropagation** (Rumelhart et al. [139]). This algorithm computes the gradient of the loss function with respect to each parameter in the network, layer by layer, starting from the output layer and moving backward to the input layer. Once the gradients are computed, **gradient descent** (or its variants like stochastic and mini-batch gradient descent) is used to update the parameters in the direction that reduces the loss:

$$W^{(l)} = W^{(l)} - \eta \frac{\partial J}{\partial W^{(l)}} \quad (4.19)$$

$$b^{(l)} = b^{(l)} - \eta \frac{\partial J}{\partial b^{(l)}} \quad (4.20)$$

where η is the learning rate, and J is the loss function.

The process is repeated for all training examples in the dataset (or a mini-batch if using mini-batch gradient descent). After processing all training examples, this constitutes one epoch. Training typically involves multiple epochs.

Neural networks provide a versatile framework for both regression and classification, capable of scaling from simple models with a few layers to highly complex architectures, such as **deep neural networks** with many hidden layers. The depth and non-linearity enable neural networks to perform well in a wide variety of real-world applications, from predicting continuous values to assigning probabilities to multiple classes.

4.1.3 Support Vector Machines (SVMs)

Support Vector Machines (SVMs), developed in the 1990s by Vapnik et al. [157], are versatile machine learning models primarily used for classification but also capable of handling regression and outlier detection. SVMs excel in classifying complex, small- to medium-sized datasets by identifying an optimal hyperplane that maximizes the margin between classes. This hyperplane, determined by support vectors (the data points closest to the boundary), provides a robust decision boundary, allowing SVMs to generalize effectively on new data.

When the data is linearly separable, the SVM finds a straightforward hyperplane in the input feature space. For non-linear data, SVMs apply the *kernel trick*, mapping data into a higher-dimensional space to enable separation. Common kernel functions include linear, polynomial, radial basis

function (RBF), and sigmoid kernels, with the choice depending on the data characteristics and specific application.

Although SVMs can handle regression, they are most effective in classification tasks, finding the optimal hyperplane across an N-dimensional feature space to separate classes. The dimension of this hyperplane aligns with the number of features, making SVMs flexible across varied datasets and dimensionalities.

Linear SVM

In linear SVMs, designed for linearly separable data, the algorithm identifies a hyperplane that divides classes without transforming the data. This hyperplane maximizes the distance between classes, with support vectors forming the edges.

For binary classification, given training data points $(\mathbf{x}_i, y_i)$ where $y_i \in \{-1, 1\}$, the hyperplane is defined by:

$$\mathbf{w}^\top \mathbf{x} + b = 0 \quad (4.21)$$

where $\mathbf{w}$ is the weight vector normal to the hyperplane, b is the bias and $\frac{b}{\|\mathbf{w}\|}$ is the perpendicular distance from the hyperplane to the origin.

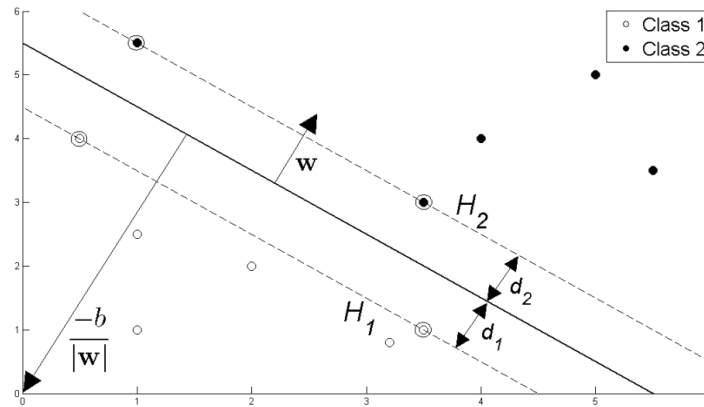


Figure 4.3: Hyperplane through two linearly separable classes

The goal is to maximize the **margin**, which is the distance between the hyperplane and the closest data points (support vectors):

Support Vectors are the examples closest to the separating hyperplane and the aim of SVM is to orientate this hyperplane in such a way as to be as far as possible from the closest members of both classes. This is equivalent

to selecting the variables $\mathbf{w}$ and b so that the training data can be described by:

$$\mathbf{w}^\top \mathbf{x}_i + b \geq +1 \quad \text{for } y_i = +1 \quad (4.22)$$

$$\mathbf{w}^\top \mathbf{x}_i + b \leq -1 \quad \text{for } y_i = -1. \quad (4.23)$$

These equations can be combined into:

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \quad \forall i. \quad (4.24)$$

If we now just consider the points that lie closest to the separating hyperplane, *i.e.*, the Support Vectors, then the two planes H_1 and H_2 that these points lie on can be described by:

$$\mathbf{w}^\top \mathbf{x}_i + b = +1 \quad \text{for } H_1 \quad (4.25)$$

$$\mathbf{w}^\top \mathbf{x}_i + b = -1 \quad \text{for } H_2. \quad (4.26)$$

Referring to Figure 4.3, we define d_1 as being the distance from H_1 to the hyperplane and d_2 from H_2 to it. The hyperplane's equidistance from H_1 and H_2 means that $d_1 = d_2$ —a quantity known as the SVM's **margin**. In order to orientate the hyperplane to be as far from the Support Vectors as possible, we need to maximize this margin. Simple vector geometry shows that the margin is equal to $\frac{1}{\|\mathbf{w}\|}$ and maximizing it subject to the constraint in 4.24 is equivalent to finding:

$$\text{Minimize } \|\mathbf{w}\| \quad \text{such that} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 \geq 0 \quad \forall i. \quad (4.27)$$

Minimizing $\|\mathbf{w}\|$ is equivalent to minimizing $\frac{1}{2}\|\mathbf{w}\|^2$ and the use of this term makes it possible to perform *Quadratic Programming* (QP) optimization. We therefore need to find:

$$\text{Minimize } \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{such that} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 \geq 0 \quad \forall i. \quad (4.28)$$

In order to extend the SVM methodology to handle data that is not fully linearly separable, the constraints in equations 4.22 and 4.23 are slightly relaxed to allow for misclassified points (Cortes and Vapnik [37]). This is achieved by introducing a positive slack variable ξ_i , $i = 1, \dots, L$:

$$\mathbf{w}^\top \mathbf{x}_i + b \geq +1 - \xi_i \quad \text{for } y_i = +1 \quad (4.29)$$

$$\mathbf{w}^\top \mathbf{x}_i + b \leq -1 + \xi_i \quad \text{for } y_i = -1 \quad (4.30)$$

$$\xi_i \geq 0 \quad \forall i. \quad (4.31)$$

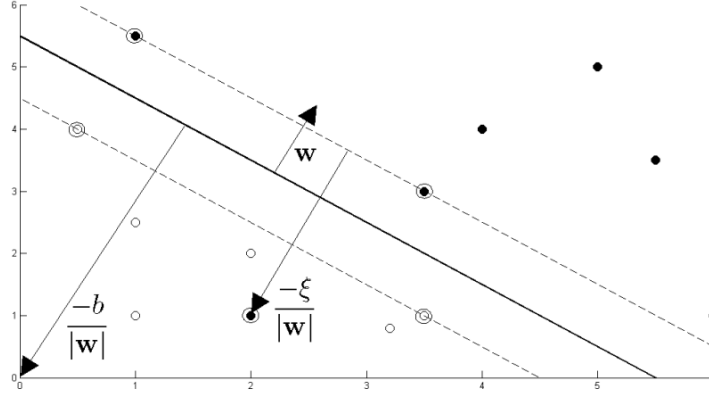


Figure 4.4: Hyperplane through two not fully linearly separable classes

Which can be combined into:

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 + \xi_i \geq 0 \quad \forall i. \quad (4.32)$$

In this *soft margin* SVM, data points on the incorrect side of the margin boundary have a penalty that increases with the distance from it. As we are trying to reduce the number of misclassifications, a sensible way to adapt our objective function 4.28 from previously, is to find:

$$\text{Minimize } \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^L \xi_i \quad \text{such that} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 + \xi_i \geq 0 \quad \forall i. \quad (4.33)$$

Where the parameter C controls the trade-off between the slack variable penalty and the size of the margin.

Non-linear SVM

When the data is not linearly separable in its original space, we can map it to a higher-dimensional space where it might become linearly separable. Let $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^d$ be a feature mapping function that transforms the input vector $\mathbf{x}$ into a higher-dimensional space:

$$\mathbf{x} \rightarrow \phi(\mathbf{x}). \quad (4.34)$$

Now, in the transformed space, the decision function becomes:

$$f(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x}) + b. \quad (4.35)$$

Directly computing $\phi(\mathbf{x})$ and then finding the separating hyperplane in the high-dimensional space can be computationally expensive, especially if the transformation increases the dimensionality significantly. For example, mapping a 2D dataset to a 1000-dimensional space would result in large computational costs.

The *kernel trick* provides a solution by allowing us to compute inner products in the transformed space without explicitly computing $\phi(\mathbf{x})$. Instead of directly mapping $\mathbf{x}_i$ and $\mathbf{x}_j$ to $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$, we use a **kernel function** $k(\mathbf{x}_i, \mathbf{x}_j)$ that computes the inner product of these transformed vectors (Schölkopf et al. [144]):

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j). \quad (4.36)$$

By using this kernel function, we can work in the high-dimensional feature space implicitly, avoiding the explicit computation of $\phi(\mathbf{x})$. This allows us to use complex transformations without the computational cost of directly handling high-dimensional vectors.

Different kernel functions are used to implicitly map data into higher-dimensional spaces. Some popular kernel functions are:

- **Linear Kernel:** For cases where the data is already linearly separable.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j. \quad (4.37)$$

- **Polynomial Kernel:** Useful for polynomial decision boundaries.

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + c)^d \quad (4.38)$$

where c is a constant and d is the degree of the polynomial.

- **Radial Basis Function (RBF) or Gaussian Kernel:** Widely used for non-linear classification.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2) \quad (4.39)$$

where γ is a parameter that defines the spread of the kernel.

- **Sigmoid Kernel:** Often used in neural networks.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\alpha \mathbf{x}_i \cdot \mathbf{x}_j + c) \quad (4.40)$$

where α and c are kernel parameters.

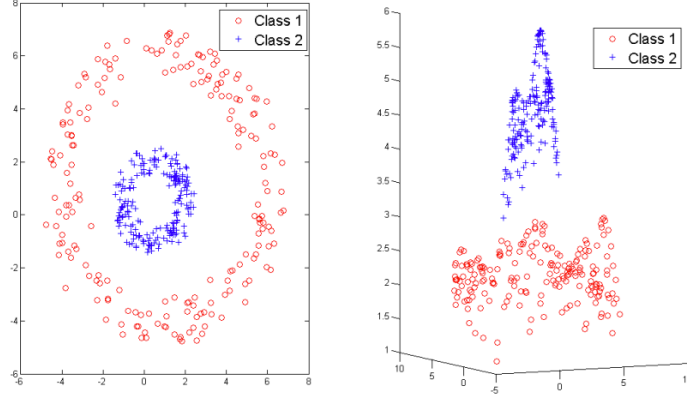


Figure 4.5: Dichotomous data re-mapped using Radial Basis Kernel

Using the kernel trick, the optimization problem for SVMs can be expressed in terms of the kernel function. The objective function for training a nonlinear SVM becomes:

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^n \alpha_i \quad (4.41)$$

subject to:

$$\sum_{i=1}^n \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C \quad (4.42)$$

where:

- α_i are the Lagrange multipliers,
- y_i are the class labels (± 1),
- C is the regularization parameter that controls the trade-off between maximizing the margin and minimizing classification error.

The dual formulation of the SVM problem allows us to solve it by calculating $k(\mathbf{x}_i, \mathbf{x}_j)$ for pairs of training points, using the kernel function to implicitly work in the higher-dimensional space without explicitly transforming the data.

Once the SVM is trained, the decision function for a new input $\mathbf{x}$ is given by:

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b \quad (4.43)$$

where only the support vectors (*i.e.*, the points for which $\alpha_i > 0$) contribute to the sum. This allows SVMs to make predictions by evaluating the kernel function between the new input and the support vectors, again without needing to compute $\phi(\mathbf{x})$ directly.

4.1.4 Other Supervised Learning Models

In addition, several other supervised learning methods are worth mentioning.

Decision Trees They are structured models that make predictions by recursively splitting the feature space based on feature values. The goal of each split is to partition the data into subsets that are as homogeneous as possible with respect to the target variable. In a Decision Tree, each internal node represents a test on a feature, and each branch corresponds to the outcome of that test. At each step, the algorithm evaluates all possible splits and selects the one that minimizes a specific impurity measure. For classification, common measures include Gini impurity and entropy, while for regression, the MSE or *Mean Absolute Error* (MAE) is typically used. Decision trees are easy to interpret and visualize, as each path from the root to a leaf represents a rule that can be directly understood by humans. They also handle both numerical and categorical data and require minimal data preprocessing. However, they tend to overfit the training data, especially when they become deep with many splits, and can be highly sensitive to small changes in data. Pruning techniques can be applied to limit tree depth and reduce overfitting, although this may reduce interpretability. See Gordon et al. [66].

K-Nearest Neighbors (KNN) It is a lazy learner, meaning it doesn't construct an explicit model during training. Instead, it stores the training instances and makes predictions based on the closest K training samples (neighbors) to a given input. A distance metric, such as Euclidean or Manhattan distance, is used to measure proximity. For classification, KNN assigns the majority class label among neighbors, and for regression, it averages the neighbors' output values. KNN is simple to implement, requires no training time, and can capture complex decision boundaries with enough neighbors. It is also highly flexible, as it can be adapted for both classification and regression. However, it becomes computationally expensive as the dataset size grows because it has to compute the distance to all training points for each prediction. It's also sensitive to irrelevant features and data scaling; normalization or

feature selection can help improve performance. See Cover and Hart [38].

Naive Bayes It is a **probabilistic classifier** based on Bayes’ theorem, which calculates the probability of each class given an input. The “naive” part of Naive Bayes assumes that all features are conditionally independent given the class label, simplifying computation by estimating individual feature probabilities. There are different variants depending on data type: *Gaussian Naive Bayes* for continuous data, *Multinomial Naive Bayes* for count data (*e.g.*, word frequencies), and *Bernoulli Naive Bayes* for binary data. It is computationally efficient, especially with high-dimensional data, and performs well in text classification tasks despite the independence assumption. It can handle large datasets with many features, making it particularly suitable for document and spam classification. However, the strong independence assumption can lead to suboptimal performance if features are highly correlated. Naive Bayes often still achieves reasonable accuracy due to its robustness in many cases. See Zhang [172].

Ensemble Methods Like Random Forests and Gradient Boosting, they combine multiple models to achieve higher accuracy and robustness than individual models. Random Forests reduce overfitting by aggregating predictions from multiple decision trees, while Gradient Boosting sequentially builds trees to correct the errors of previous trees. These methods are highly effective in real-world applications and are commonly used in predictive analytics and competition-grade modeling. See Breiman [25], Friedman [55]

4.2 Unsupervised Learning

Unsupervised learning is a branch of ML that involves training models on data without labeled outputs (Naeem et al. [122]). The primary objective is to uncover hidden structures, patterns, and relationships in the data. Unlike supervised learning, where training data includes input-output pairs, unsupervised learning focuses on analyzing data where the outcomes are unknown. Two key areas within unsupervised learning are clustering and dimensionality reduction.

Clustering It is a technique used to group similar data points together into clusters, allowing for the discovery of inherent groupings within the

data (Rodriguez et al. [135]). This can be useful in various applications such as market segmentation, anomaly detection, and pattern recognition. One of the most widely used methods for clustering is **K-means**, which partitions data into distinct groups based on similarities among the data points.

Dimensionality Reduction It refers to methods that reduce the number of features (or variables) in a dataset while preserving as much information as possible (Mladenić [118]). This helps to simplify the analysis and visualization of complex, high-dimensional data. A key technique for dimensionality reduction is **Principal Component Analysis** (PCA), which transforms data into a smaller number of uncorrelated components that capture the maximum variance of the original data, facilitating easier visualization and interpretation.

4.2.1 K-means Clustering

K-means clustering is a widely used algorithm for partitioning data into a predefined number of clusters. It falls under the category of centroid-based clustering, where each cluster is represented by the centroid (mean) of the data points within it (Chong [34]). The algorithm aims to minimize the variance within each cluster by iteratively assigning data points to clusters based on their distance from the centroid and recalculating the centroids.

Let the dataset $X = \{x_1, x_2, \dots, x_n\}$ contain n data points, and let each point be assigned to one of K clusters $\{C_1, C_2, \dots, C_K\}$. The algorithm aims to minimize the within-cluster variance.

Algorithm Steps

1. *Initialization*: Choose K initial centroids $\mu_1, \mu_2, \dots, \mu_K$ (randomly or using a heuristic like K-means++).
2. *Assignment*: Assign each point x_i to the cluster with the nearest centroid μ_j , based on the Euclidean distance:

$$C_i = \{x_j : \|x_j - \mu_i\|^2 \leq \|x_j - \mu_k\|^2, \forall k = 1, \dots, K\}.$$

3. *Update*: Update the centroids of the clusters by calculating the mean of all points assigned to each cluster:

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x.$$

4. *Repeat* steps 2 and 3 until convergence (when assignments no longer change).

The K-means algorithm minimizes the *within-cluster sum of squares (WCSS)*, defined as:

$$\min_{\{C_j\}} \sum_{i=1}^K \sum_{x \in C_i} \|x - \mu_i\|^2$$

where $\|\cdot\|$ denotes the Euclidean distance.

4.2.2 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique commonly used in unsupervised learning. PCA aims to reduce the number of variables in a dataset while retaining as much variance (information) as possible. It achieves this by transforming the original features into new, orthogonal features known as principal components. See Jolliffe and Cadima [93].

Given the dataset $X \in \mathbb{R}^{n \times d}$, where n is the number of samples and d is the number of features, PCA seeks to transform the data into a lower-dimensional space while preserving the maximum variance.

Algorithm Steps :

1. *Center the data* by subtracting the mean from each feature:

$$X_{\text{centered}} = X - \text{mean}(X).$$

2. *Compute the covariance matrix* of the centered data:

$$\Sigma = \frac{1}{n} X_{\text{centered}}^\top X_{\text{centered}}.$$

3. *Eigen decomposition*: Solve for the eigenvalues λ_i and eigenvectors v_i of the covariance matrix Σ :

$$\Sigma v_i = \lambda_i v_i.$$

4. *Project data* onto the top k eigenvectors with the highest eigenvalues:

$$X_{\text{new}} = X_{\text{centered}} V_k$$

where V_k is the matrix of the top k eigenvectors.

4.3 Limitations of ML Algorithms

Classical machine learning models face limitations in computational complexity. These limitations become particularly pronounced when scaling to large, high-dimensional datasets or in applications requiring rapid, real-time processing.

For instance, training *linear regression* models using the **normal equation** (see 4.6) requires computing the inverse of $X^\top X$, a $(d + 1) \times (d + 1)$ matrix (where d is the number of parameters). The complexity of inverting this matrix is typically between $\mathcal{O}(d^{2.4})$ and $\mathcal{O}(d^3)$, depending on the implementation (Bishop [21]). Similarly, the **SVD** approach has a time complexity of approximately $\mathcal{O}(d^2)$. Both methods become computationally demanding when the number of features grows large. However, both are linear with respect to the number of instances, n , in the training set (*i.e.*, $\mathcal{O}(n)$), making them efficient for large training sets—provided they fit in memory. **Gradient descent** methods, in contrast, avoid matrix inversion by iteratively updating parameters and are less sensitive to feature dimensionality (J and Bhandari [85]). Techniques like **SGD** and **mini-batch**, combined with parallel processing, allow gradient descent to handle millions of samples and tens of thousands of features.

SVM with kernel methods algorithms are also computationally intensive. Training requires solving a quadratic optimization problem, scaling roughly as $\mathcal{O}(n^2)$ to $\mathcal{O}(n^3)$ with respect to the number of samples, n (Burges [26]). This scaling makes SVMs impractical for datasets with $n \geq 10^4$ samples due to their quadratic and cubic complexity.

Extending classical ML to include *neural networks* and *deep learning* further expands model capabilities but also increases computational demands. These models often involve complex architectures, with deep learning layers containing millions of neurons. Training deep learning models involves optimizing millions (or even billions) of parameters, with time complexity depending on factors such as the number of layers L , neurons per layer h , epochs E , parameters d , and samples n . For a single-layer fully connected neural network, complexity is approximately $\mathcal{O}(n \cdot E \cdot h^2 \cdot d)$. For deep networks, this scales further, growing to $\mathcal{O}(n \cdot E \cdot L \cdot h^2 \cdot d)$. With increased depth and width, training can become prohibitively time-consuming, even with advanced hardware. See Goodfellow et al. [65].

Inference in classical ML models is typically more efficient than training, though real-time applications can still pose challenges. For instance, the *k-nearest neighbors* (*k-NN*) algorithm has high inference costs for large datasets, as it requires storing the entire training set and calculating distances for every query, scaling poorly with data size and dimensionality (Cover and

Hart [38]). The complexity, $\mathcal{O}(n \cdot d)$, makes k-NN inefficient as d increases; in high-dimensional spaces (*e.g.*, $d = 10^4$), each distance computation is computationally expensive. Likewise, large neural networks add inference complexity; for example, *Convolutional Neural Networks* (CNNs) used in high-resolution image recognition can struggle with latency in real-time applications due to computational demands. See He et al. [77].

Other algorithms also demonstrate similar computational concerns. The *k-means algorithm* has a time complexity of approximately $\mathcal{O}(n \cdot k \cdot i \cdot d)$, where i is the number of iterations and k is the number of clusters. This complexity grows as n , d , or k increases, particularly if k is large or convergence is slow. Additionally, multiple random initializations are often necessary to avoid suboptimal clustering. See Arthur and Vassilvitskii [12]. PCA requires calculating the covariance matrix and performing an eigenvalue decomposition, yielding a time complexity of $\mathcal{O}(d^2 \cdot n + d^3)$, where d is the number of features and n is the number of samples. The eigenvalue decomposition ($\mathcal{O}(d^3)$) makes PCA computationally challenging for high-dimensional data, though techniques like randomized PCA can improve efficiency on very large datasets. See Jolliffe [92].

These computational challenges highlight the need for more efficient solutions capable of managing the vast data sizes and high-dimensional feature spaces that characterize modern machine learning tasks. QML offers a promising alternative to classical approaches by leveraging the unique properties of quantum computing, such as superposition and entanglement, to perform certain calculations more efficiently than classical systems. Unlike traditional methods, QML has the potential to exponentially speed up tasks like matrix inversion, optimization, and large-scale data processing, which are crucial in machine learning. This advantage could allow QML to handle problems that would be infeasible for classical models, opening new possibilities for scalable and rapid analysis in applications where classical methods are computationally prohibitive. Although still in development, quantum machine learning represents a powerful emerging tool, potentially enabling breakthroughs across data-intensive fields.

Chapter 5

State of the Art: Quantum Algorithms

5.1 Fundamentals of Quantum Information

Quantum computing represents a revolutionary shift in the way we approach computation and problem-solving. While classical computers operate on bits, the basic units of information that take on binary states of 0 or 1, quantum computers utilize *quantum bits* or *qubits*. Unlike classical bits, qubits can exist in multiple states simultaneously, thanks to the principles of quantum mechanics. By harnessing these unique properties, quantum computers have the potential to solve certain types of problems far faster and more efficiently than classical computers could ever achieve.

At the heart of quantum computing is the idea of using quantum states to perform complex calculations. In classical computing, information is processed through a series of logical gates that manipulate binary bits. However, in quantum computing, information is processed through *quantum gates* that act on qubits. Quantum gates are different from their classical counterparts: they use quantum mechanics to enable superposition, entanglement, and interference—three core properties that give quantum computers their distinctive power.

The appeal of quantum computing lies in its potential to tackle challenges that are currently impractical for classical machines. Problems like prime factorization, complex optimization, and molecular modeling require enormous computational resources on classical computers. With quantum computing, however, these problems could be approached more feasibly, as qubits enable new computational pathways that dramatically reduce the time required for solutions.

First, we will present some foundational concepts of quantum mechanics, followed by a discussion of their specific applications in the field of quantum computing

5.1.1 Core Concepts of Quantum Mechanics

Quantum State

A quantum state is the complete mathematical description of a quantum system, containing all the information necessary to predict the probabilities of outcomes for any measurement performed on the system. Quantum states represent the physical properties of a system and are typically described by vectors in an abstract complex vector space. Quantum states can exhibit unique properties, such as superposition and entanglement, which distinguish them from classical states.

Hilbert Space

The Hilbert space is the mathematical structure in which quantum states reside. It is a complex vector space equipped with an inner product that allows for the computation of probabilities and expectations. Each state in a Hilbert space is represented by a vector, and quantum operations are represented by transformations (unitary operators) acting on this space. The dimensionality of the Hilbert space depends on the system: for a single particle, the Hilbert space may be finite-dimensional, while for systems with continuous variables, such as position and momentum, the space can be infinite-dimensional.

Basis States

Basis states are a set of reference states in Hilbert space that form an orthonormal basis, meaning any quantum state in the space can be represented as a linear combination (or superposition, see below) of these basis states. The coefficients in this combination represent the probability amplitudes, which determine the probabilities of measuring the system in each basis state. In a finite-dimensional Hilbert space, the basis states are often chosen as the eigenstates of a particular observable, such as energy or spin, and each measurement collapses the quantum state into one of these basis states.

Mathematical Representation of Quantum States

Dirac Notation Quantum states are often represented in Dirac notation, where $|\psi\rangle$ represents a column vector known as a *ket*, and $\langle\psi|$ represents its conjugate transpose, known as a *bra*. For example, a single-qubit state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ can be represented as:

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}. \quad (5.1)$$

Operators Quantum gates are represented as operators (matrices) that act on state vectors (see below). For instance, the *Pauli-X* gate¹ is represented by the matrix:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \quad (5.2)$$

Applying X to a qubit in state $|0\rangle$ results in $X|0\rangle = |1\rangle$.

State Vector

In quantum mechanics, the state vector is a mathematical representation of a quantum system in a pure state (see below). It is a vector in a complex Hilbert space, describing the complete information about the quantum system. For a system with basis states $|\phi_i\rangle$, the state vector $|\psi\rangle$ can be expressed as:

$$|\psi\rangle = \sum_i c_i |\phi_i\rangle,$$

where c_i are complex coefficients, and $|c_i|^2$ represents the probability of finding the system in the basis state $|\phi_i\rangle$. These coefficients satisfy the normalization condition $\sum_i |c_i|^2 = 1$.

The state vector provides a compact and powerful way to represent the quantum state of a system, particularly for pure states. It forms the basis for describing quantum phenomena like superposition and entanglement and is central to the mathematical framework of quantum mechanics.

Superimposition State

One of the fundamental principles of quantum mechanics is superposition, which allows a quantum system to exist in multiple states simultaneously.

¹which acts as a NOT operation.

For a system with two possible basis states, denoted as $|\phi_0\rangle$ and $|\phi_1\rangle$, the general state of the system can be written as:

$$|\psi\rangle = \alpha|\phi_0\rangle + \beta|\phi_1\rangle,$$

where α and β are complex coefficients satisfying $|\alpha|^2 + |\beta|^2 = 1$. These coefficients determine the probabilities of the system being found in $|\phi_0\rangle$ or $|\phi_1\rangle$ upon measurement, with $|\alpha|^2$ and $|\beta|^2$ representing the respective probabilities.

Superposition is a uniquely quantum phenomenon that allows the system to simultaneously “exist” in all possible configurations defined by the basis states. This property is a cornerstone of quantum mechanics, enabling quantum systems to explore multiple possibilities in parallel. However, when the system is measured, the superposition collapses into one of the basis states, with the outcome governed by the probability distribution defined by the coefficients.

Density Matrix

In quantum mechanics, the **density matrix** is a mathematical framework that describes the state of a quantum system. It is particularly useful for representing *mixed ensembles* (see below), where the system exists in a probabilistic combination of different pure states (see below), or when the information about the system is incomplete. For a quantum system with basis states $|\phi_i\rangle$, any pure state $|\psi\rangle$ can be written as a linear combination of the basis states:

$$|\psi\rangle = \sum_i c_i |\phi_i\rangle,$$

where c_i are complex coefficients satisfying $\sum_i |c_i|^2 = 1$. The density matrix ρ is then defined as:

$$\rho = \sum_j p_j |\psi_j\rangle \langle \psi_j|,$$

where $|\psi_j\rangle$ are the relevant pure states of the system in the given context, and p_j is the probability of the system being in each pure state, with $\sum_j p_j = 1$.

The density matrix generalizes the concept of a quantum state. While pure states are represented as a single vector $|\psi\rangle$, mixed ensembles, which arise when a system is in a probabilistic combination of states, are captured by the density matrix. The density matrix encodes both the probabilities

of the states and their coherences, making it a powerful representation for analyzing quantum systems, particularly in scenarios involving noise or entanglement.

Pure State

A pure state is a quantum state that has a complete and definite description in Hilbert space, represented by a single vector. Pure states are typically expressed as a linear combination of basis states, and their properties are fully determined by this vector. Mathematically, if a quantum state $|\psi\rangle$ is pure, its **density matrix** is given by $\rho = |\psi\rangle\langle\psi|$. The density matrix for a pure state satisfies² $\text{Tr}(\rho^2) = 1$, which distinguishes it from a mixed ensemble where $\text{Tr}(\rho^2) < 1$. Pure states can exhibit quantum superposition, allowing the system to exist in a combination of basis states simultaneously.

Mixed Ensembles

A mixed ensemble (sometimes ambiguously called mixed state) is a quantum state that represents a probabilistic mixture of different pure states, rather than a single state vector. Mixed ensembles arise when there is partial knowledge about the system, often due to interaction with an environment or inherent uncertainties. A mixed state is described by a **density matrix** ρ , which is a weighted sum of the density matrices of individual pure states:

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|,$$

where p_i represents the probability of each pure state $|\psi_i\rangle$. Unlike pure states, mixed states satisfy $\text{Tr}(\rho^2) < 1$, indicating that they are not fully coherent. Mixed states are essential for describing systems in thermal equilibrium, systems affected by noise, or any quantum system where there is incomplete knowledge about its exact configuration.

5.1.2 Core Concepts of Quantum Computing

Qubits

A qubit (quantum bit) is the fundamental unit of quantum information and are described by quantum states in a two-dimensional Hilbert space. The basis states for a qubit are often labeled as $|0\rangle$ and $|1\rangle$, corresponding to the

² $\text{Tr}(\rho)$ represents the *trace* of the matrix, defined as the sum of its diagonal elements.

classical binary states 0 and 1. However, unlike classical bits, a qubit can exist in a superposition of states (see below):

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad (5.3)$$

where $|0\rangle$ and $|1\rangle$ represent the two basis states, and α and β are complex probability amplitudes such that $|\alpha|^2 + |\beta|^2 = 1$.

For a mixed state of a qubit, the density matrix ρ represents a probabilistic mixture of pure states, capturing uncertainties in the state. This distinction between pure and mixed states in qubits is crucial for understanding the effects of decoherence, noise, and measurement outcomes in quantum computing.

Quantum Superposition

A qubit's ability to be in a superposition is one of its defining properties. This means that, until measured, a qubit exists simultaneously in both $|0\rangle$ and $|1\rangle$ states, with probabilities given by $|\alpha|^2$ and $|\beta|^2$. In computational terms, this means that a quantum computer with n qubits can represent 2^n states simultaneously.

This parallelism has enormous implications for processing power: while a classical computer would have to evaluate each possible state sequentially, a quantum computer can evaluate them all at once, exponentially increasing its ability to solve certain types of problems.

Bloch Sphere

The Bloch sphere is a geometric representation of the state of a single qubit. A qubit state can exist in any superposition of the basis states $|0\rangle$ and $|1\rangle$, and the Bloch sphere maps this state to a point on the surface of a 3D sphere. Any qubit state can be expressed as:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle, \quad (5.4)$$

where θ represents the angle from the north pole, and ϕ is the azimuthal angle around the equator. The north pole of the sphere corresponds to the $|0\rangle$ state, while the south pole represents $|1\rangle$. Points along the equator represent equal superpositions, such as $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

This visualization is particularly useful for understanding quantum gates, as each gate can be seen as a rotation on the Bloch sphere. For example, applying an X (NOT) gate corresponds to a 180° rotation around the x-axis, flipping $|0\rangle$ to $|1\rangle$.

Entanglement

Entanglement is a unique quantum phenomenon where the states of two or more qubits become correlated in such a way that the state of one qubit directly influences the state of the other(s). Entangled qubits share information in a unique way, where the measurement of one qubit immediately determines the state of its entangled partner(s). For instance, a pair of qubits can form a Bell state:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \quad (5.5)$$

where the measurement of one qubit immediately determines the state of the other.

Mathematically speaking, if we have two entangled qubits A and B in a joint quantum state, the probability $P(A = a | B = b)$ of measuring qubit A in state a , given that we measured qubit B in state b , differs from the independent probabilities. For entangled qubits, $P(A = a | B = b)$ does not equal $P(A = a)$ or $P(B = b)$, indicating that the outcome of one qubit measurement directly affects the other's probability distribution.

In other words, for an entangled state:

$$P(A = a \text{ and } B = b) \neq P(A = a) \cdot P(B = b),$$

demonstrating that their outcomes are not independent. This dependency defines entanglement, as opposed to separable states, where measurement outcomes would satisfy $P(A = a \text{ and } B = b) = P(A = a) \cdot P(B = b)$.

Entanglement enables complex correlations between qubits, which quantum algorithms can exploit to perform computations that require relationships or dependencies between data elements. This property is especially valuable in algorithms that rely on parallelized computations across qubits, such as certain quantum machine learning and optimization algorithms.

Quantum Measurement

In general, measurements provide an interface between quantum and classical information. When a measurement is performed on a system in a quantum state, classical information is extracted, revealing something about that quantum state, and generally changing or destroying it in the process. In particular, quantum measurement is the process by which the state of a qubit collapses to a definite value, either $|0\rangle$ and $|1\rangle$, when observed. This measurement process introduces a probabilistic element, as the outcome of a measurement is not deterministic but rather governed by the probabilities

associated with the qubit’s amplitudes. Because of this probabilistic nature, quantum computations often involve repeated measurements to gather a reliable distribution of results. This distribution allows us to infer solutions, and for certain algorithms, the goal is to maximize the probability of obtaining the correct result through constructive interference.

Observables

In quantum mechanics, *observables* are physical quantities that can be measured, such as position, momentum, or energy. Each observable is represented mathematically by a *Hermitian operator* $\hat{O}$ that acts on quantum states in a given Hilbert space. The eigenvalues of this operator correspond to the possible measurement outcomes for that observable, and the eigenvectors represent the states in which these outcomes are certain. In quantum computing, common observables include Pauli operators (X, Y, Z), which can represent spin or polarization measurements, and Hamiltonians, which represent the total energy of a system.

For example, if $\hat{O}$ is the Pauli-Z operator, measuring a qubit in the state $|0\rangle$ yields a result of +1, while measuring it in $|1\rangle$ yields −1. Observables are central to quantum computation because they allow us to extract information from quantum states, typically at the end of a quantum algorithm. However, measurement collapses the quantum state to an eigenstate of the observable, so the outcome of each measurement only reveals partial information about the system’s prior superposition.

Expectation Value

The **expectation value** of an observable is the average outcome of repeated measurements of that observable on a quantum system in a given state. For a quantum state $|\psi\rangle$ and an observable $\hat{O}$, the expectation value $\langle\hat{O}\rangle$ is defined as:

$$\langle\hat{O}\rangle = \langle\psi|\hat{O}|\psi\rangle.$$

This value represents the “mean” result you would obtain if you measured the observable $\hat{O}$ many times on identically prepared systems in the state $|\psi\rangle$.

In quantum computing, expectation values are often calculated rather than measured directly, especially in algorithms like the *Variational Quantum Eigensolver* (VQE), where the expectation value of the Hamiltonian represents the energy of a molecular system. Since the expectation value of an observable can reveal important information about the system without fully collapsing its state, it is a critical tool for extracting data from

quantum computations, particularly in applications involving optimization, physics, and chemistry.

Interference

Interference is a fundamental concept in quantum computing that enables quantum algorithms to amplify correct solutions while diminishing incorrect ones. In classical probability, different outcomes are simply summed up to give the total probability, but in quantum mechanics, probabilities are derived from complex amplitudes, which can interfere constructively or destructively.

When quantum states interfere constructively, their amplitude probabilities combine to increase the likelihood of certain outcomes. This is the basis for quantum algorithms like Grover's search, where constructive interference amplifies the probability of correct answers.

Destructive interference occurs when amplitudes cancel each other out, reducing the probability of certain outcomes. Quantum algorithms can use destructive interference to suppress incorrect solutions, steering the quantum system toward states that represent correct answers.

Quantum algorithms leverage interference by carefully adjusting phase rotations and entangling operations, creating patterns of constructive and destructive interference across the quantum state space. By shaping these interference patterns, quantum computers can explore complex solution spaces more efficiently than classical systems, amplifying paths that lead to optimal solutions while reducing others. Through interference, quantum computing gains the ability to perform computations in parallel, reinforcing solutions that satisfy problem constraints and canceling out solutions that do not. This mechanism is key to the efficiency and power of many quantum algorithms, making interference one of the defining principles that distinguishes quantum computing from classical computation.

Quantum Gates

In quantum computing, quantum gates are the basic building blocks for manipulating qubits, similar to how logic gates function in classical computing. However, unlike classical gates that operate deterministically, quantum gates are unitary operations (*i.e.*, preserve the inner product) that act on quantum states in a reversible manner, preserving the total probability. These gates enable the unique transformations necessary for quantum algorithms by exploiting quantum phenomena like superposition, entanglement, and interference.

Single-qubit gates operate on individual qubits and include:

- **Pauli Gates:** The Pauli-X, Y, and Z gates perform specific rotations on the Bloch sphere. For example, the **Pauli-X gate** acts like a quantum “NOT”, flipping the qubit state from $|0\rangle$ to $|1\rangle$ and vice versa.
- **Hadamard Gate (H):** This gate creates superposition by transforming a qubit from a definite state (*e.g.*, $|0\rangle$) into an equal superposition of $|0\rangle$ and $|1\rangle$.
- **Phase Gates:** Gates like $R_Z(\theta)$ rotate the qubit’s phase around the Z-axis, adding a controllable phase to the qubit state, which is essential for interference.

Multi-qubit gates operate on two or more qubits and are essential for creating entanglement, which links the states of multiple qubits. Key multi-qubit gates include:

- **Controlled NOT (CNOT):** This two-qubit gate flips the target qubit’s state if the control qubit is $|1\rangle$. The CNOT gate is foundational for entanglement and is commonly used in creating entangled pairs in quantum circuits.
- **Controlled Phase Gates:** These apply a phase shift to a target qubit depending on the state of a control qubit, allowing for intricate interference patterns in algorithms.
- **SWAP Gate:** The SWAP gate exchanges the states of two qubits and is often used in algorithms to rearrange qubit states or in error correction.

Quantum Circuits

Quantum circuits are sequences of quantum gates arranged to perform specific computations. A quantum circuit begins with qubits in a well-defined initial state (often $|0\rangle$), and a sequence of gates is applied to transform these qubits into a superposition or entangled state that encodes the solution to a problem.

Quantum circuits can be organized into **layers**, where each layer consists of parallel gates that act on different qubits simultaneously. Layered structures are common in quantum algorithms, as they allow for efficient implementation of complex transformations.

At the end of a quantum circuit, **measurements** collapse the qubits' superposition into classical bitstrings, providing output data. The probability distribution of these outputs reveals information about the computation.

Quantum gates and circuits enable the core operations of **quantum algorithms**. By applying quantum gates in a structured sequence, quantum circuits execute algorithms designed to solve specific problems. By carefully arranging gates and selecting parameters, quantum circuits harness superposition, entanglement, and interference to solve specific problems. Quantum circuits are designed to emphasize constructive interference for solutions that satisfy the problem's constraints and to use destructive interference to reduce incorrect solutions.

Variational Method in Quantum Mechanics

The *variational method* is a powerful technique for approximating the ground state energy of a quantum system. The approach relies on the *variational principle*, which states that for any normalized trial wave function ψ , the expectation value of the Hamiltonian H calculated using ψ will always be greater than or equal to the true ground state energy E_0 , *i.e.*:

$$\langle \psi | H | \psi \rangle \geq E_0.$$

The variational method proceeds as follows:

1. *Trial Wave Function*: choose a trial wave function ψ_{trial} that depends on one or more adjustable parameters.
2. *Energy Minimization*: calculate the expectation value of the Hamiltonian, *i.e.*:

$$\langle \psi_{\text{trial}} | H | \psi_{\text{trial}} \rangle,$$

and adjust the parameters to minimize this value.

3. *Approximate Solution*: the minimized expectation value provides an approximation of the ground state energy, while the optimized trial wave function approximates the ground state wave function.

The variational method is widely applied in quantum mechanics, particularly for complex systems where finding the exact ground state solution is computationally prohibitive.

5.2 Quantum Optimization

When approaching optimization problems from a quantum perspective, the initial task is to translate the original problem into quantum circuits and operators. This process involves the following key steps:

1. *Mathematical Reformulation*: start by expressing the original problem using the *Quadratic Unconstrained Binary Optimization* (QUBO) or Ising notation. This representation facilitates the mapping of the problem into a format suitable for quantum computation.
2. *Hamiltonian Representation*: reformulate the optimization problem as a Hamiltonian, where the ground state of this Hamiltonian corresponds to the solution that minimizes the cost function.

Remark 2 *Note that the objective function serves as a mathematical representation of the system's energy. Our ultimate goal is to minimize this energy; a lower energy state corresponds to a better solution. Therefore, we will frame every optimization problem as a minimization problem, a transformation that is always mathematically feasible:*

$$\max_{x \in \mathcal{X}} f(x) \quad \Longleftrightarrow \quad -\min_{x \in \mathcal{X}} -f(x).$$

5.2.1 Mathematical Reformulation: Objective Function

The objective functions can be expressed in two equivalent forms: *Ising Hamiltonians* and *QUBO formulations*.

The Ising Model is a well-known framework in statistical mechanics used to describe interactions in a system (Lucas [113]). An Ising Hamiltonian involves variables that take values of $+1$ or -1 . Mathematically, an Ising Hamiltonian is expressed as:

$$\min_{s \in \{-1, 1\}^n} \sum_{i=1}^n h_i s_i + \sum_{1 \leq i < j \leq n} J_{ij} s_i s_j \quad (5.6)$$

where:

- s_i and s_j are the variables taking values in $\{+1, -1\}$.
- The coefficients h_i are the bias terms,
- The coefficients J_{ij} are the interaction (or coupling) constants between i and j , capturing their coupling strength.

The first term, $\sum_{i=1}^n h_i s_i$ represents the interaction with an external field. The second term, $\sum_{i=1}^n \sum_{j=i+1}^n J_{ij} s_i s_j$, represents the pairwise interactions between i and j .

Quadratic Unconstrained Binary Optimization (QUBO) problems are well-known in the field of combinatorial optimization .

A QUBO problem is defined by a $n \times n$ upper triangular matrix Q and a vector of n binary variables x . The mathematical form of the QUBO problem is given by:

$$\min_{x \in \{0,1\}^n} \sum_{i=1}^n Q_{ii} x_i + \sum_{1 \leq i < j \leq n} Q_{ij} x_i x_j \iff \min_{x \in \{0,1\}^n} x^\top Q x. \quad (5.7)$$

In this formulation:

- The diagonal elements Q_{ii} of the matrix Q are the linear coefficients, representing the contribution of each binary variable x_i individually.
- The off-diagonal elements Q_{ij} (where $i \neq j$) are the quadratic coefficients, representing the interaction between binary variables x_i and x_j .

Remark 3 Any QUBO problem can be easily mapped into an Ising problem through simple equivalence:

$$s_i \mapsto 2x_i - 1 \quad \text{and} \quad x_i \mapsto \frac{s_i + 1}{2}.$$

Remark 4 Any mathematical problem can be mapped into a QUBO problem.

5.2.2 Mathematical Reformulation: Constraints

By definition, a QUBO problem does not inherently accommodate constraints. However, there is a method to incorporate them. In this section, we will explore how to implement a linear constraint within a QUBO formulation.

The key lies in the concept of a *penalty function*, which is an additional term added to the original objective function. This penalty must be minimized along with the original objective, ensuring that the entire problem is optimized while satisfying the constraint.

Suppose we want to add the following constraint to our optimization problem:

$$\sum_{i=1}^n x_i = \varphi,$$

where φ is a given constant. We can incorporate this constraint into the objective function by minimizing the square of the difference between the sum and φ :

$$\sum_{i=1}^n x_i = \varphi \quad \Rightarrow \quad \min \left(\sum_{i=1}^n x_i - \varphi \right)^2.$$

Expanding the square yields:

$$\left(\sum_{i=1}^n x_i - \varphi \right)^2 = \left(\sum_{i=1}^n x_i \right)^2 + \varphi^2 - 2\varphi \sum_{i=1}^n x_i.$$

Since φ^2 is a constant, it can be omitted from the objective function without affecting the solution. Thus, we have:

$$\left(\sum_{i=1}^n x_i \right)^2 - 2\varphi \sum_{i=1}^n x_i = \sum_{i=1}^n x_i^2 + 2 \sum_{1 \leq i < j \leq n} x_i x_j - 2\varphi \sum_{i=1}^n x_i.$$

Noting that $x_i^2 = x_i$ for binary variables $x_i \in \{0, 1\}$, this simplifies to:

$$\sum_{i=1}^n x_i - 2 \sum_{1 \leq i < j \leq n} x_i x_j - 2\varphi \sum_{i=1}^n x_i.$$

We can further combine terms to get:

$$(1 - 2\varphi) \sum_{i=1}^n x_i - 2 \sum_{1 \leq i < j \leq n} x_i x_j.$$

Thus, the new objective function that includes the equality constraint becomes:

$$\min_{x \in \{0,1\}^n} \sum_{i=1}^n Q_{ii} x_i + \sum_{1 \leq i < j \leq n} Q_{ij} x_i x_j + \beta \left[(1 - 2\varphi) \sum_{i=1}^n x_i - 2 \sum_{1 \leq i < j \leq n} x_i x_j \right],$$

where β is a penalty coefficient that controls the strength of the constraint relative to the original objective function.

Remark 5 *The penalty coefficient β plays a critical role in ensuring that the equality constraint is respected when solving the optimization problem.*

- *Large β : the optimization process will heavily penalize any solution that violates the constraint. This means that the solver will prioritize finding solutions that satisfy the constraint even if it results in a slightly worse original objective function value.*

- *Small β : the solver may not penalize constraint violations strongly enough. As a result, the optimization process might return a solution that has a lower original objective value but violates the constraint.*

In other words, β controls the trade-off between minimizing the original objective function and satisfying the constraint. Typically, β is chosen through experimentation or based on prior knowledge of the problem. We start with a moderate β value that is comparable to the magnitude of the original objective function terms. Then, if the solution violates the constraint, we increase β incrementally until the constraint is satisfied. Conversely, if the constraint is satisfied but the original objective function value is poor, we consider reducing β slightly.

Suppose we want to add the following constraint to our optimization problem:

$$\sum_{i=1}^n x_i \leq \vartheta,$$

where ζ is a given constant. How to deal with it? One way can be to re-state this inequality into an equality using ϑ new *slack binary variables*:

$$\sum_{i=1}^n x_i + \sum_{i=1}^{\vartheta} x_i = \zeta \quad \Longleftrightarrow \quad \sum_{i=1}^{n+\vartheta} x_i = \zeta.$$

At this point one can use the procedure described for equality constraints.

5.2.3 Hamiltonian Representation

In transitioning from a classical optimization problem to its quantum formulation, the first step is to convert classical variables into quantum operators. Specifically, we map the classical (binary or $-1, +1$) variables, typically denoted as x_i with $i \in \{1, \dots, n\}$, to quantum mechanical operators—more precisely, to Pauli Z matrices. These Pauli Z matrices, denoted by Z_i with $i \in \{1, \dots, n\}$, operate on qubits and take the eigenvalues ± 1 , representing the states of the quantum system. This transformation is essential for representing classical decision variables in a quantum mechanical framework, as quantum systems inherently work with operators instead of static variables.

Definition 8 *The Pauli Z gate is a single-qubit rotation through π radians around the z -axis.*

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

When we substitute these Pauli Z matrices into the classical optimization problem, we obtain a quantum mechanical object known as the Hamiltonian, denoted H_C . This Hamiltonian encodes the cost function of the original problem, and is referred to as the *cost function Hamiltonian*. The cost function hence Hamiltonian takes the following form:

$$H_C = f(Z_1, Z_2, \dots, Z_n), \quad (5.8)$$

where f is the original cost function reformulated in terms of the Pauli operators.

The key property of (5.8) is that its ground state (the lowest energy state of the system) corresponds to the solution that minimizes the original cost function (3.1). In other words, the quantum states with the lowest energy levels represent the optimal or near-optimal solutions to the problem. Thus, to solve the problem on a quantum computer, the goal is to evolve the quantum system into a state that either reaches or has a high overlap with the ground state of H_C .

Techniques like *Quantum Annealing* (QA) or the *Quantum Approximate Optimization Algorithm* (QAOA) are employed to drive the system towards this ground state. For approaches like the QAOA, a parameterized quantum circuit is constructed to approximate the ground state of the Hamiltonian, using alternating layers of mixing and cost Hamiltonians. In contrast, QA directly evolves the system under a time-dependent Hamiltonian to reach the ground state. Once prepared, we can measure the quantum system and sample from this low-energy state, which, with high probability, will yield the optimal or near-optimal solution to the original optimization problem. See Hadfield [71], Abbas et al. [4], Blekos et al. [22], Sachdeva et al. [140], and references therein for a deeper investigation on quantum algorithms for optimization problems.

5.3 Quantum Annealing

Once the Hamiltonian representation for the optimization problem is formulated, the next steps in QA involve setting up the system's evolution and guiding it to reach the ground state of this Hamiltonian, corresponding to the optimal solution. The process can be broken down into the following steps:

Step 0: Define the Initial Hamiltonian H_0

The process begins by selecting a simple Hamiltonian, often called the *initial Hamiltonian*, H_0 . A common choice for H_0 is:

$$H_0 = - \sum_i X_i,$$

where X_i is the Pauli X operator, which creates a superposition of all possible states.

Step 1: Set the Time-Dependent Hamiltonian $H(t)$

QA operates by gradually transitioning from the initial Hamiltonian H_0 to the *target Hamiltonian*, which encodes the optimization objective; for all $t \in \{0, \dots, T\}$ we have:

$$H(t) = [1 - s(t)] \cdot H_0 + s(t) \cdot H_C.$$

The function $s(t)$ defines the *annealing schedule*, controlling the transition from H_0 to H_C over the total annealing time T . At $t = 0$ we have $s(0) = 0$ so that $H(0) = H_0$; at $t = T$, we have $s(T) = 1$ so that $H(T) = H_C$.

Step 2: Choose an Annealing Schedule

The annealing schedule $s(t)$ plays a critical role in the process. A linear schedule, where $s(t)$ changes at a constant rate, is commonly used, but an adaptive or non-linear schedule can sometimes improve performance by allowing more time for complex parts of the problem. The goal is to evolve the system slowly enough for it to remain in its *ground state*. This principle, known as the *adiabatic theorem*, ensures that if the system starts in the ground state of H_0 , it will end in the ground state of H_C , yielding the optimal solution. Note that longer annealing times increase the chance of ending in the ground state, though at the cost of computation time. A common linear schedule is:

$$s(t) = \frac{t}{T}.$$

With this schedule:

- At the start ($t = 0$), $s(0) = \frac{0}{T} = 0$, so the Hamiltonian is $H(0) = H_0$.
- At the end ($t = T$), $s(T) = \frac{T}{T} = 1$, so the Hamiltonian is $H(T) = H_C$.

The Hamiltonian at any time t during the annealing process can then be expressed as:

$$H(t) = [1 - s(t)] \cdot H_0 + s(t) \cdot H_C = \left(1 - \frac{t}{T}\right) \cdot H_0 + \frac{t}{T} \cdot H_C.$$

Step 3: Measurement and Solution Extraction

After the annealing process, measurements are made on the final state of the system to determine the final configurations (*i.e.*, the values of the variables) that represent the solution. Since quantum systems are probabilistic, multiple runs are often required to ensure that the observed state is indeed the ground state.

5.4 Quantum Approximate Optimization Algorithm (QAOA)

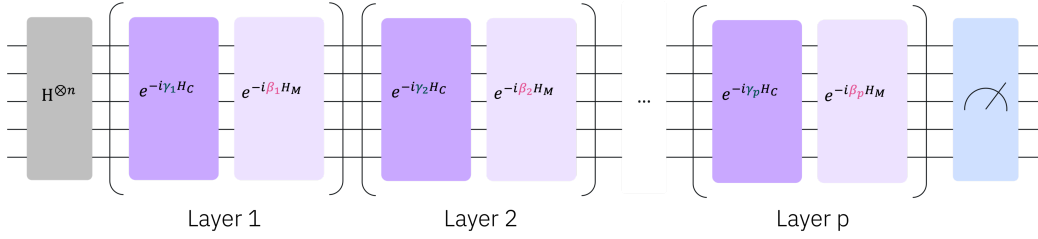


Figure 5.1: QAOA circuit diagram. Taken and adapted from the Qiskit documentation

Combinatorial optimization problems frequently require identifying an n -bit string that maximizes a given objective function, denoted as $C(z)$, where $z = z_1 z_2 \dots z_n$ represents the bit string. Typically, C is composed of multiple clauses, each contributing to the overall objective based on specific subsets of bits. Formally, we define:

$$C(z) = \sum_{\alpha=1}^m C_{\alpha}(z),$$

where $C_{\alpha}(z) = 1$ if z satisfies clause α , and 0 otherwise.

The *Quantum Approximate Optimization Algorithm (QAOA)* is a hybrid quantum-classical algorithm developed to approximate solutions to such problems using quantum resources. Working within a 2^n -dimensional Hilbert space spanned by computational basis states $|z\rangle$, QAOA leverages quantum phase and mixing operations to concentrate the probability distribution around solutions that maximize (or minimize) the objective function. It is particularly suitable for problems expressed in Quadratic Unconstrained Binary Optimization (QUBO) form, including $\mathcal{NP}$ -complete cases like MaxCut and Max E3LIN2.

At the heart of QAOA are two alternating unitary operators that depend on tunable parameters: the **Phase Operator** $U(C, \gamma)$ and the **Mixing Operator** $U(B, \beta)$. These operators are defined as:

$$U(C, \gamma) = e^{-i\gamma C} = \prod_{\alpha=1}^m e^{-i\gamma C_{\alpha}},$$

where $\gamma \in [0, 2\pi]$, and

$$U(B, \beta) = e^{-i\beta B} = \prod_{j=1}^n e^{-i\beta \sigma_j^x},$$

with $\beta \in [0, \pi]$.

Here, $B = \sum_{j=1}^n \sigma_j^x$, where σ_j^x is the Pauli-X operator applied to bit j , which serves to explore different configurations. See Farhi et al. [52].

5.4.1 Key Algorithmic Steps

- **Initialization in Superposition** – QAOA begins by preparing all qubits in an equal superposition state:

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_z |z\rangle,$$

using Hadamard gates to ensure each bitstring has equal probability. This superposition makes all potential solutions available for subsequent transformations.

- **Cost Hamiltonian** – The cost function C of the optimization problem is encoded into a Hamiltonian, known as the **Cost Hamiltonian** H_C . Each configuration of the system corresponds to a quantum state whose energy is defined by H_C . For example, in a MaxCut problem, where qubits represent graph nodes and edges indicate interactions, H_C could be formulated as:

$$H_C = \sum_{\langle i,j \rangle} Z_i Z_j,$$

where Z_i is the Pauli-Z operator on qubit i , and $\langle i, j \rangle$ represents an edge between nodes i and j .

- **Phase Operator** – The **Phase Operator** $U(C, \gamma) = e^{-i\gamma H_C}$ introduces phase shifts according to the values of the cost function, effectively amplifying states with lower costs. By adjusting the parameter γ , the algorithm emphasizes configurations with favorable values,

promoting constructive interference for low-cost states while creating destructive interference for high-cost configurations.

- **Mixing Hamiltonian** – To prevent the algorithm from becoming trapped in local minima, the **Mixing Hamiltonian** H_B is employed to encourage transitions between states:

$$H_B = \sum_{i=1}^n X_i,$$

where X_i is the Pauli-X operator on qubit i . The mixing Hamiltonian facilitates exploration by “flipping” qubit states, thus expanding the search across different configurations.

- **Mixing Operator** – The **Mixing Operator** $U(B, \beta) = e^{-i\beta H_B}$, implemented as rotations around the X-axis by an angle β , acts as:

$$U(B, \beta) = R_X(\beta)^{\otimes n},$$

where each qubit is rotated independently. This operator encourages movement within the solution space, balancing exploration with the selective reinforcement from the Phase Operator.

- **Constructing the QAOA Circuit** – The QAOA circuit consists of applying p layers of alternating phase and mixing operations to the initial state $|s\rangle$, forming a parameter-dependent state $|\gamma, \beta\rangle$:

$$|\gamma, \beta\rangle = U(B, \beta_p)U(C, \gamma_p) \cdots U(B, \beta_1)U(C, \gamma_1) |s\rangle.$$

The expectation value of C with respect to this state is given by:

$$F_p(\gamma, \beta) = \langle \gamma, \beta | C | \gamma, \beta \rangle.$$

Maximizing $F_p(\gamma, \beta)$ over the parameters (γ, β) yields an approximation to the optimal solution. Importantly, as p increases, the quality of this approximation improves, approaching the true maximum of $C(z)$.

- **Parameter Optimization** – QAOA is a hybrid algorithm, utilizing classical optimization to iteratively refine the parameters γ and β to maximize the probability of measuring the optimal or near-optimal solution. Through classical optimization routines, these parameters are fine-tuned over multiple executions, gradually improving the solution quality with each iteration.

- **Sampling and Measurement** – After executing the QAOA circuit with the optimal parameters (γ, β) , the final quantum state is measured multiple times. The most frequently observed configurations—those that correspond to low-cost solutions—represent the algorithm’s best approximation of the optimal solution.
- **Iterative Improvement** – By iterating the classical optimization of (γ, β) , QAOA dynamically enhances the distribution of measurement outcomes, focusing on configurations that yield minimal cost.

Recent improvements to QAOA, such as Multi-Angle QAOA and Digitized Counterdiabatic QAOA, aim to increase the algorithm’s convergence speed and resilience to noise, particularly beneficial in the context of *Noisy Intermediate-Scale Quantum* (NISQ) devices. These variations introduce additional angles or customized ansätze that can yield better approximation ratios, enabling more efficient optimization over complex problem instances (Blekos et al. [22]).

In particular, the multi-angle QAOA (ma-QAOA) introduces separate angles for each term within these operators. Specifically, while the standard QAOA applies cost and mixing operators as:

$$U(\gamma, C) = e^{-i\gamma C} \quad \text{and} \quad U(\beta, B) = e^{-i\beta B},$$

where γ and β are real-valued angles shared across all terms, ma-QAOA replaces these with:

$$U(\vec{\gamma}_l, C) = e^{-i\sum_a \gamma_{l,a} C_a} \quad \text{and} \quad U(\vec{\beta}_l, B) = e^{-i\sum_v \beta_{l,v} B_v},$$

where each layer l has its own set of parameters $\vec{\gamma}_l$ and $\vec{\beta}_l$, allowing independent control over each clause a in the cost operator and each qubit v in the mixing operator. This finer granularity in ma-QAOA yields a higher approximation ratio by enabling more precise control over the state evolution, achieving improved performance in fewer layers. For example, empirical results show that for MaxCut problems, a single ma-QAOA iteration matches the approximation quality of three QAOA layers, suggesting that ma-QAOA is particularly advantageous for near-term quantum devices where circuit depth and noise are limiting factors (Herrman et al. [79]).

On the other hand, the *Digitized-Counterdiabatic QAOA* (DC-QAOA) enhances the conventional QAOA by integrating counterdiabatic (CD) driving terms to the problem Hamiltonian, reducing nonadiabatic transitions and accelerating convergence. Standard QAOA evolution from the mixing to cost Hamiltonian relies on:

$$U(\gamma, \beta) = e^{-i\beta B} e^{-i\gamma C},$$

where β and γ are layer-specific parameters for the mixer B and cost C operators. In DC-QAOA, we augment this with an additional CD driving unitary:

$$U(\gamma, \beta, \alpha) = e^{-i\alpha A} e^{-i\beta B} e^{-i\gamma C},$$

where A represents the CD operator selected based on the problem Hamiltonian's structure. By introducing the variational parameter α for the CD term, DC-QAOA improves the approximation ratio, especially at shallow depths. For instance, in certain cases like Ising models, DC-QAOA achieves optimal approximation with just $p = 1$, compared to higher depths needed in QAOA. The cost function $F(\gamma, \beta, \alpha) = \langle \psi(\gamma, \beta, \alpha) | H | \psi(\gamma, \beta, \alpha) \rangle$, optimized with respect to γ , β , and α , results in faster convergence to the ground state while reducing circuit depth. Thus, DC-QAOA's structured inclusion of CD terms makes it more efficient on near-term quantum devices for combinatorial and ground-state problems. See Chandarana et al. [31].

The QAOA offers a versatile framework for solving combinatorial optimization problems on quantum devices, with the potential for scalable, high-quality solutions as p grows. Through the interplay of quantum interference and classical optimization, QAOA marks a significant step toward addressing computationally challenging problems in the quantum era.

5.5 Quantum Machine Learning

Quantum machine learning harnesses the intrinsic parallel processing capabilities of quantum computing, elevating the performance of traditional machine learning methods. Its core principle involves translating information from classical learning algorithms into quantum states and conducting unitary evolutions on these states for computational purposes. Fundamentally, quantum machine learning unites the realms of quantum computing and traditional machine learning. By categorising data sources as either classical (C) or quantum (Q) and distinguishing the computational device for data processing as classical (C) or quantum (Q), the fusion of machine learning and quantum computing can be approached through four distinctive strategies (Jadhav et al. [86]), as depicted in Figure 5.2.

- **Classic-classic (CC) method:** This is the classical computing approach, which relies on traditional hardware and algorithms for processing information. In the CC scenario, data is handled using conventional machine learning techniques.
- **Classic-quantum (QC) method:** This scenario examines how machine learning might aid quantum computing. For instance, by obtain-

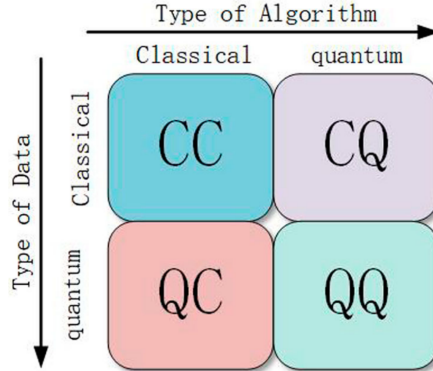


Figure 5.2: Four distinct approaches to combining quantum computing and machine learning. See Chen et al. [33]

ing a complete description of the state of a computing device from a few measurements and analyzing the measurement data using ML. See Aaronson [3]

- **Classic-quantum (CQ) method:** This method leverages quantum computing to analyze classical datasets, such as images and text, via quantum algorithms. The primary goal is to develop algorithms by either adapting classical machine learning models or creating new ones suited to quantum computing’s unique properties, such as parallelism, superposition, and entanglement (Abohashima et al. [8]). This method combines quantum and classical approaches to accelerate machine learning algorithms, achieving higher performance.
- **Quantum-quantum (QQ) method:** The last scenario examines the processing of “quantum data” by a quantum device. There are two possible interpretations of this. The information may be obtained through a quantum experiment involving a quantum system and the subsequent input of those experimental measurements into a different quantum computer. In a second interpretation, a quantum computer simulates the behavior of the quantum system and then uses the state of this system as such an input of a QML algorithm performed on the same computer.

In our study, we concentrate on the CQ scenario. There are two distinct approaches to the development of quantum machine learning models. The first approach involves running traditional machine learning algorithms on quantum computers or simulators in an effort to achieve algorithmic

speedups. This approach needs to translate conventional data into quantum data, a process known as quantum encoding. Another approach is to create QML algorithms based on quantum subroutines, including Grover’s algorithm, HHL algorithm, quantum phase estimation, and *Variational Quantum Circuit* (VQC), which we cover later in detail. Figure 5.3 illustrates the processing strategies used in conventional machine learning and QML. In traditional machine learning, data are a direct input to the algorithm, which then analyses the data and generates an output. QML, on the other hand, demands the initial encoding of the data into quantum data. QML receives quantum data as input, processes it, and generates quantum data as output. The quantum data are then converted to conventional data.

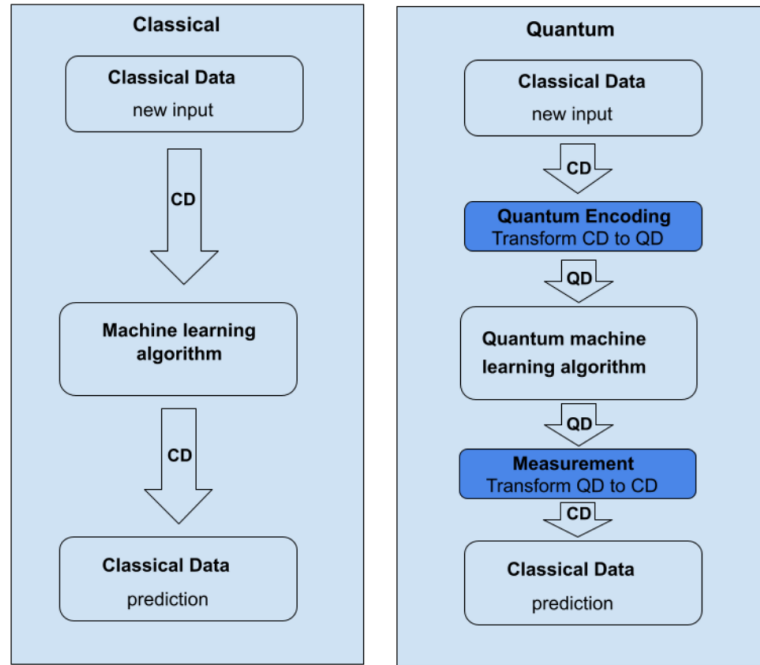


Figure 5.3: Processing techniques of conventional machine learning and quantum machine learning. CD represents classical data and QD represents quantum data. See Zegundry et al. [171]

5.5.1 Quantum Encoding

The majority of ML models depends on sets of data with $\{x_i\}_i \in [N]$ samples, where $x_i \in \mathbb{R}^d$ for every i . Additionally, one requires classical outcomes including labels or classes $\{y_i\}_i \in [N]$, where $y_i \in \mathbb{R}^{d'}$ for every i . In a Hilbert space, there are several strategies for encoding data from classical data to

quantum data. In other words, data encoding requires putting conventional data into a quantum computer as quantum states. We describe four methods. See Ranga et al. [132].

Basis Encoding Basis encoding links a computational basis state for an n -qubit system. Therefore, traditional data should be the binary strings, and encoded quantum state is the bit-wise conversion of a binary string towards the equivalent states of the quantum subsystems. In example, $x = 1001$ is encoded by the 4-qubit quantum state $|1001\rangle$. Furthermore, one bit of classical data are encoded by one quantum subsystem. In some ways, it is the most basic type of computing since every bit is effectively replaced with a qubit, and computations are carried out simultaneously on all bits in a superposition. Consequently, the bit encoding for a vector is the following: $m + \lceil \log(d) \rceil$ qubits could a vector $x = (x_1, \dots, x_d) \in \mathbb{R}^d$ as a quantum superposition of bit strings, where each instance must be a binary string with N bits for the basis encoding. $x_i = (b_1, \dots, b_j, \dots, b_N)$ for $j = 1, \dots, N$ with $b_j \in \{0, 1\}$:

$$|x\rangle = \frac{1}{\sqrt{d}} \sum_{i=1}^d |x_i\rangle, \quad (5.9)$$

where m denotes the amount of qubits used for the precision, while d represents the number of samples. To encode a vector $x = (2, 3)$ in basis encoding. First, we must transform it to a binary format, with 2 bits, $x_1 = 10$, $x_2 = 11$. The corresponding basis encoding makes use of two qubits to represent the data as below:

$$|x\rangle = \frac{1}{\sqrt{2}} |10\rangle + \frac{1}{\sqrt{2}} |11\rangle. \quad (5.10)$$

Amplitude Encoding To employ qubits to represent classical vectors or matrices, the most efficient encoding is by far the amplitude encoding approach. It is the theoretical link between quantum computing and linear algebra that uses quantum features to the maximum. The amplitudes of an n -qubit quantum state $|\psi_x\rangle$ as $|\psi_x\rangle = \sum_{i=1}^d x_i |i\rangle$, with $d = 2^n$ represent a normalized classical d -dimensional set of data x , where x_i is the i -th item, and $|i\rangle$ is the i -th computational base state. However, in this case, x_i might be any numeric value, such as a floating or integer point. For instance, suppose we want to use amplitude encoding to encode the four-dimensional array $x = (1.0, 3.0, -2.5, 0.0)$. The first step is to normalize this array, which is achieved by setting $x_{norm} = \frac{1}{\sqrt{16.25}}(1.0, 3.0, -2.5, 0.0)$. Two qubits are used to encode

xnorm in the corresponding amplitude encoding as follows:

$$|\psi_{x_{norm}}\rangle = \frac{1}{\sqrt{16, 25}} [|00\rangle + 3|01\rangle - 2.5|10\rangle]. \quad (5.11)$$

This result state will at the same moment encode the matrix below:

$$X = \frac{1}{\sqrt{16, 25}} \begin{pmatrix} 1.0 & 3.0 \\ -2.5 & 0.0 \end{pmatrix} \quad (5.12)$$

where the first qubit represents the index for such a row, while the second qubit represents the index for such a column.

Qsample Encoding Qsample encoding is a hybrid scenario of amplitude and basis encoding, and it combines a real amplitude vector with conventional binary probability distributions. Suppose an amplitude vector $v = (v_1, \dots, v_{2_n})^\top$ that describes an n-qubit quantum state. Sampling a series of bits of size n with a set of discrete probabilities $p_1 = |v_1|^2, \dots, p_{2_n} = |v_{2_n}|^2$ is comparable to measuring the n qubits. This means that the amplitude vector may be used to characterize the distribution of an a classical discrete random variable. Taking into account a discrete classical probability distribution over binary representation $p_1, \dots, p_{2_n}$, the quantum state is:

$$|\phi\rangle = \sum_{i=1}^{2^n} \sqrt{p_i} |i\rangle. \quad (5.13)$$

Angle Encoding Angle encoding, also referred to as qubit encoding or tensor product encoding, is a technique employed by various QML algorithms to encode input data features into the rotation angles of qubits. For a feature vector $x = [x_1, \dots, x_d]$ belonging to the space X^d , angle encoding is mathematically represented as:

$$|x\rangle = \bigotimes_{i=1}^d (\cos(x_i) |0\rangle + \sin(x_i) |1\rangle) \quad (5.14)$$

where d features are encoded into an n-qubit system. The unitary operation for state preparation of a single qubit in qubit encoding is given by:

$$S_{x_j} = \bigotimes_{i=1}^d U_i \quad (5.15)$$

with each U_i defined as:

$$U_i = \begin{pmatrix} \cos \left(x_j^{(i)} \right) & -\sin \left(x_j^{(i)} \right) \\ \sin \left(x_j^{(i)} \right) & \cos \left(x_j^{(i)} \right) \end{pmatrix}. \quad (5.16)$$

Dense qubit encoding, a variant of angle encoding, allows for the encoding of two features within a single qubit, thereby enhancing the density of the encoded information. See LaRose and Coyle [108].

The ZZFeatureMap in Qiskit is an example of angle encoding with added entanglement. In angle encoding, classical features are mapped onto qubit phase rotations. For the ZZFeatureMap, each classical feature modulates the rotation angle along the Z-axis of each qubit, followed by a ZZ interaction between pairs of qubits.

This structure uses the ZZ term to introduce entanglement, which is essential for representing complex relationships among data features. By combining feature-based phase rotations with entangling operations, the ZZFeatureMap efficiently captures nonlinear patterns in the data, making it a valuable tool in quantum machine learning.

5.5.2 HHL Algorithm

Many traditional machine learning processes require linear algebra computation of high-dimensional vectors. The mathematics behind quantum mechanics is basically linear algebraic computation in 2^n dimensions, requiring only n qubits. More concretely, calculations can be formulated via a mapping of an N -dimensional vector $|v\rangle$ to a quantum state of $\log_2 N$ qubits of which basis is labeled by q_α :

$$|v\rangle = \frac{1}{\sqrt{\sum_{i=1}^N v_i^2}} \sum_{i=1}^N v_i |i\rangle, \quad \text{where } |i\rangle = |q_1, \dots, q_\alpha\rangle. \quad (5.17)$$

A quantum device achieving this mapping is called *quantum random access memory* (qRAM, Giovannetti et al. [61]). Solving a system of linear equations is comparable to the matrix inversion issue. The objective of the matrix inversion problem, with a matrix M and a vector $|b\rangle$, is to determine the vector $|x\rangle$:

$$M |x\rangle = |b\rangle. \quad (5.18)$$

Numerous algorithms for ML determine their parameter θ through solving the $M\vec{\theta} = \vec{v}$ matrix inversion issue. In general, the matrix M represents the

input characteristics of the training data points expressed in the matrix X . The data point matrix X and the targeted vector $\vec{Y}$ are both used to create the training data point vector $\vec{v}$. In linear regression, in which the predicted output is $\vec{y} = \vec{\theta}^\top \cdot \vec{x}$, determining $\vec{\theta}$ is as simple as solving the matrix inversion issue presented below:

$$(X^\top X)\vec{\theta} = X^\top \vec{Y} \quad (5.19)$$

thus, $M = X^\top X$ and $\vec{v} = X^\top \vec{Y}$ in linear regression.

The *Harrow–Hassidim–Lloyd algorithm* (HHL) algorithm (Harrow et al. [73]) can solve this task in $\mathcal{O}((\log N)^2)$ time complexity under certain conditions on X , but any classical algorithm requires time complexity at least proportional to N . Many other linear algebra problems more or less can be reduced to above formulation, for example, PCA and SVM. There remain, however, a few caveats (Najafi et al. [124]): (1) this requires a qRAM; it is still unknown how to realize this efficiently in any quantum computation model; (2) the output is a quantum state; due to the wave function collapse in quantum measurement, only one specific bitstring can be obtained according to the probability which is the absolute square of the corresponding coefficient. The latter has been solved in some application (Kerenidis and Prakash [97]), but the former is still unknown. In fact, if a similar assumption is made for classical computations (namely, encoding a high-dimensional vector with nonnegative entries into a probabilistic distribution), some of these quantum algorithms can be translated into the classical domain, which means the time complexity is also for classical computation. See Tang [154], Najafi et al. [124]. Although there is no known explicit realizations of HHL algorithm at the large scale at the moment, it is still an exciting direction deserves further studies, especially when quantum devices are fast developing with more and more capabilities.

5.5.3 Quantum Support Vector Machine (QSVM)

One of the early works in QSVM was proposed by Rebentrost et al. [134], which introduced a quantum algorithm for SVM classification that provides an exponential speedup over classical algorithms. Over the years, there has been significant research in the field of QSVM, exploring various theoretical and practical aspects of the algorithm. Researchers have developed several techniques to enhance the performance of QSVM, including the development of quantum kernel methods, quantum feature maps, and quantum optimization techniques. This earlier form of the QSVM relies on the HHL algorithm to solve the linear systems associated with SVM optimization, specifically by solving the dual form of the SVM problem in eq. 4.41 and eq. 4.42.

This approach applies quantum computing to speed up the solution of large linear systems, where classical SVMs would traditionally require extensive computational resources. Using the HHL algorithm, the QSVM computes this optimization by solving the linear system:

$$\mathbf{A}\alpha = b \quad (5.20)$$

where $\mathbf{A}$ and b represent matrices derived from the kernel function values and class labels.

Various techniques have been combined to improve the QSVM. Recent studies by Innan et al. [83] explore advancements such as the *quantum kernel support vector machine* (QK-SVM), the *quantum variational support vector machine* (QV-SVM), and the *quantum variational kernel support vector machine* (QVK-SVM), which combines the quantum kernel approach with a quantum variational circuit.

In **QK-SVM**, a **quantum kernel** is used to map data into a high-dimensional feature space without explicitly computing the transformation. The dual form of the SVM remains the same as above, but here, the **quantum kernel** $K(x_i, x_j)$ is calculated by a quantum computer. Specifically, the data points $\mathbf{x}_i$ and $\mathbf{x}_j$, are encoded into quantum states $|\phi(\mathbf{x}_i)\rangle$ and $|\phi(\mathbf{x}_j)\rangle$. The kernel is defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = |\langle \phi(\mathbf{x}_i) | \phi(\mathbf{x}_j) \rangle|^2. \quad (5.21)$$

The steps are.

1. Encode each data point x_i as a quantum state using a quantum feature map $U(x)$, such that $U(x)|0\rangle = |\phi(x)\rangle$.
2. Compute the kernel matrix element $K(x_i, x_j) = |\langle \phi(x_i) | \phi(x_j) \rangle|^2$ using a quantum circuit.

This kernel matrix is then used in a **classical SVM** to perform classification. QK-SVM does not involve training quantum parameters, only computing inner products with quantum circuits leveraging the kernel trick to classify data non-linearly using quantum-enhanced features.

In **QV-SVM**, a **parameterized quantum circuit** is used to transform input data into quantum states, learning an effective feature map during training. The parameters θ of a variational quantum circuit $U(\mathbf{x}; \theta)$, which maps each data point x to a quantum state, are trained:

$$|\psi(\mathbf{x}; \theta)\rangle = U(\mathbf{x}; \theta)|0\rangle. \quad (5.22)$$

To classify, the expectation of a specific observable O is measured as follows:

$$f(\mathbf{x}; \theta) = \langle \psi(x; \theta) | O | \psi(x; \theta) \rangle. \quad (5.23)$$

The objective is to optimize θ to minimize a **hinge loss** (or another loss function) based on the margin between classes:

$$\mathcal{L}(\theta) = \sum_{i=1}^N \max(0, 1 - y_i f(x_i; \theta)), \quad (5.24)$$

where y_i are the class labels. The variational circuit parameters θ are optimized, allowing the feature map to be learned from data. QV-SVM provides flexibility but requires computationally expensive quantum circuit optimization.

In **QVK-SVM**, the **kernel trick** is combined with a **variational quantum circuit**, making the kernel itself parameterized and learnable. The QVK-SVM computes a **parameterized quantum kernel** $K(\mathbf{x}_i, \mathbf{x}_j; \theta)$, where θ represents the tunable parameters of the variational circuit. For data points $\mathbf{x}_i$ and $\mathbf{x}_j$, we encode them as quantum states $|\psi(\mathbf{x}_i; \theta)\rangle$ and $|\psi(\mathbf{x}_j; \theta)\rangle$ using a parameterized circuit $U(\mathbf{x}; \theta)$, then define the kernel as:

$$K(\mathbf{x}_i, \mathbf{x}_j; \theta) = |\langle \psi(\mathbf{x}_i; \theta) | \psi(\mathbf{x}_j; \theta) \rangle|^2 \quad (5.25)$$

where:

- Each data point x_i is mapped to a state $|\psi(x_i; \theta)\rangle$.
- The kernel matrix element $K(x_i, x_j; \theta) = |\langle \psi(x_i; \theta) | \psi(x_j; \theta) \rangle|^2$ is calculated.
- The **parameters** θ of the kernel are optimized along with a classical SVM to maximize classification accuracy.

The QVK-SVM learns the feature map by optimizing the kernel parameters, allowing for adaptive kernel functions. This approach combines the best aspects of kernel-based methods and variational circuits but is computationally demanding.

5.5.4 Variational Quantum Circuit (VQC)

Variational Quantum Circuits (VQCs) are a class of quantum circuits that use a parameterized set of quantum gates whose parameters are optimized variationally to minimize a cost function. These circuits play a crucial role

in near-term quantum computing, especially in applications QML and optimization problems. See Stechly [150].

A **quantum circuit** is a sequence of quantum gates acting on qubits. The gates are applied to manipulate qubits to encode information, process it, and then produce an output. In VQCs, certain parameters, such as the rotation angles of the quantum gates, are not fixed but are instead treated as adjustable parameters, which can be optimized during training.

In the context of quantum machine learning, VQCs serve as a foundational element for creating models such as *Variational Quantum Classifiers* (VQCs). These classifiers are specifically designed to leverage quantum circuits to process and classify data that may be difficult for classical algorithms to handle efficiently.

A typical variational quantum circuit consists of several layers of quantum gates, where each layer contains parameterized gates. The general form of a VQC can be written as:

$$|\psi(\boldsymbol{\theta})\rangle = U(\boldsymbol{\theta}) |\psi_0\rangle \quad (5.26)$$

where:

- $|\psi(\boldsymbol{\theta})\rangle$ is the quantum state after applying the parameterized quantum circuit.
- $|\psi_0\rangle$ is the initial quantum state (usually a simple state like the computational basis state $|0\rangle$).
- $U(\boldsymbol{\theta})$ is a unitary operator that is a function of the variational parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \dots, \theta_n)$.

These parameters are typically associated with the angles of quantum gates such as **rotation gates** (e.g., $R_y(\theta)$, $R_z(\theta)$), which are applied to qubits. The number of layers and the type of gates used are critical factors in determining the expressive power of the circuit.

Once the quantum circuit is prepared, the quantum system is measured to extract classical data. In a typical classification problem, the measurements can be mapped to a binary or multi-class outcome based on the measurement results.

For a variational quantum classifier, the **cost function** measures how well the circuit's output matches the expected outcome. A standard method for this is to use the **mean squared error (MSE)** or a **cross-entropy loss** function, for example, in binary classification:

$$L(\boldsymbol{\theta}) = -(y \log(p(\boldsymbol{\theta})) + (1 - y) \log(1 - p(\boldsymbol{\theta}))) \quad (5.27)$$

where:

- y is the true label (0 or 1).
- $p(\boldsymbol{\theta})$ is the probability of the measurement outcome corresponding to the positive class, which is computed from the quantum circuit.

In the quantum setting, the probability $p(\boldsymbol{\theta})$ is determined by performing measurements on the quantum state and observing the frequency of the desired outcomes, typically in the computational basis.

A **Variational Quantum Classifier (VQC)** is a quantum machine learning algorithm that uses a variational quantum circuit to solve classification tasks. The key steps involved in a VQC are as follows:

1. **Data Encoding:** Classical data must be encoded into quantum states. The most common approach is to use amplitude encoding or basis encoding, where classical data is transformed into quantum states by applying a series of quantum gates (*e.g.*, Hadamard gates, Pauli rotations) to the qubits.
2. **Circuit Design:** A quantum circuit is designed with parameterized gates to perform computations on the encoded data. This quantum circuit can be composed of several layers of gates, each parameterized by variational parameters.
3. **Measurement and Loss Function:** After executing the quantum circuit, measurements are performed on the qubits. The outcome of these measurements is a probability distribution over possible classical states. The classifier then computes the loss function (like cross-entropy loss) based on the predicted probabilities and the true labels.
4. **Optimization:** The parameters $\boldsymbol{\theta}$ of the quantum circuit are optimized using classical optimization techniques (*e.g.*, gradient descent or other variational optimization algorithms) to minimize the loss function. The optimization process is iterated until convergence, where the quantum circuit learns to classify data effectively.

The mathematical formulation of the classifier output for a binary classification task can be written as:

$$p(\boldsymbol{\theta}) = \left| \langle \psi(\boldsymbol{\theta}) | \hat{O} | \psi(\boldsymbol{\theta}) \rangle \right|^2 \quad (5.28)$$

where $\hat{O}$ is an operator that projects onto the measurement outcome of interest (*e.g.*, the $|1\rangle$ state), and $|\psi(\boldsymbol{\theta})\rangle$ is the quantum state generated by the VQC.

The optimization of VQCs is crucial, as it is the process that fine-tunes the parameters θ to minimize the cost function. This is done by using classical optimization algorithms such as (Cerezo et al. [30]):

Gradient Descent Standard gradient-based methods that compute the gradient of the cost function with respect to the circuit parameters. However, due to the nature of quantum circuits, gradients are often approximated using techniques like *parameter-shift rules* or *finite difference methods*.

Adam Optimizer A widely-used adaptive learning rate optimization algorithm that can be applied to VQCs to improve convergence speed and stability.

Gradient-Free Optimization Methods like genetic algorithms or simulated annealing are sometimes used when gradients are difficult to compute or the cost function is highly non-convex.

Applications of VQCs in Quantum Machine Learning

Quantum Data Classification One of the primary applications of VQCs is in the classification of quantum data or classical data encoded into quantum states. The variational quantum classifier has shown promise in handling certain types of data more efficiently than classical algorithms, particularly in problems related to quantum state discrimination or problems involving quantum-enhanced features.

Quantum Chemistry VQCs are also used in the simulation of quantum systems and the prediction of molecular properties. By leveraging the quantum nature of the system, these models are expected to outperform classical models in predicting properties of large molecules and solving optimization problems in chemistry.

Quantum-enhanced Feature Learning VQCs can be used to learn high-dimensional features of classical data through quantum feature maps. This is particularly useful in tasks like clustering, classification, and regression where the data is embedded into a higher-dimensional quantum Hilbert space.

Variational Quantum Algorithms for Optimization VQCs are applied in optimization problems, especially in combinatorial optimization, such as finding the ground state of a Hamiltonian or solving traveling salesman problems. VQCs are particularly appealing for these tasks because

they can leverage quantum parallelism to explore many solutions simultaneously.

5.5.5 Quantum Neural Network (QNN)

Quantum neural networks are a subclass of variational quantum algorithms, comprising of quantum circuits that contain parameterised gate operations (see Abbas et al. [5], Zhao and Wang [173]). The process is a quantum-classical hybrid: the algorithms themselves are executed on a quantum computer, but the optimization process is done classically. *Parametrised Quantum Circuits* (PQC), in particular, are used for the implementation. Infor-

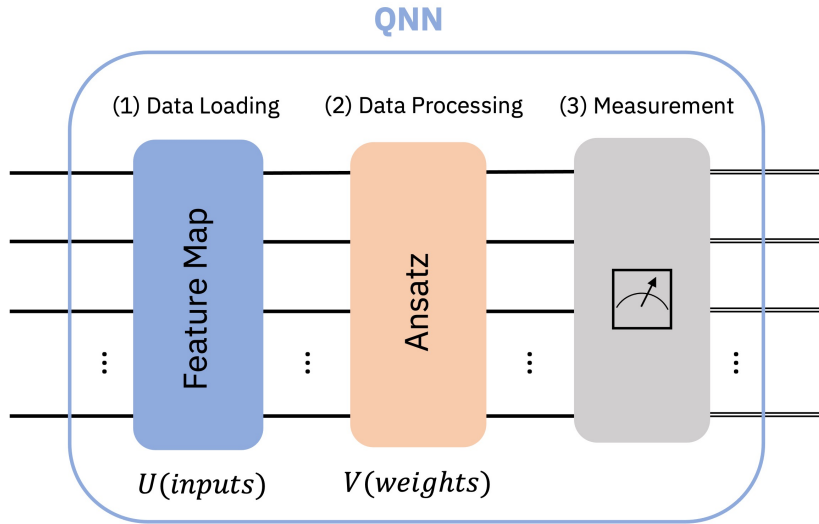


Figure 5.4: QNN circuit diagram. Taken and adapted from Qiskit documentation

mation is first encoded into a quantum state via a state preparation routine or feature map. Classical data $\mathbf{x} \in \mathbb{R}^d$ is first encoded into quantum states $|\psi(\mathbf{x})\rangle$ using an embedding function:

$$|\psi(\mathbf{x})\rangle = U_{\text{enc}}(\mathbf{x})|0\rangle^{\otimes n},$$

where $|0\rangle^{\otimes n}$ is the initial quantum state of n -qubits, typically in the ground state and $U_{\text{enc}}(\mathbf{x})$ is the data encoding unitary operator. After encoding, the circuit applies a parameterized quantum transformation $U(\boldsymbol{\theta})$:

$$|\psi_{\text{out}}\rangle = U(\boldsymbol{\theta})|\psi(\mathbf{x})\rangle,$$

where $\boldsymbol{\theta} = \{\theta_1, \theta_2, \dots, \theta_p\}$ are trainable parameters and $U(\boldsymbol{\theta})$ is a sequence of parameterized quantum gates, such as:

- Rotational gates: $R_x(\theta), R_y(\theta), R_z(\theta)$,
- Entangling gates: Controlled-NOT (CNOT) or Controlled-Z (CZ).

The circuit consists of **multiple layers** of these gates, analogous to layers in classical neural networks:

$$U(\boldsymbol{\theta}) = \prod_{l=1}^L U_l(\boldsymbol{\theta}_l),$$

where L is the number of layers.

The quantum state $|\psi_{\text{out}}\rangle$ is measured in the computational basis to obtain classical information. A set of measurement operators $\{M_i\}$ is applied to extract expectation values:

$$\langle M_i \rangle = \langle \psi_{\text{out}} | M_i | \psi_{\text{out}} \rangle.$$

The outputs are mapped to predictions for tasks like classification or regression. The training objective is to minimize a cost function $C(\boldsymbol{\theta})$, defined as:

$$C(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y_i, f(\mathbf{x}_i, \boldsymbol{\theta})),$$

where $\mathcal{L}(y_i, f(\mathbf{x}_i, \boldsymbol{\theta}))$ is a loss function (*e.g.*, mean squared error or cross-entropy) and $f(\mathbf{x}_i, \boldsymbol{\theta})$ is the prediction function based on $\langle M_i \rangle$.

The parameters $\boldsymbol{\theta}$ are optimized using classical methods. Gradients of the cost function are estimated using the *parameter-shift rule*:

$$\frac{\partial C}{\partial \theta_k} = C(\boldsymbol{\theta}_k^+) - C(\boldsymbol{\theta}_k^-),$$

where $\boldsymbol{\theta}_k^\pm = \boldsymbol{\theta} \pm \frac{\pi}{2} e_k$, and e_k is a unit vector.

This hybrid optimization process enables the circuit to iteratively improve the model's performance, using classical optimizers like gradient descent.

5.6 Hardware Limitations

Quantum algorithms designed for applications in optimization and machine learning impose stringent demands on hardware, pushing current quantum

systems to their limits. To understand the limitations, it's essential to explore the specific needs of the most commonly used algorithms in these domains—such as the QAOA, the VQE, and quantum machine learning models (*e.g.*, quantum neural networks, quantum kernel methods). These algorithms rely heavily on the unique properties of quantum hardware to perform complex computations, but their effectiveness is directly limited by the technical constraints of current systems.

5.6.1 Introduction to IBM Quantum Hardware

The activities related to the INGENIOS project focus on IBM[®] hardware. IBM's quantum computers are based on **superconducting qubits** implemented using **Josephson junctions**. These devices leverage the principles of superconductivity and quantum mechanics to create and control qubit states, enabling complex quantum computations. Here's an overview of the main elements:

Josephson Junctions and Qubit Design:

- IBM's qubits are built using **Josephson junctions**, which consist of two superconducting materials separated by a thin insulating layer. This structure allows the controlled flow of supercurrents without electrical resistance, enabling stable quantum states.
- By applying specific microwave pulses, IBM's system can drive the qubit into a **superposition** of its two lowest energy states. These states represent the quantum analogs of classical binary 0 and 1 states, forming the basis for computations.

Gate Operations:

- Quantum gates on IBM's hardware are realized using precise microwave pulses that control the qubit's energy levels and phase. For instance:
 - **Single-qubit gates** manipulate individual qubits, such as **X**, **Y**, and **Z gates** for rotation, or the **Hadamard gate** for creating superposition.
 - **Two-qubit gates**, particularly the **CNOT gate**, are crucial for entangling qubits. IBM's two-qubit gates are implemented by coupling qubits through resonators, allowing one qubit to control the state of another.

- These gate operations are implemented with high accuracy but are limited by gate fidelity, which affects the reliability of quantum computations, especially in deep circuits.

Entanglement and Connectivity:

- IBM’s architecture supports **nearest-neighbor entanglement** on a 2D grid, where each qubit can only interact directly with adjacent qubits. Entanglement is achieved through **controlled gates** (such as CNOT), which link qubits in correlated states.
- Limited connectivity necessitates additional **SWAP gates** to allow distant qubits to interact, which increases circuit depth and error potential.

Measurement and Readout:

- At the end of each computation, IBM’s quantum systems measure qubit states to produce classical output. Each qubit’s state collapses into either the 0 or 1 basis state, with probabilities dependent on the qubit’s superposition before measurement.
- Readout errors are a challenge in superconducting qubit systems, as measurement is a noisy process, which can introduce inaccuracies in the results.

This hardware design allows IBM’s quantum processors to perform a variety of quantum algorithms, although current technological limits in coherence, fidelity, and connectivity impose constraints on complex computations, such as those in optimization and machine learning.

5.6.2 Requirements of Quantum Algorithms

Quantum algorithms designed for applications in **optimization** and **machine learning** impose stringent demands on hardware, pushing current quantum systems to their limits. To understand the limitations, it’s essential to explore the specific needs of the most commonly used algorithms in these domains—such as the QAOA, the VQE, and quantum machine learning models (*e.g.*, quantum neural networks, quantum kernel methods). These algorithms rely heavily on the unique properties of quantum hardware to perform complex computations, but their effectiveness is directly limited by the technical constraints of current systems.

Limitations Due to the Number of Qubits

One of the most straightforward but critical limitations of current IBM quantum hardware is the **limited number of qubits** available for computation. While IBM's processors have steadily increased qubit counts in recent years, with devices reaching up to 127 qubits (*e.g.*, IBM Quantum Eagle), this number remains insufficient for most complex quantum algorithms in optimization and machine learning. Here's how this impacts algorithm performance:

- **State Space Limitations:**

- Quantum algorithms benefit from larger state spaces, as each qubit doubles the number of possible states (increasing as 2^n for n qubits). Optimization problems and machine learning tasks often require exploring large solution spaces or encoding high-dimensional data, which necessitates many qubits to represent all possible configurations or features.
- With a limited number of qubits, IBM's current hardware cannot scale to solve large instances of these problems, which restricts the use of quantum algorithms to small, simplified versions of real-world tasks.

- **Data Encoding Constraints:**

- Many quantum machine learning models require encoding classical data into quantum states, which can be challenging with limited qubits. For instance, high-dimensional data may require mapping features across multiple qubits, but in current systems, the number of qubits is often insufficient to capture the full dimensionality of complex datasets.
- This limitation forces trade-offs in data representation, often requiring dimensionality reduction or more compact encoding methods, which may reduce the effectiveness of the algorithms.

- **Circuit Depth versus Qubit Count:**

- For algorithms like QAOA and VQE, increasing the number of qubits directly correlates with the size of problems that can be tackled. However, given limited qubits, there is often a balance between circuit depth (to achieve accuracy) and the number of qubits (to represent the problem size). This trade-off is particularly restrictive in current hardware, as increasing depth leads to increased error and limits practical problem-solving capability.

In summary, the limited qubit count in IBM’s quantum devices directly constrains the size and complexity of problems that can be addressed, preventing these systems from achieving true quantum advantage in large-scale optimization and machine learning applications. Increasing the number of qubits, alongside improvements in coherence and fidelity, remains a crucial goal for advancing the field.

Circuit Depth and Coherence Time

- **High Circuit Depth for Complex Algorithms:** Algorithms like QAOA and VQE are **variational** and involve multiple repetitions of parameterized quantum circuits. Each layer of these circuits increases the **circuit depth**, which is the number of sequential gate operations required. For optimization, each added layer improves the accuracy, making higher circuit depth desirable.
- **Importance of Coherence Time:** **Decoherence** occurs when a qubit loses its quantum state due to interactions with its environment. It affects both T_1 (energy relaxation) and T_2 (dephasing or coherence) times.
 - **Energy Relaxation (T_1):** This type of noise refers to the process where an excited qubit ($|1\rangle$) decays to its ground state ($|0\rangle$) due to energy loss, often to the surrounding environment. In superconducting qubits, this relaxation is due to the dissipation of energy to the quantum circuit and substrate materials.
 - **Dephasing Noise (T_2):** Dephasing noise occurs when a qubit loses coherence between the $|0\rangle$ and $|1\rangle$ states due to phase noise. This happens when a qubit’s phase is disturbed by external electromagnetic fields or temperature fluctuations, even without a change in energy levels.

In IBM’s superconducting qubit systems, coherence times typically range from tens to hundreds of microseconds. However, for deep circuits required in QAOA, VQE, and machine learning, longer coherence times are essential. The limited coherence time of current qubits thus restricts the depth and complexity of achievable circuits, leading to less accurate results in tasks that require multiple iterations. Additionally, when a quantum circuit is implemented on real IBM hardware, the effective circuit depth often increases. This is due to limitations in the *Instruction Set Architecture* (ISA), which may require breaking down

higher-level operations into multiple lower-level gates supported by the hardware. As a result, the circuit depth on hardware can be significantly greater than the depth in the idealized model, compounding coherence and fidelity challenges. This topic will be explored in more detail in later workpackage of the INGENIOS project.

High Fidelity in Gate Operations

Quantum gates manipulate qubits, and any inaccuracy in gate operations causes **gate errors**. These errors can occur due to improper calibration, timing imprecision, or control pulse distortions.

- **Over-Rotation or Under-Rotation:** When applying a quantum gate (such as an X , Y , or Z gate), slight errors in the pulse length or amplitude can cause the qubit to rotate more or less than intended, leading to errors.
- **Crosstalk between Qubits:** During multi-qubit operations, such as CNOT or controlled-phase gates, crosstalk between neighboring qubits can cause unintentional interactions that introduce errors into the system.
- **Need for Precise Quantum Gates:** Quantum gates are fundamental operations in quantum circuits, but **gate fidelity** (accuracy of the operation) directly affects overall performance. In IBM's hardware, gate fidelities have improved but remain imperfect, particularly for two-qubit gates, which are typically less reliable than single-qubit gates. For quantum machine learning and optimization algorithms, where small errors accumulate with each gate, even minor inaccuracies can significantly impact the results.
- **Demand for Reliable Multi-Layered Circuits:** Algorithms like VQE and QAOA repeatedly apply layers of gates to achieve higher accuracy in finding optimal solutions or approximations. However, lower fidelity introduces more noise as layers increase, restricting the complexity of circuits that can be implemented. For ML algorithms, which require extensive data encoding and parameterized circuits, gate errors lead to noise that affects the model's accuracy and the ability to generalize, posing a barrier to practical application.

Connectivity and Entanglement Requirements

- **Qubit Connectivity Constraints:** Algorithms for optimization and machine learning often require **entanglement** across multiple qubits to process data in highly entangled states. Ideally, every qubit could interact with any other qubit in the system, but IBM’s superconducting architecture typically supports only nearest-neighbor connectivity.
- **Multi-Qubit Entanglement for Correlated States:** For algorithms like QAOA and quantum neural networks, entanglement enables correlated states that encode complex relationships in optimization and data patterns. Restricted connectivity limits the level of entanglement achievable in practice, which constrains the ability to explore the solution space effectively or capture detailed patterns in data.

Accurate Readout and Measurement Precision

- **Impact of Measurement Errors:** Quantum algorithms end with a measurement step, where the quantum states are read out into classical results. For IBM hardware, measurement or readout errors introduce inaccuracies that can affect the reliability of final results, especially when probabilities of different states need to be estimated accurately.
- **Measurement Fidelity for Optimization and Machine Learning:** Algorithms like QAOA rely on sampling the state probabilities after the circuit runs to identify solutions. Similarly, quantum machine learning models require precise measurements to correctly interpret learned states. Measurement errors, if significant, can distort results and reduce confidence in the output, particularly for algorithms sensitive to small probability differences.

These core requirements—high coherence time, high gate fidelity, connectivity for entanglement, and precise measurement—highlight the hardware constraints that currently limit IBM’s quantum systems from realizing full quantum advantage in optimization and ML. Current hardware improvements focus on extending coherence, enhancing fidelity, and increasing qubit connectivity, but achieving true quantum advantage will require further advances to meet the demanding specifications of these algorithms.

5.7 Mitigation Strategies for Hardware Limitations

Quantum computing faces inherent hardware limitations that impact its ability to achieve high accuracy and scalability. Limitations such as **noise**, **limited qubit count**, and **restricted circuit depth** currently prevent practical quantum advantage in many areas, especially in optimization and machine learning applications. Various *mitigation strategies* have been developed to address these constraints, often balancing trade-offs between resource use and computational fidelity. These techniques, which range from quantum error correction to circuit optimization and post-processing, allow quantum systems to push beyond their hardware-imposed boundaries and maximize their computational potential. The following overview introduces the primary approaches to mitigation, setting the foundation for more advanced strategies to be discussed in depth later in this project.

5.7.1 Quantum Error Correction

Quantum Error Correction (QEC) is a robust framework designed to protect quantum information from errors due to decoherence and gate imperfections. QEC is essential for long-term quantum computing as it provides a way to maintain the stability of quantum states in systems where even minor errors can compromise the computation. Unlike classical error correction, which can use duplication (copying data) to detect and correct errors, quantum information cannot be copied directly due to the *no-cloning theorem*. Instead, QEC uses **redundant encoding** across multiple qubits to detect and correct errors.

- **Encoding Logical Qubits with Physical Qubits:** QEC schemes encode a logical qubit (a unit of quantum information) across multiple physical qubits to detect errors without collapsing the quantum state. Popular QEC codes include:
 - *The Shor Code:* This code uses 9 physical qubits to protect against both bit-flip and phase-flip errors on a single logical qubit.
 - *Surface Codes:* Surface codes have become a primary focus for scalable QEC due to their two-dimensional layout and high tolerance for error. However, they typically require dozens of physical qubits to encode a single logical qubit, which imposes a high overhead on systems with limited qubits.

- **Trade-off with Qubit Count:** Implementing QEC on near-term quantum devices comes with a significant trade-off: since each logical qubit requires many physical qubits for encoding, the number of qubits available for computation is effectively reduced. For instance, in systems with fewer qubits, dedicating many of them to QEC reduces the effective computational space, limiting the scope of problems that can be addressed. However, QEC remains a critical component of long-term quantum development, as error-corrected systems will eventually allow for deeper, more accurate circuits.

5.7.2 Strategies for Reducing Qubit Requirements

Given the current limitations in qubit count, several strategies have been developed to allow complex computations to be performed with fewer qubits:

- **Circuit Cutting:** Circuit cutting techniques allow large quantum circuits to be divided into smaller subcircuits. Each subcircuit is executed independently on a smaller set of qubits, and the results are combined through classical post-processing to approximate the results of the original, larger circuit. Circuit cutting can be used to solve larger problems on smaller hardware, albeit with increased computational cost and potential accuracy trade-offs in the classical stitching process.
- **Resource-Efficient Encoding:** In some cases, classical data can be efficiently encoded using fewer qubits by employing techniques like *amplitude encoding* or *basis encoding*, which compress information into limited quantum resources. These approaches reduce the number of qubits needed, though they may limit the types of data or operations that can be performed effectively.

By using these strategies, quantum systems can perform more complex calculations despite hardware limitations, allowing them to tackle larger or more sophisticated tasks than would otherwise be possible.

5.7.3 Strategies for Minimizing Circuit Depth

Reducing circuit depth is essential for increasing the fidelity of quantum algorithms, as the probability of errors increases with circuit depth due to limited coherence times. Several circuit optimization techniques help manage circuit depth without compromising algorithmic accuracy:

- **Gate Decomposition Optimization:** Quantum algorithms often use complex, multi-qubit gates that are not directly supported by hardware. To implement these gates, they must be decomposed into a series of simpler, native gates. Optimizing the decomposition process minimizes the number of gates, which in turn reduces circuit depth and error accumulation.
- **Compiler-Level Optimizations:** Quantum compilers, such as IBM’s Qiskit, use advanced optimization routines to restructure circuits, eliminating unnecessary gates and reordering operations to minimize depth. Compiler-level optimizations are essential for managing circuit complexity while maintaining algorithmic functionality.
- **Native Gate Usage:** Many quantum hardware platforms have a limited set of native gates that they can execute directly. By designing circuits to use these native gates wherever possible, circuit depth is reduced since fewer translation steps are needed. This helps ensure that circuits execute within the qubit coherence time, reducing error rates and improving accuracy.

Circuit optimization techniques, especially when applied systematically across all layers of a quantum algorithm, can significantly enhance the reliability of computations on near-term quantum devices.

5.7.4 Noise Reduction Strategies

Noise remains one of the most challenging barriers to reliable quantum computation. Mitigating noise is critical for improving algorithm performance, and several strategies have been developed to reduce its impact:

- **Dynamical Decoupling:** Dynamical decoupling uses a series of pulses applied to a qubit to counteract environmental noise. These pulses effectively “echo” the qubit’s state, canceling out unwanted interactions with the environment and prolonging coherence time. Dynamical decoupling is particularly useful in long-duration computations where decoherence poses a major risk.
- **Gate Calibration and Pulse Optimization:** By carefully calibrating gate parameters and fine-tuning pulse shapes, gate fidelities can be improved significantly. In IBM’s hardware, frequent calibration routines ensure that gate operations stay within tolerance limits, minimizing error. Additionally, *pulse-level optimization* can enhance the pre-

cision of gate operations, which is crucial for algorithms that require precise control over phase and amplitude.

- **Frequency Tuning and Error-Aware Scheduling:** By adjusting the frequency of qubit operations to minimize crosstalk and interference, and scheduling operations in ways that minimize error propagation, noise levels can be further reduced.

Noise reduction strategies are essential for maximizing fidelity in quantum circuits, especially for algorithms that require high precision, such as variational quantum algorithms and error-sensitive machine learning models.

5.7.5 Noise-Resilient Algorithms

Certain quantum algorithms are designed to be inherently more resilient to noise, making them better suited to current hardware:

- **Variational Quantum Algorithms:** Variational algorithms like VQE and QAOA use iterative optimization to refine parameter values. Their iterative nature allows them to tolerate certain levels of noise, as the optimization process can absorb minor deviations in gate operations. This resilience to noise makes variational algorithms particularly valuable for near-term applications, as they can still produce useful results on noisy hardware.
- **Hybrid Quantum-Classical Algorithms:** Hybrid approaches split computation between quantum and classical systems, reducing the quantum portion's complexity and making the algorithm less sensitive to noise. By offloading certain calculations to classical processors, hybrid algorithms can achieve higher accuracy while utilizing shorter, less error-prone quantum circuits.

Noise-resilient algorithms provide a pathway for performing meaningful computations on current quantum hardware, maximizing the potential of available resources while awaiting hardware improvements.

5.7.6 Post-Processing Error Mitigation

Post-processing techniques correct errors after the quantum computation has been completed, making it possible to extract useful information from noisy measurements:

- **Zero-Noise Extrapolation (ZNE):** ZNE artificially amplifies the noise in a quantum circuit and then uses this data to extrapolate back to an estimated result as if there were no noise. By running a circuit at varying levels of noise, ZNE provides a means of compensating for noise-related errors without altering the circuit itself.
- **Measurement Error Mitigation:** Measurement errors, which occur during the final readout of qubit states, can be mitigated by calibrating the readout process in advance. By determining the error rates for specific measurement outcomes, a correction matrix can be applied to account for misidentifications, improving the reliability of the measurement results.
- **Probabilistic Error Cancellation:** Probabilistic error cancellation uses classical post-processing to estimate and correct errors based on known error profiles. This method uses error models to statistically reverse the effects of noise, providing an error-mitigated result from noisy output.

These post-processing techniques are valuable in extracting accurate results from near-term quantum computations, especially in scenarios where altering the quantum circuit itself is not feasible. By correcting for errors after computation, they enhance the practicality of noisy quantum devices in real-world applications.

Chapter 6

Comparison between Classical and Quantum methods

6.1 Advantages of Quantum Approaches in Optimization

Quantum optimization methods, such as *Quantum Annealing* (QA) and the *Quantum Approximate Optimization Algorithm* (QAOA), offer a compelling alternative to classical techniques, especially for problems where classical approaches face scalability or complexity limitations. In quantum computing, two types of performance improvement are often discussed: *quantum supremacy* and *quantum advantage*.

6.1.1 Quantum Supremacy

It refers to the theoretical capability of quantum computers to solve problems that are infeasible for classical computers, regardless of the classical hardware or algorithms used. Algorithms such as Shor's and Grover's demonstrate this potential for full supremacy. Shor's algorithm, for example, allows for polynomial-time factoring of large integers, a problem considered exponentially hard for classical systems. Grover's algorithm, while more modest in speed-up, reduces the complexity of unsorted search from $\mathcal{O}(N)$ to $\mathcal{O}(\sqrt{N})$, showcasing an intrinsic quantum speed-up. However, these algorithms cannot currently be scaled for real-world supremacy due to hardware limitations, including error rates, qubit coherence, and gate fidelity.

6.1.2 Quantum Advantage

It refers to a more achievable near-term goal where quantum algorithms outperform classical counterparts on specific tasks without necessarily achieving full supremacy. Certain quantum optimization and machine learning tasks, for instance, may benefit from quantum advantage. The *kernel trick* is an example in which quantum systems access complex, high-dimensional feature spaces more naturally than classical methods, aiding in tasks like classification or clustering in machine learning. Additionally, quantum optimization algorithms, such as QAOA, exhibit scalability advantages that allow them to handle specific problem classes in polynomial time, providing substantial speed-ups over classical alternatives for certain cases.

In summary, quantum optimization methods offer several key benefits for addressing optimization problems:

1. *Exploration of vast solution spaces*: quantum algorithms use superposition to evaluate multiple solutions simultaneously, accelerating the search within large solution spaces compared to the sequential or sampled approach of classical methods.
2. *Tunneling through local minima*: quantum tunneling allows particles to pass through potential barriers, giving quantum optimization methods the ability to escape local minima in highly non-convex optimization landscapes. This advantage is especially valuable in problems where classical algorithms tend to get stuck in suboptimal regions.
3. *Optimization speed-up for specific problems*: for certain classes, quantum algorithms may provide polynomial or exponential speed-ups over classical methods. For example, Grover's and Shor's algorithms illustrate how quantum computation could theoretically solve problems beyond classical reach, while QAOA offers practical advantages for combinatorial problems as circuit depth scales.
4. *Efficiency in approximating solutions*: QAOA finds approximate solutions by optimizing quantum circuit parameters, making it particularly effective for near-optimal solutions within polynomial time. Quantum interference can further refine solution quality in cases where exact solutions are computationally infeasible.
5. *Potential for scaling with quantum hardware improvements*: quantum devices continue to improve in qubit count and coherence, suggesting that quantum optimization algorithms will eventually handle larger, more complex problems beyond current classical methods.

Overall, while quantum supremacy remains a longer-term goal dependent on hardware advancements, quantum advantage presents realistic, near-term benefits that can make quantum approaches attractive for specific types of optimization and learning problems.

Achieving quantum advantage—where quantum computers outperform classical systems in practical applications—remains an aspirational goal due to significant hardware and algorithmic challenges. Current quantum processors, such as IBM’s systems, face limitations in qubit count, coherence time, gate fidelity, and connectivity, all of which hinder the execution of complex algorithms like QAOA or quantum machine learning models at scale.

For instance, encoding real-world optimization problems often requires hundreds or thousands of qubits, far exceeding the capacity of current devices. Furthermore, the short coherence times of superconducting qubits and cumulative errors from low-fidelity gates limit the depth and accuracy of quantum circuits.

Future advancements in hardware, such as IBM’s plans for Condor processors with over 1,000 qubits, and algorithmic innovations, including noise-resilient algorithms and hybrid error mitigation techniques, hold promise for overcoming these barriers. By aligning hardware capabilities with the demands of quantum algorithms, the realization of quantum advantage could become a reality within the next decade.

6.2 Advantages of Quantum Approaches in Machine Learning

QML extends the principles of quantum computing to traditional machine learning tasks, offering the potential to surpass classical methods in efficiency and scalability. Key advantages include:

- **Access to High-Dimensional Feature Spaces**

Quantum systems naturally handle large, high-dimensional feature spaces through techniques such as quantum kernels. These enable the efficient exploration of complex data structures, a capability particularly valuable in classification, clustering, and regression tasks. For instance, *Quantum Support Vector Machines* (QSVM) utilize quantum kernels to separate non-linear data with polynomial or exponential speedup compared to classical SVMs (see Table 6.1, Chen et al. [33]).

- **Accelerated Linear Algebra Operations**

Many machine learning algorithms rely heavily on matrix manipulations, such as inversions and multiplications. Quantum algorithms,

Table 6.1: Comparison of time Complexity.

Method	Time Complexity		References
	QML	ML	
k-nearest	$\mathcal{O}(\log(n \cdot m))$	$\mathcal{O}(n \cdot m)$	[166]
PCA	$\mathcal{O}(\log(d))$	$\mathcal{O}(d)$	[112]
NN	$\mathcal{O}\left(\sqrt{N} \cdot E^2 \cdot \left(\sqrt{k} + \max_X \sqrt{k_X}\right)\right)$	$\mathcal{O}(N \cdot E \cdot k)$	[167]
k-Means	$\mathcal{O}(k \cdot \log(k \cdot m \cdot n))$	$\mathcal{O}(m \cdot n)$	[111]
SVM	$\mathcal{O}(\log(m \cdot n))$	$\mathcal{O}(m \cdot n)$	[110]

such as the HHL algorithm, significantly accelerate these computations, which are central to models like PCA and *Regression*, by reducing complexity from polynomial to logarithmic scales under specific conditions.

- Enhanced Model Expressiveness with Quantum Neural Networks (QNNs)** *Quantum Neural Networks* (QNNs) exploit parameterized quantum circuits to represent complex decision boundaries. Compared to classical neural networks, QNNs can potentially achieve equivalent performance with fewer parameters, improving training efficiency and reducing resource demands.
- Scalability to Large Datasets** By encoding data into quantum states and using quantum operations, QML can process massive datasets with fewer computational resources. This is particularly advantageous in fields like image recognition, natural language processing, and genomics, where data sizes are growing exponentially. Quantum algorithms like QkNN can exploit quantum algorithms to speed up distance calculations, reducing the time complexity from the classical $\mathcal{O}(n^2)$ (for large datasets) to potentially polynomial or logarithmic time, depending on the specific quantum algorithm used. This is particularly beneficial for large datasets, where classical kNN suffers from poor scalability as the number of data points grows. QkNN leverages the principle of quantum superposition, which allows quantum computers to process multiple possibilities simultaneously. This means that QkNN can evaluate multiple nearest neighbors in parallel, potentially reducing the time required to compute similarity and distance metrics compared to classical kNN, which processes each data point one by one.
- Overcoming Classical Limitations** Like quantum optimization, also QML benefits from the ability to escape local minima in non-convex optimization landscapes. This is achieved through techniques like quan-

tum tunneling and variational quantum algorithms, ensuring more reliable convergence in models prone to poor local optima.

- **Feasibility in Hybrid Approaches** Hybrid quantum-classical algorithms allow classical systems to handle preprocessing and postprocessing, while quantum systems address computational bottlenecks. This division of labor maximizes the utility of current *Noisy Intermediate-Scale Quantum* (NISQ) devices while preparing for future fault-tolerant quantum systems.

As with optimization tasks, QML’s success depends on the availability of robust quantum hardware. Key metrics such as *qubit coherence*, *gate fidelity*, *circuit depth*, and *error rates* will determine the practicality and scalability of QML implementations. Nonetheless, early demonstrations of quantum advantage in tasks like classification and clustering suggest a promising future for QML in both academic and industrial applications.

6.3 Disadvantages of Quantum Approaches

Quantum computing holds significant promise in optimization and machine learning, offering unique computational capabilities in specific scenarios, as discussed in the previous sections. However, it also faces several critical challenges that limit its current applicability, categorized into hardware-related and non-hardware-related constraints.

6.3.1 Hardware Limitations

As discussed in section 5.6, quantum hardware imposes significant constraints on the scalability and reliability of quantum computing solutions:

Limited Qubit Count Current quantum devices lack a sufficient number of high-quality qubits to tackle large-scale problems, making them unsuitable for many practical applications.

Short Coherence Times The fragile nature of qubits causes them to lose quantum states quickly due to decoherence, which limits the depth and complexity of executable quantum circuits.

Gate Errors and Noise Quantum gates are prone to errors, and noise in quantum systems significantly impacts the accuracy of computations, necessitating error mitigation techniques that increase computational overhead.

Hardware-Specific Constraints Physical qubit connectivity, gate fidelity, and differences in architecture across devices (*e.g.*, trapped ions versus superconducting qubits) restrict the portability and efficiency of quantum algorithms.

6.3.2 Non-Hardware Limitations

Beyond hardware issues, several challenges arise from theoretical, algorithmic, and practical considerations.

Algorithm Scalability While quantum algorithms offer theoretical speedups, their scalability often depends on problem-specific parameters. For instance, variational algorithms like QAOA may require an exponentially increasing number of layers to achieve competitive accuracy for certain problems.

Problem-Specific Speedups Quantum advantage is problem-dependent; for many tasks, classical algorithms remain more efficient.

Data Loading Overheads Quantum systems require classical data to be encoded into quantum states, a process that can become a computational bottleneck for high-dimensional or large-scale datasets.

Latency in Hybrid Workflows Many practical quantum algorithms rely on hybrid classical quantum workflows, which introduce additional complexity and latency, reducing overall efficiency.

Ethical and Security Implications Quantum computing introduces new ethical considerations and security challenges, particularly in cryptography, that need careful examination as the technology evolves.

Despite the promise of quantum computing, these limitations underscore the need for continued advancements in hardware, algorithm development, and integration strategies to address both the technical and practical barriers. For the foreseeable future, quantum methods remain complementary rather than a replacement for classical optimization and machine learning approaches.

The hardware and non-hardware limitations of quantum computing highlight the complex and multifaceted barriers that currently impede its extensive adoption. These general issues affect nearly every quantum application, irrespective of the specific domain. However, when applied to specialized fields like optimization and machine learning, these limitations manifest in unique and domain-specific ways.

The following sections examine the domain-specific disadvantages of quantum computing in optimization and machine learning, examining how the general constraints discussed above intersect with and exacerbate the challenges within these fields.

6.3.3 Disadvantages of Quantum Approaches in Optimization

While quantum optimization algorithms, such as the QAOA and QA, promise theoretical speed-ups over classical methods for certain problem classes, they face several significant limitations that restrict their practical utility:

Problem-Specific Applicability Quantum optimization methods are very specialized and work effectively only for a narrow subset of problems, such as those expressible as QUBO problems or Ising models. Reformulating general optimization problems into these forms often introduces overheads or compromises problem fidelity (Lucas [113]). Furthermore, problems with non-binary variables or constraints that are not easily captured in these frameworks may require extensive pre-processing, reducing the computational efficiency gained from the quantum approach.

Scalability Challenges for Real-World Problems Even though QAOA and QA have demonstrated promise in solving small-scale combinatorial problems, their performance on larger, real-world instances remains unproven. Studies suggest that the depth of quantum circuits required for QAOA to achieve competitive results increases significantly with problem size, often exponentially (Zhou et al. [174]). This requirement places an unsustainable demand on current hardware, limiting the algorithm’s scalability.

Similarly, quantum annealers, such as D-Wave’s systems, have shown performance improvements in specific cases but struggle to outperform classical solvers like simulated annealing or mixed-integer programming solvers on complex, large-scale problem. See Hauke et al. [75].

Sensitivity to Noise and Errors Quantum optimization methods are particularly vulnerable to noise and decoherence, which can disrupt quantum states during computation. For example, in QAOA, small errors in the implementation of quantum gates or inaccuracies in parameter tuning can significantly impact the quality of the obtained solutions (Guerreschi and Smelyanskiy [67]). Quantum annealers are also sensitive to environmental noise, which can lead to suboptimal solutions

due to premature qubit decoherence during the annealing process. See Preskill [130].

Performance Gaps with Classical Counterparts Quantum approaches have yet to consistently outperform advanced classical algorithms for many practical optimization problems. For instance, classical commercial solvers such as Gurobi or CPLEX, which leverage decades of algorithmic and hardware optimization, often achieve better results within shorter timeframes for industrial-scale problems. This raises questions about the practicality of deploying quantum algorithms outside niche use cases (Rønnow et al. [136], Egger et al. [48]).

Limited Flexibility in Problem Constraints Quantum optimization algorithms are less flexible than classical solvers in handling complex constraints, such as integer variables, non-linearities, or dynamically changing objectives. Encoding these constraints in quantum frameworks like QUBO often requires introducing penalty terms, which can inflate problem size and degrade solution accuracy; see Hauke et al. [75], Lucas [113].

High Dependence on Hybrid Workflows Many quantum optimization methods rely on hybrid classical-quantum architectures, where the quantum system evaluates potential solutions while a classical system optimizes parameters or pre-processes data. These hybrid workflows introduce latency and require significant computational resources, often negating the potential benefits of the quantum component. See Egger et al. [48].

Energy and Cost Considerations The operational cost of quantum optimization remains high due to the energy-intensive requirements of maintaining quantum systems at cryogenic temperatures and mitigating noise. In comparison, classical optimization solvers are more energy-efficient and cost-effective, making them a more practical choice for most applications. See Preskill [130].

These limitations highlight the current immaturity of quantum optimization algorithms and their reliance on hardware advancements and algorithmic innovation. While ongoing research explores ways to mitigate these challenges, quantum optimization methods remain complementary to, rather than a replacement for, classical optimization techniques in most real-world scenarios.

6.3.4 Disadvantages of Quantum Approaches in Machine Learning

QML holds the promise of leveraging quantum computing to revolutionize traditional machine learning tasks. However, it faces several unique challenges that hinder its development and practical application. These challenges stem from both hardware limitations and theoretical complexities.

Training Challenges and Barren Plateaus One of the primary issues in QML is the difficulty of training quantum models. *Variational Quantum Circuits* (VQCs), commonly used in QML, suffer from barren plateaus—regions in the optimization landscape where the gradient vanishes, making it extremely difficult to update parameters effectively. This issue worsens as the system scales, leading to exponentially flat landscapes in deep circuits. McClean et al. [117] showed that barren plateaus become more pronounced with increasing circuit depth and qubit count, significantly hindering optimization efforts in QML models like QNNs and *Variational Quantum Classifiers* (VQCs).

Data Encoding Bottlenecks Encoding large-scale, high-dimensional classical data into quantum states remains a critical bottleneck in QML. The process of mapping classical data into quantum amplitudes, known as quantum state preparation, often has a complexity of $\mathcal{O}(2^n)$, where n is the number of qubits required to represent the data. This step can negate potential quantum speed-ups, especially for high-dimensional datasets commonly encountered in machine learning. Schuld and Petruccione [145] emphasize that without efficient encoding schemes, the practical advantages of QML are limited to relatively small or low-dimensional datasets.

Hardware Constraints and Circuit Depth Many QML algorithms rely on deep quantum circuits to exploit quantum entanglement and interference. However, current quantum hardware suffers from noise, short coherence times, and low gate fidelity, making it challenging to execute deep or complex circuits. For instance, Wang et al. [161] highlighted that even shallow circuits for QSVMs experience significant performance degradation due to hardware noise, limiting their practical scalability. As a result, most QML implementations remain experimental and constrained to small datasets with relatively simple circuits.

Interpretability Challenges Interpretability is another critical issue in QML. Classical machine learning models, such as linear regression or

decision trees, are often valued for their transparency and ability to explain predictions. In contrast, quantum models—relying on quantum states and operators—are inherently non-intuitive and challenging to interpret. Lloyd et al. [112] note that the abstract nature of quantum mechanics introduces significant barriers to understanding how quantum systems process information, making QML models harder to trust in critical applications such as healthcare or finance.

Hybrid Complexity Most practical QML applications today rely on hybrid quantum-classical algorithms, where classical systems handle preprocessing and postprocessing, while quantum systems execute the computationally intensive core tasks. While this approach makes use of current *Noisy Intermediate-Scale Quantum* (NISQ) devices, it introduces significant integration challenges. Communication overhead between quantum and classical systems can erode speed-ups, as noted by Benedetti et al. [18], who demonstrated that the overall performance of hybrid QML often depends more on the classical component than the quantum one.

Benchmarking Against Classical ML Despite theoretical advantages, a lot of QML approaches struggle to outperform highly optimized classical algorithms in practice. For instance, Tang [153] showed that certain quantum-inspired algorithms for kernel-based methods can achieve similar speed-ups on classical hardware without requiring quantum computation. This raises questions about the practical utility of quantum methods, especially given the significant hardware constraints and algorithmic challenges.

Resource-Intensive Requirements QML models require extensive computational resources, even in their classical preprocessing stages, to prepare data and optimize parameters. For instance, hybrid QNNs often require classical optimization algorithms to tune quantum circuit parameters, which can be computationally expensive, particularly for large datasets or deep circuits. As the number of qubits increases, the resource demands for classical simulation of quantum models grow exponentially, as highlighted by Preskill [130] in his discussion of the NISQ era.

Quantum machine learning offers exciting potential but faces several disadvantages that limit its current applicability. The above listed challenges underscore the need for further advancements. Addressing these barriers will require significant improvements in both quantum hardware and algorithm

design, as well as a deeper understanding of how to effectively integrate quantum computing into existing machine learning workflows.

Chapter 7

Conclusions and Recommendations

The initial stages of the INGENIOS project have provided valuable insights into both the potential and the limitations of quantum computing to solve complex, real-world problems relevant to our business and clients. By rigorously analyzing and simulating quantum algorithms for optimization and machine learning tasks, we have identified areas where quantum computing could deliver distinct advantages, along with significant technical barriers that remain. In particular, the VRP and quantum machine learning applications were evaluated with realistic constraints, revealing both the practical feasibility and current challenges posed by quantum hardware limitations.

This chapter summarizes the key findings and presents a roadmap for the next phases of the project, aiming to address the technical, economic, and workflow requirements necessary to bring quantum computing into practical use.

7.1 Key Findings

- *Quantum Computing Feasibility:* quantum computing shows promise for solving complex optimization problems like VRP and supporting machine learning applications under certain conditions. Nevertheless, achieving this potential in practical applications requires overcoming hardware limitations, such as limited qubit counts, coherence times, and high error rates, which restrict the depth and complexity of feasible circuits.
- *Performance Limitations on Current Hardware:* IBM's quantum processors, while steadily improving, still present constraints in terms of

qubit connectivity, gate fidelity, and coherence times. This imposes practical limits on both the circuit depth and size, restricting the complexity of solvable problems. Additionally, the financial costs associated with running complex algorithms on premium quantum hardware highlight a need for further cost-benefit analysis.

- *Workflow Complexity and Resource Requirements*: the complete workflow, from classical problem formulation to quantum circuit design, transpilation, and execution, requires significant computational and human resources. Classical simulations, circuit transpilation, and performance optimization require considerable time and expertise, especially given the need for algorithm adjustments to fit the hardware’s technical constraints.
- *Mitigation Strategies in Practice*: effective error mitigation and circuit optimization techniques are essential for improving computation quality on current quantum hardware. Techniques like quantum error correction, circuit transpilation, and noise mitigation (*e.g.*, zero-noise extrapolation) were found to be effective but resource-intensive. The analysis reveals that a hybrid quantum-classical approach is often necessary to achieve reasonable accuracy.

7.2 Strategic Roadmap for Future Phases

To leverage quantum computing effectively and prepare for a potential quantum advantage, the following roadmap is recommended for the next phases of the INGENIOS project:

1. Advanced Hardware Benchmarking and Selection:

- Conduct systematic performance evaluations across different quantum processors to identify optimal hardware platforms for various use cases, focusing on quantum volume, gate fidelity, and cost per computation.
- Assess potential collaborations with quantum hardware providers to access cutting-edge machines with higher qubit counts and improved error rates, enabling more complex problem-solving capabilities.

2. Enhanced Problem Modeling and Complexity Analysis:

- Deepen the analysis of complex real-world constraints in problems like VRP by exploring advanced quantum algorithms (*e.g.*, more layered QAOA or hybrid classical-quantum algorithms) that could address constraints like employee shifts, vehicle types, and routing options.
- Conduct a scalability assessment to determine how algorithm performance changes with increasing problem size and complexity, providing insight into the feasibility of addressing larger, more intricate cases as hardware capabilities advance.

3. Development of Efficient Workflow Pipelines:

- Establish automated pipelines for the workflow stages from classical problem formulation to quantum computation, emphasizing circuit optimization and transpilation. This will reduce resource consumption and streamline the process for future projects.
- Incorporate end-to-end cost tracking for each workflow stage to monitor financial efficiency and adjust approaches based on budget constraints.

4. Implementation of Advanced Error Mitigation Techniques:

- Expand the use of error mitigation strategies, such as dynamical decoupling and measurement error mitigation, to improve computation accuracy without significant increases in circuit depth.
- Continue to develop and test hybrid quantum-classical algorithms, which offer greater noise resilience and could yield usable results on near-term quantum devices even with current error rates.

5. Building a Business Case for Quantum Adoption:

- Quantify the potential value-add of quantum computing for our specific business applications, integrating financial cost, computational gains, and scalability into the analysis.
- Prepare a cost-benefit assessment to determine under what conditions and for which applications quantum computing would offer a meaningful advantage over classical computing.
- Develop educational initiatives to build quantum expertise within the company, ensuring that our team is well-prepared to integrate quantum solutions into real-world operations as the technology matures.

6. Long-Term Research and Development:

- Pursue ongoing R&D on emerging quantum algorithms and technologies, such as quantum machine learning architectures and error-corrected quantum computing, that could bring us closer to practical quantum advantage.
- Investigate potential quantum-cloud partnerships and use cases for client applications, positioning us to leverage quantum computing as it becomes more commercially viable.

7.3 Conclusion

The INGENIOS project’s findings underscore both the promise and the current limitations of quantum computing. Although real-world applications remain challenging, quantum computing offers unique problem-solving capabilities that are expected to become transformative as hardware and algorithms advance. By following this roadmap, we aim to ensure that our organization remains at the forefront of this emerging field, positioning ourselves to harness quantum advantage as the technology develops. Through rigorous benchmarking, workflow optimization, and continuous R&D, we can create a robust foundation for the effective integration of quantum computing into practical applications, thereby delivering significant value to our clients and staying ahead in a competitive landscape.

Bibliography

- [1] Genetic algorithms | ACM Computing Surveys.
<https://dl.acm.org/doi/10.1145/234313.234350>.
- [2] Branch and bound methods. In Panos M. Pardalos and J. Ben Rosen, editors, *Constrained Global Optimization: Algorithms and Applications*, volume 268, pages 58–74, Berlin/Heidelberg, 1987. Springer-Verlag. ISBN 978-3-540-18095-1. doi: 10.1007/BFb0000041.
- [3] Scott Aaronson. The Learnability of Quantum States. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 463(2088):3089–3114, December 2007. ISSN 1364-5021, 1471-2946. doi: 10.1098/rspa.2007.0113.
- [4] Amira Abbas, David Sutter, Alessio Figalli, and Stefan Woerner. Effective dimension of machine learning models, December 2021.
- [5] Amira Abbas, David Sutter, Christa Zoufal, Aurélien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, June 2021. ISSN 2662-8457. doi: 10.1038/s43588-021-00084-1.
- [6] Amira Abbas, Andris Ambainis, Brandon Augustino, Andreas Bärtschi, Harry Buhrman, Carleton Coffrin, Giorgio Cortiana, Vedran Dunjko, Daniel J. Egger, Bruce G. Elmegreen, Nicola Franco, Filippo Fratini, Bryce Fuller, Julien Gacon, Constantin Gonciulea, Sander Gribling, Swati Gupta, Stuart Hadfield, Raoul Heese, Gerhard Kircher, Thomas Kleinert, Thorsten Koch, Georgios Korpas, Steve Lenk, Jakub Marecek, Vanio Markov, Guglielmo Mazzola, Stefano Mensa, Naeimeh Mohseni, Giacomo Nannicini, Corey O’Meara, Elena Peña Tapia, Sebastian Pokutta, Manuel Proissl, Patrick Rebentrost, Emre Sahin, Benjamin C. B. Symons, Sabine Törnøw, Victor Valls, Stefan Woerner, Mira L. Wolf-Bauwens, Jon Yard, Sheir Yarkoni, Dirk Zechiel, Sergiy

- Zhuk, and Christa Zoufal. Quantum Optimization: Potential, Challenges, and the Path Forward, September 2024.
- [7] Shigeo Abe and Takuya Inoue. Fast Training of Support Vector Machines by Extracting Boundary Data. In Gerhard Goos, Juris Hartmanis, Jan Van Leeuwen, Georg Dorffner, Horst Bischof, and Kurt Hornik, editors, *Artificial Neural Networks — ICANN 2001*, volume 2130, pages 308–313. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. ISBN 978-3-540-42486-4 978-3-540-44668-2. doi: 10.1007/3-540-44668-0_44.
 - [8] Zainab Abohashima, Mohamed Elhosen, Essam H. Houssein, and Waleed M. Mohamed. Classification with Quantum Machine Learning: A Survey, 2020.
 - [9] Ali Asghar Ahmadikia, Arman Shirzad, and Ali Mohammad Saghiri. A Bibliometric Analysis of Quantum Machine Learning Research. *Science & Technology Libraries*, 43(2):202–223, April 2024. ISSN 0194-262X, 1541-1109. doi: 10.1080/0194262X.2023.2292049.
 - [10] Mohammad Hassan Almaspoor, Ali Safaei, Afshin Salajegheh, and Behrouz Minaei-Bidgoli. Support Vector Machines in Big Data Classification: A Systematic Literature Review, August 2021.
 - [11] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1):53, March 2021. ISSN 2196-1115. doi: 10.1186/s40537-021-00444-8.
 - [12] David Arthur and Sergei Vassilvitskii. K-means++: The Advantages of Careful Seeding. *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, January 2007.
 - [13] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent Measures of Risk. *Mathematical Finance*, 9(3):203–228, 1999. ISSN 1467-9965. doi: 10.1111/1467-9965.00068.
 - [14] Enrico Barbierato and Alice Gatti. The Challenges of Machine Learning: A Critical Review. *Electronics*, 13(2):416, January 2024. ISSN 2079-9292. doi: 10.3390/electronics13020416.

- [15] Güzin Bayraksan and David K. Love. Data-Driven Stochastic Programming Using Phi-Divergences. In Dionne Aleman, Aurélie Thiele, J. Cole Smith, and Harvey J. Greenberg, editors, *The Operations Research Revolution*, pages 1–19. INFORMS, September 2015. ISBN 978-0-9843378-8-0. doi: 10.1287/educ.2015.0134.
- [16] G. Bebis and M. Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31, October 1994. ISSN 0278-6648. doi: 10.1109/45.329294.
- [17] Aharon Ben-Tal, Laurent Ghaoui, and Arkadi Nemirovski. *Robust Optimization*. August 2009. ISBN 978-1-4008-3105-0. doi: 10.1515/9781400831050.
- [18] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. 2019. doi: 10.48550/ARXIV.1906.07682.
- [19] Massimo Bernaschi, Isidoro González-Adalid Pemartín, Víctor Martín-Mayor, and Giorgio Parisi. The quantum transition of the two-dimensional Ising spin glass. *Nature*, 631(8022):749–754, July 2024. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-024-07647-y.
- [20] Dimitris Bertsimas and John Tsitsiklis. Simulated Annealing. *Statistical Science*, 8(1), February 1993. ISSN 0883-4237. doi: 10.1214/ss/1177011077.
- [21] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York, 2006. ISBN 978-0-387-31073-2.
- [22] Kostas Blekos, Dean Brand, Andrea Ceschini, Chiao-Hui Chou, Rui-Hao Li, Komal Pandya, and Alessandro Summer. A review on Quantum Approximate Optimization Algorithm and its variants. *Physics Reports*, 1068:1–66, June 2024. ISSN 0370-1573. doi: 10.1016/j.physrep.2024.03.002.
- [23] Sami Boulebnane and Ashley Montanaro. Solving boolean satisfiability problems with the quantum approximate optimization algorithm, August 2022.
- [24] Sergey Bravyi, Andrew W. Cross, Jay M. Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J. Yoder. High-threshold and low-overhead

- fault-tolerant quantum memory. *Nature*, 627(8005):778–782, March 2024. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-024-07107-7.
- [25] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001. ISSN 08856125. doi: 10.1023/A:1010933404324.
- [26] Christopher J.C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2): 121–167, 1998. ISSN 13845810. doi: 10.1023/A:1009715923555.
- [27] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The State of the Art of Nurse Rostering. *Journal of Scheduling*, 7(6):441–499, November 2004. ISSN 1099-1425. doi: 10.1023/B:JOSH.0000046076.75950.0b.
- [28] Aishwarya C, Venkatesan M, and Prabhavathy P. Research Oriented Reviewing of Quantum Machine Learning. *International Research Journal on Advanced Science Hub*, 5(Issue 05S):165–177, May 2023. ISSN 2582-4376. doi: 10.47392/irjash.2023.S022.
- [29] Almudena Carrera Vazquez, Caroline Tornow, Diego Ristè, Stefan Wöerner, Maika Takita, and Daniel J. Egger. Combining quantum processors with real-time classical communication. *Nature*, November 2024. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-024-08178-2.
- [30] M. Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. Variational Quantum Algorithms. *Nature Reviews Physics*, 3(9):625–644, August 2021. ISSN 2522-5820. doi: 10.1038/s42254-021-00348-9.
- [31] P. Chandarana, N. N. Hegade, K. Paul, F. Albarrán-Arriagada, E. Solano, A. Del Campo, and Xi Chen. Digitized-counterdiabatic quantum approximate optimization algorithm. *Physical Review Research*, 4(1):013141, February 2022. ISSN 2643-1564. doi: 10.1103/PhysRevResearch.4.013141.
- [32] Linshu Chen, Lei Zhao, Zhenguo Xiao, Yuanhui Liu, and Jiayang Wang. A Granular Computing Based Classification Method From Algebraic Granule Structure. *IEEE Access*, 9:68118–68126, 2021. ISSN 2169-3536. doi: 10.1109/ACCESS.2021.3077409.

- [33] Linshu Chen, Tao Li, Yuxiang Chen, Xiaoyan Chen, Marcin Wozniak, Neal Xiong, and Wei Liang. Design and analysis of quantum machine learning: A survey. *Connection Science*, 36(1):2312121, December 2024. ISSN 0954-0091, 1360-0494. doi: 10.1080/09540091.2024.2312121.
- [34] Bao Chong. K-means clustering algorithm: A brief review. *Academic Journal of Computing & Information Science*, 4(5), 2021. ISSN 26165775. doi: 10.25236/AJCIS.2021.040506.
- [35] Jean-François Cordeau, Gilbert Laporte, Martin W.P. Savelsbergh, and Daniele Vigo. Chapter 6 Vehicle Routing. In *Handbooks in Operations Research and Management Science*, volume 14, pages 367–428. Elsevier, 2007. ISBN 978-0-444-51346-5. doi: 10.1016/S0927-0507(06)14006-2.
- [36] Thomas H. Cormen, Charles Eric Leiserson, Ronald Linn Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts London, fourth edition edition, 2022. ISBN 978-0-262-04630-5 978-0-262-36750-9.
- [37] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00994018.
- [38] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967. ISSN 0018-9448, 1557-9654. doi: 10.1109/TIT.1967.1053964.
- [39] Giacomo Da Col and Erich C. Teppan. Industrial-size job shop scheduling with constraint programming. *Operations Research Perspectives*, 9: 100249, 2022. ISSN 22147160. doi: 10.1016/j.orp.2022.100249.
- [40] James Dargan. What Types of Quantum Computers Exist In 2024? <https://thequantuminsider.com/2023/06/06/types-of-quantum-computers/>, June 2023.
- [41] Matthew DeCross, Eli Chertkov, Megan Kohagen, and Michael Foss-Feig. Qubit-reuse compilation with mid-circuit measurement and reset, October 2022.
- [42] Simon J. Devitt, Kae Nemoto, and William J. Munro. Quantum Error Correction for Beginners. *Reports on Progress in Physics*, 76(7):076001,

- July 2013. ISSN 0034-4885, 1361-6633. doi: 10.1088/0034-4885/76/7/076001.
- [43] Marco Dorigo, Mauro Birattari, and Thomas Stützle. Ant Colony Optimization. *Computational Intelligence Magazine, IEEE*, 1:28–39, December 2006. doi: 10.1109/MCI.2006.329691.
 - [44] Tamer Doyuran and Bülent Çatay. A robust enhancement to the Clarke-Wright savings algorithm. *JORS*, 62:223–231, February 2011. doi: 10.1057/jors.2009.176.
 - [45] Ding-Zhu Du and Ker-I. Ko. *Theory of Computational Complexity*. John Wiley & Sons, October 2011. ISBN 978-1-118-03116-2.
 - [46] Yuxuan Du, Tao Huang, Shan You, Min-Hsiu Hsieh, and Dacheng Tao. Quantum circuit architecture search for variational quantum algorithms. *npj Quantum Information*, 8(1):62, May 2022. ISSN 2056-6387. doi: 10.1038/s41534-022-00570-y.
 - [47] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Quantum-Enhanced Machine Learning. *Physical Review Letters*, 117(13):130501, September 2016. ISSN 0031-9007, 1079-7114. doi: 10.1103/PhysRevLett.117.130501.
 - [48] Daniel J. Egger, Jakub Marecek, and Stefan Woerner. Warm-starting quantum optimization. 2020. doi: 10.48550/ARXIV.2009.10095.
 - [49] Ethan N. Evans, Dominic Byrne, and Matthew G. Cook. A Quick Introduction to Quantum Machine Learning for Non-Practitioners, February 2024.
 - [50] Nic Ezzell, Bibek Pokharel, Lina Tewala, Gregory Quiroz, and Daniel A. Lidar. Dynamical decoupling for superconducting qubits: A performance survey. *Physical Review Applied*, 20(6):064027, December 2023. ISSN 2331-7019. doi: 10.1103/PhysRevApplied.20.064027.
 - [51] Jennifer Faj, Ivy Peng, Jacob Wahlgren, and Stefano Markidis. Quantum Computer Simulations at Warp Speed: Assessing the Impact of GPU Acceleration, July 2023.
 - [52] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A Quantum Approximate Optimization Algorithm, November 2014.
 - [53] Matteo Fischetti. *Introduction to Mathematical Optimization*. M. Fischetti, 2019. ISBN 978-1-6927-9202-2.

- [54] Francisca Chibugo Udegbe, Ogochukwu Roseline Ebulue, Charles Chukwudalu Ebulue, and Chukwunonso Sylvester Ekesiobi. MACHINE LEARNING IN DRUG DISCOVERY: A CRITICAL REVIEW OF APPLICATIONS AND CHALLENGES. *Computer Science & IT Research Journal*, 5(4):892–902, April 2024. ISSN 2709-0051, 2709-0043. doi: 10.51594/csitrj.v5i4.1048.
- [55] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5), October 2001. ISSN 0090-5364. doi: 10.1214/aos/1013203451.
- [56] Elies M Gil Fuster and Jose Ignacio Latorre. Variational Quantum Classifier.
- [57] Jian Gao, Yida Xu, Julian Barreiro-Gomez, Massa Ndong, Michail Smyrnakis, Hamidou Gao, Michalis Smyrnakis, and Hamidou Tembine. Distributionally Robust Optimization. September 2018. ISBN 978-1-78923-676-7. doi: 10.5772/intechopen.76686.
- [58] Babacar Gaye, Dezheng Zhang, and Aziguli Wulamu. Improvement of Support Vector Machine Algorithm in Big Data Background. *Mathematical Problems in Engineering*, 2021:1–9, June 2021. ISSN 1563-5147, 1024-123X. doi: 10.1155/2021/5594899.
- [59] Gian Gentinetta, Arne Thomsen, David Sutter, and Stefan Woerner. The complexity of quantum support vector machines. *Quantum*, 8:1225, January 2024. ISSN 2521-327X. doi: 10.22331/q-2024-01-11-1225.
- [60] Gian Gentinetta, Arne Thomsen, David Sutter, and Stefan Woerner. The complexity of quantum support vector machines. *Quantum*, 8:1225, January 2024. ISSN 2521-327X. doi: 10.22331/q-2024-01-11-1225.
- [61] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum Random Access Memory. *Physical Review Letters*, 100(16):160501, April 2008. ISSN 0031-9007, 1079-7114. doi: 10.1103/PhysRevLett.100.160501.
- [62] Fred Glover. Tabu Search: A Tutorial. *Interfaces*, 20(4):74–94, August 1990. ISSN 0092-2102, 1526-551X. doi: 10.1287/inte.20.4.74.
- [63] Bruce Golden, S. Raghavan, and Edward Wasil, editors. *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43

- of *Operations Research/Computer Science Interfaces*. Springer US, Boston, MA, 2008. ISBN 978-0-387-77777-1 978-0-387-77778-8. doi: 10.1007/978-0-387-77778-8.
- [64] Daniel Goldsmith and M M Hassan Mahmud. Machine Learning for Quantum Computing Specialists.
 - [65] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning. The MIT press, Cambridge, Mass, 2016. ISBN 978-0-262-03561-3.
 - [66] A. D. Gordon, L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and Regression Trees. *Biometrics*, 40(3):874, September 1984. ISSN 0006341X. doi: 10.2307/2530946.
 - [67] Gian Giacomo Guerreschi and Mikhail Smelyanskiy. Practical optimization for hybrid quantum-classical algorithms, 2017.
 - [68] Rosita Guido, Stefania Ferrisi, Danilo Lofaro, and Domenico Conforti. An Overview on the Advancements of Support Vector Machine Models in Healthcare Applications: A Review. *Information*, 15(4):235, April 2024. ISSN 2078-2489. doi: 10.3390/info15040235.
 - [69] Valérie Guihaire and Jin-Kao Hao. Transit network design and scheduling: A global review. *Transportation Research Part A: Policy and Practice*, 42(10):1251–1273, December 2008. ISSN 0965-8564. doi: 10.1016/j.tra.2008.03.011.
 - [70] Yaswitha Gujju, Atsushi Matsuo, and Rudy Raymond. Quantum Machine Learning on Near-Term Quantum Devices: Current State of Supervised and Unsupervised Techniques for Real-World Applications, June 2024.
 - [71] Stuart Hadfield. Quantum Algorithms for Scientific Computing and Approximate Optimization, May 2018.
 - [72] Stuart Hadfield, Zhihui Wang, Bryan O’Gorman, Eleanor G. Rieffel, Davide Venturelli, and Rupak Biswas. From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz. *Algorithms*, 12(2):34, February 2019. ISSN 1999-4893. doi: 10.3390/a12020034.

- [73] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum Algorithm for Linear Systems of Equations. *Physical Review Letters*, 103(15):150502, October 2009. ISSN 0031-9007, 1079-7114. doi: 10.1103/PhysRevLett.103.150502.
- [74] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, New York, NY, 10. [print.], (corr. as of 4. print.) edition, 2008. ISBN 978-0-387-95284-0.
- [75] Philipp Hauke, Helmut G. Katzgraber, Wolfgang Lechner, Hidetoshi Nishimori, and William D. Oliver. Perspectives of quantum annealing: Methods and implementations. 2019. doi: 10.48550/ARXIV.1903.06559.
- [76] Vojtech Havlicek, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. Supervised learning with quantum enhanced feature spaces. *Nature*, 567(7747):209–212, March 2019. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-019-0980-2.
- [77] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016. IEEE. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.90.
- [78] Itay Hen and Federico M. Spedalieri. Quantum Annealing for Constrained Optimization. *Physical Review Applied*, 5(3):034007, March 2016. ISSN 2331-7019. doi: 10.1103/PhysRevApplied.5.034007.
- [79] Rebekah Herrman, Phillip C. Lotshaw, James Ostrowski, Travis S. Humble, and George Siopsis. Multi-angle quantum approximate optimization algorithm. *Scientific Reports*, 12(1):6781, April 2022. ISSN 2045-2322. doi: 10.1038/s41598-022-10555-8.
- [80] Alain Hertz and Marino Widmer. Guidelines for the use of meta-heuristics in combinatorial optimization. *European Journal of Operational Research*, 151(2):247–252, December 2003. ISSN 0377-2217. doi: 10.1016/S0377-2217(02)00823-8.
- [81] Hsin-Yuan Huang, Michael Broughton, Masoud Mohseni, Ryan Babush, Sergio Boixo, Hartmut Neven, and Jarrod R. McClean. Power

- of data in quantum machine learning. *Nature Communications*, 12(1): 2631, May 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-22539-9.
- [82] Lawrence Hubert, Phipps Arabie, and Jacqueline Meulman. *Combinatorial Data Analysis: Optimization by Dynamic Programming*. Society for Industrial and Applied Mathematics, January 2001. ISBN 978-0-89871-478-4 978-0-89871-855-3. doi: 10.1137/1.9780898718553.
- [83] Nouhaila Innan, Muhammad Al-Zafar Khan, Biswaranjan Panda, and Mohamed Bennai. Enhancing Quantum Support Vector Machines through Variational Kernel Training. *Quantum Information Processing*, 22(10):374, October 2023. ISSN 1573-1332. doi: 10.1007/s11128-023-04138-3.
- [84] Abhijith J, Adetokunbo Adedoyin, John Ambrosiano, Petr Anisimov, William Casper, Gopinath Chennupati, Carleton Coffrin, Hristo Djidjev, David Gunter, Satish Karra, Nathan Lemons, Shizeng Lin, Alexander Malyzhenkov, David Mascarenas, Susan Mniszewski, Balu Nadiga, Daniel O’Malley, Diane Oyen, Scott Pakin, Lakshman Prasad, Randy Roberts, Phillip Romero, Nandakishore Santhi, Nikolai Sinityn, Pieter J. Swart, James G. Wendelberger, Boram Yoon, Richard Zamora, Wei Zhu, Stephan Eidenbenz, Andreas Bäertschi, Patrick J. Coles, Marc Vuffray, and Andrey Y. Lokhov. Quantum Algorithm Implementations for Beginners. *ACM Transactions on Quantum Computing*, 3(4):1–92, December 2022. ISSN 2643-6809, 2643-6817. doi: 10.1145/3517340.
- [85] Nishchal J and Neel Bhandari. Computational Complexity of Gradient Descent Algorithm, May 2021.
- [86] Abhishek Jadhav, Akhtar Rasool, and Manasi Gyanchandani. Quantum Machine Learning: Scope for real-world problems. *Procedia Computer Science*, 218:2612–2625, 2023. ISSN 18770509. doi: 10.1016/j.procs.2023.01.235.
- [87] Abhishek Jadhav, Akhtar Rasool, and Manasi Gyanchandani. Quantum Machine Learning: Scope for real-world problems. *Procedia Computer Science*, 218:2612–2625, 2023. ISSN 18770509. doi: 10.1016/j.procs.2023.01.235.
- [88] Subrahmanyam Jammalamadaka and Aswani Kumar Cherukuri. Insights into Quantum Support Vector Machine, June 2024.

- [89] Muhammad Adeel Javaid. Understanding Dijkstra Algorithm. *SSRN Electronic Journal*, 2013. ISSN 1556-5068. doi: 10.2139/ssrn.2340905.
- [90] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic Generalization Measures and Where to Find Them, December 2019.
- [91] M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. S. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, May 2011. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature10012.
- [92] I. T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, New York, 2nd ed edition, 2002. ISBN 978-0-387-95442-4.
- [93] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: A review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, April 2016. ISSN 1364-503X, 1471-2962. doi: 10.1098/rsta.2015.0202.
- [94] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse Ising model. *Physical Review E*, 58(5):5355–5363, November 1998. ISSN 1063-651X, 1095-3787. doi: 10.1103/PhysRevE.58.5355.
- [95] Subhash C. Kak. Quantum Neural Computing. In *Advances in Imaging and Electron Physics*, volume 94, pages 259–313. Elsevier, 1995. ISBN 978-0-12-014736-6. doi: 10.1016/S1076-5670(08)70147-2.
- [96] Aleksandr M. Kazachkov, Pierre Le Bodic, and Sriram Sankaranarayanan. An abstract model for branch and cut. *Mathematical Programming*, 206(1):175–202, July 2024. ISSN 1436-4646. doi: 10.1007/s10107-023-01991-z.
- [97] Iordanis Kerenidis and Anupam Prakash. Quantum Recommendation Systems, 2016.
- [98] Sarit Khirirat, Hamid Reza Feyzmahdavian, and Mikael Johansson. Mini-batch gradient descent: Faster convergence under data sparsity. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*,

- pages 2880–2887, Melbourne, Australia, December 2017. IEEE. ISBN 978-1-5090-2873-3. doi: 10.1109/CDC.2017.8264077.
- [99] Gitae Kim, Yew-Soon Ong, Chen Kim Heng, Puay Siew Tan, and Nengsheng Allan Zhang. City Vehicle Routing Problem (City VRP): A Review. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):1654–1666, August 2015. ISSN 1524-9050, 1558-0016. doi: 10.1109/TITS.2015.2395536.
 - [100] Youngseok Kim, Andrew Eddins, Sajant Anand, Ken Xuan Wei, Ewout Van Den Berg, Sami Rosenblatt, Hasan Nayfeh, Yantao Wu, Michael Zaletel, Kristan Temme, and Abhinav Kandala. Evidence for the utility of quantum computing before fault tolerance. *Nature*, 618(7965):500–505, June 2023. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-023-06096-3.
 - [101] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Publishing Company, Incorporated, 5th edition, 2012. ISBN 978-3-642-24487-2.
 - [102] Oliver Kramer. *Genetic Algorithms*, volume 679, pages 11–19. Springer International Publishing, Cham, 2017. ISBN 978-3-319-52155-8 978-3-319-52156-5. doi: 10.1007/978-3-319-52156-5_2.
 - [103] Philip Krantz, Morten Kjaergaard, Fei Yan, Terry P. Orlando, Simon Gustavsson, and William D. Oliver. A Quantum Engineer’s Guide to Superconducting Qubits. *Applied Physics Reviews*, 6(2):021318, June 2019. ISSN 1931-9401. doi: 10.1063/1.5089550.
 - [104] David Kremer, Victor Villar, Hanhee Paik, Ivan Duran, Ismael Faro, and Juan Cruz-Benito. Practical and efficient quantum circuit synthesis and transpiling with Reinforcement Learning, May 2024.
 - [105] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. ISSN 1931-9193. doi: 10.1002/nav.3800020109.
 - [106] Amit Kumar, Neha Sharma, Nikhil Kumar Marriwala, Sunita Panda, M. Aruna, and Jeetendra Kumar. Quantum machine learning with Qiskit: Evaluating regression accuracy and noise impact. *IET Quantum Communication*, page qtc2.12100, July 2024. ISSN 2632-8925, 2632-8925. doi: 10.1049/qtc2.12100.

- [107] Gilbert Laporte. Fifty Years of Vehicle Routing. *Transportation Science*, 43(4):408–416, November 2009. ISSN 0041-1655, 1526-5447. doi: 10.1287/trsc.1090.0301.
- [108] Ryan LaRose and Brian Coyle. Robust data encodings for quantum classifiers. *Physical Review A*, 102(3):032420, September 2020. ISSN 2469-9926, 2469-9934. doi: 10.1103/PhysRevA.102.032420.
- [109] Kai Li, Hongyi Dai, Feng Jing, Mei Gao, Bin Xue, Pu Wang, and Ming Zhang. Quantum Algorithms for Solving Linear Regression Equation. *Journal of Physics: Conference Series*, 1738(1):012063, January 2021. ISSN 1742-6588, 1742-6596. doi: 10.1088/1742-6596/1738/1/012063.
- [110] Zhaokai Li, Xiaomei Liu, Nanyang Xu, and Jiangfeng Du. Experimental Realization of a Quantum Support Vector Machine. *Physical Review Letters*, 114(14):140504, April 2015. ISSN 0031-9007, 1079-7114. doi: 10.1103/PhysRevLett.114.140504.
- [111] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning, November 2013.
- [112] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. 2013. doi: 10.48550/ARXIV.1307.0401.
- [113] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2, 2014. ISSN 2296-424X. doi: 10.3389/fphy.2014.00005.
- [114] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2, 2014. ISSN 2296-424X. doi: 10.3389/fphy.2014.00005.
- [115] John Mandel. Use of the Singular Value Decomposition in Regression Analysis. *The American Statistician*, 36(1):15–24, February 1982. ISSN 0003-1305, 1537-2731. doi: 10.1080/00031305.1982.10482771.
- [116] Dastan Maulud and Adnan M. Abdulazeez. A Review on Linear Regression Comprehensive in Machine Learning. *Journal of Applied Science and Technology Trends*, 1(2):140–147, December 2020. ISSN 2708-0757. doi: 10.38094/jastt1457.
- [117] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. 2018. doi: 10.48550/ARXIV.1803.11173.

- [118] Dunja Mladenić. Feature Selection for Dimensionality Reduction. In Craig Saunders, Marko Grobelnik, Steve Gunn, and John Shawe-Taylor, editors, *Subspace, Latent Structure and Feature Selection*, volume 3940, pages 84–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-34137-6 978-3-540-34138-3. doi: 10.1007/11752790_5.
- [119] Nishikanta Mohanty, Bikash K. Behera, and Christopher Ferrie. Solving The Vehicle Routing Problem via Quantum Support Vector Machines, August 2023.
- [120] Nikolaj Moll, Panagiotis Barkoutsos, Lev S Bishop, Jerry M Chow, Andrew Cross, Daniel J Egger, Stefan Filipp, Andreas Fuhrer, Jay M Gambetta, Marc Ganzhorn, Abhinav Kandala, Antonio Mezzacapo, Peter Müller, Walter Riess, Gian Salis, John Smolin, Ivano Tavernelli, and Kristan Temme. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, 3(3):030503, July 2018. ISSN 2058-9565. doi: 10.1088/2058-9565/aab822.
- [121] M. S. Morshed, S. Meeran, and A. Jain. A State-of-the-art Review of Job-Shop Scheduling Techniques. *International Journal of Applied Operational Research - An Open Access Journal*, January 2017.
- [122] Samreen Naeem, Aqib Ali, Sania Anam, and Muhammad Munawar Ahmed. An Unsupervised Machine Learning Algorithms: Comprehensive Review. *International Journal of Computing and Digital Systems*, 13(1):911–921, April 2023. ISSN 2210-142X. doi: 10.12785/ijcds/130172.
- [123] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines.
- [124] Khadijeh Najafi, Susanne F. Yelin, and Xun Gao. The Development of Quantum Machine Learning. *Harvard Data Science Review*, January 2022. doi: 10.1162/99608f92.5a9fd72c.
- [125] Filippo Orazi, Simone Gasperini, Stefano Lodi, and Claudio Sartori. Hybrid Quantum Technologies for Quantum Support Vector Machines. *Information*, 15(2):72, January 2024. ISSN 2078-2489. doi: 10.3390/info15020072.

- [126] Mandaar B. Pande. A Comprehensive Review of Data Encoding Techniques for Quantum Machine Learning Problems. In *2024 Second International Conference on Emerging Trends in Information Technology and Engineering (ICETITE)*, pages 1–7, Vellore, India, February 2024. IEEE. ISBN 979-8-3503-2820-2. doi: 10.1109/ic-ETITE58242.2024.10493306.
- [127] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O’Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1):4213, July 2014. ISSN 2041-1723. doi: 10.1038/ncomms5213.
- [128] Niklas Pirnay, Vincent Ulitzsch, Frederik Wilde, Jens Eisert, and Jean-Pierre Seifert. An in-principle super-polynomial quantum advantage for approximating combinatorial optimization problems via computational learning theory. 2022. doi: 10.48550/ARXIV.2212.08678.
- [129] Christophe Piveteau and David Sutter. Circuit knitting with classical communication. *IEEE Transactions on Information Theory*, 70(4): 2734–2745, April 2024. ISSN 0018-9448, 1557-9654. doi: 10.1109/TIT.2023.3310797.
- [130] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018. ISSN 2521-327X. doi: 10.22331/q-2018-08-06-79.
- [131] Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. The Benders Decomposition Algorithm: A Literature Review. *European Journal of Operational Research*, 259:801–817, June 2017. doi: 10.1016/j.ejor.2016.12.005.
- [132] Deepak Ranga, Aryan Rana, Sunil Prajapat, Pankaj Kumar, Kranti Kumar, and Athanasios V. Vasilakos. Quantum Machine Learning: Exploring the Role of Data Encoding Techniques, Challenges, and Future Directions. *Mathematics*, 12(21):3318, October 2024. ISSN 2227-7390. doi: 10.3390/math12213318.
- [133] Minati Rath and Hema Date. Quantum Data Encoding: A Comparative Analysis of Classical-to-Quantum Mapping Techniques and Their Impact on Machine Learning Accuracy, November 2023.

- [134] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical Review Letters*, 113(13):130503, September 2014. ISSN 0031-9007, 1079-7114. doi: 10.1103/PhysRevLett.113.130503.
- [135] Mayra Z. Rodriguez, Cesar H. Comin, Dalcimar Casanova, Odemir M. Bruno, Diego R. Amancio, Luciano Da F. Costa, and Francisco A. Rodrigues. Clustering algorithms: A comparative approach. *PLOS ONE*, 14(1):e0210236, January 2019. ISSN 1932-6203. doi: 10.1371/journal.pone.0210236.
- [136] Troels F. Rønnow, Zhihui Wang, Joshua Job, Sergio Boixo, Sergei V. Isakov, David Wecker, John M. Martinis, Daniel A. Lidar, and Matthias Troyer. Defining and detecting quantum speedup. 2014. doi: 10.48550/ARXIV.1401.2910.
- [137] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6): 386–408, 1958. ISSN 1939-1471, 0033-295X. doi: 10.1037/h0042519.
- [138] Mochammad Choirur Roziqin, Achmad Basuki, and Tri Harsono. A comparison of Montecarlo linear and dynamic polynomial regression in predicting dengue fever case. In *2016 International Conference on Knowledge Creation and Intelligent Computing (KCIC)*, pages 213–218, Manado, Indonesia, November 2016. IEEE. ISBN 978-1-5090-5231-8. doi: 10.1109/KCIC.2016.7883649.
- [139] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323 (6088):533–536, October 1986. ISSN 0028-0836, 1476-4687. doi: 10.1038/323533a0.
- [140] Natasha Sachdeva, Gavin S. Hartnett, Smarak Maity, Samuel Marsh, Yulun Wang, Adam Winick, Ryan Dougherty, Daniel Canuto, You Quan Chong, Michael Hush, Pranav S. Mundada, Christopher D. B. Bentley, Michael J. Biercuk, and Yuval Baum. Quantum optimization using a 127-qubit gate-model IBM quantum computer can outperform quantum annealers for nontrivial binary optimization problems, July 2024.
- [141] Harvey M. Salkin and Cornelis A. De Kluyver. The knapsack problem: A survey. *Naval Research Logistics Quarterly*, 1975.

- [142] Melissa Sánchez, Jorge Cruz-Duarte, José Carlos Ortiz-Bayliss, Hector Ceballos, Hugo Terashima-Marín, and Ivan Amaya. A Systematic Review of Hyper-Heuristics on Combinatorial Optimization Problems. *IEEE Access*, PP, July 2020. doi: 10.1109/ACCESS.2020.3009318.
- [143] Iqbal H. Sarker. Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Computer Science*, 2(3):160, May 2021. ISSN 2662-995X, 2661-8907. doi: 10.1007/s42979-021-00592-x.
- [144] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In Gerhard Goos, Juris Hartmanis, Jan Van Leeuwen, Wulfram Gerstner, Alain Germond, Martin Hasler, and Jean-Daniel Nicoud, editors, *Artificial Neural Networks — ICANN’97*, volume 1327, pages 583–588. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. ISBN 978-3-540-63631-1 978-3-540-69620-9. doi: 10.1007/BFb0020217.
- [145] Maria Schuld and Francesco Petruccione. *Supervised Learning with Quantum Computers*. Quantum Science and Technology. Springer, Cham, 2018. ISBN 978-3-030-07188-2 978-3-319-96423-2.
- [146] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczyński. *Lectures on Stochastic Programming*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, January 2009. ISBN 978-0-89871-687-0. doi: 10.1137/1.9780898718751.
- [147] Koosha Sharifani and Mahyar Amini. Machine Learning and Deep Learning: A Review of Methods and Applications. 10(07), 2023.
- [148] Ruslan Shaydulin, Changhao Li, Shouvanik Chakrabarti, Matthew DeCross, Dylan Herman, Niraj Kumar, Jeffrey Larson, Danylo Lykov, Pierre Minssen, Yue Sun, Yuri Alexeev, Joan M. Dreiling, John P. Gaebler, Thomas M. Gatterman, Justin A. Gerber, Kevin Gilmore, Dan Gresh, Nathan Hewitt, Chandler V. Horst, Shaohan Hu, Jacob Johansen, Mitchell Matheny, Tanner Mengle, Michael Mills, Steven A. Moses, Brian Neyenhuis, Peter Siegfried, Romina Yalovetzky, and Marco Pistoia. Evidence of scaling advantage for the quantum approximate optimization algorithm on a classically intractable problem. *Science Advances*, 10(22):eadm6761, May 2024. ISSN 2375-2548. doi: 10.1126/sciadv.adm6761.
- [149] Marius M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*,

- 35(2):254–265, April 1987. ISSN 0030-364X, 1526-5463. doi: 10.1287/opre.35.2.254.
- [150] Michał Stechły. Introduction to Variational Quantum Algorithms, February 2024.
 - [151] Jill C. Stoltzfus. Logistic Regression: A Brief Primer. *Academic Emergency Medicine*, 18(10):1099–1104, October 2011. ISSN 1069-6563, 1553-2712. doi: 10.1111/j.1553-2712.2011.01185.x.
 - [152] Teppei Suzuki, Takashi Hasebe, and Tsubasa Miyazaki. Quantum support vector machines for classification and regression on a trapped-ion quantum computer. *Quantum Machine Intelligence*, 6(1):31, June 2024. ISSN 2524-4906, 2524-4914. doi: 10.1007/s42484-024-00165-0.
 - [153] Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. 2018. doi: 10.48550/ARXIV.1807.04271.
 - [154] Ewin Tang. Quantum Principal Component Analysis Only Achieves an Exponential Speedup Because of Its State Preparation Assumptions. *Physical Review Letters*, 127(6):060503, August 2021. ISSN 0031-9007, 1079-7114. doi: 10.1103/PhysRevLett.127.060503.
 - [155] Matthias Tschöpe. Beyond SGD: Recent improvements of Gradient Descent Methods. 2019. doi: 10.13140/RG.2.2.34359.88484.
 - [156] Kyriaki A. Tychola, Theofanis Kalampokas, and George A. Papakostas. Quantum Machine Learning—An Overview. *Electronics*, 12(11):2379, May 2023. ISSN 2079-9292. doi: 10.3390/electronics12112379.
 - [157] Vladimir Vapnik, Steven E Golowich, and Alex J Smola. Support Vector Method for Function Approximation, Regression Estimation and Signal Processing.
 - [158] V.N. Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 10(5):988–999, September 1999. ISSN 10459227. doi: 10.1109/72.788640.
 - [159] D. Wallach and B. Goffinet. Mean squared error of prediction as a criterion for evaluating and comparing system models. *Ecological Modelling*, 44(3-4):299–306, January 1989. ISSN 03043800. doi: 10.1016/0304-3800(89)90035-5.
 - [160] C.-C. Joseph Wang and Ryan S. Bennink. Variational quantum regression algorithm with encoded data structure, January 2024.

- [161] Samson Wang, Enrico Fontana, M. Cerezo, Kunal Sharma, Akira Sone, Lukasz Cincio, and Patrick J. Coles. Noise-Induced Barren Plateaus in Variational Quantum Algorithms. 2020. doi: 10.48550/ARXIV.2007.14384.
- [162] Ximing Wang, Chengran Yang, and Mile Gu. Variational Quantum Circuit Decoupling, June 2024.
- [163] Yan Wang, Jungin E. Kim, and Krishnan Suresh. Opportunities and Challenges of Quantum Computing for Engineering Optimization. *Journal of Computing and Information Science in Engineering*, 23(6):060817, December 2023. ISSN 1530-9827, 1944-7078. doi: 10.1115/1.4062969.
- [164] Yunfei Wang and Junyu Liu. A comprehensive review of Quantum Machine Learning: From NISQ to Fault Tolerance. *Reports on Progress in Physics*, 87(11):116402, November 2024. ISSN 0034-4885, 1361-6633. doi: 10.1088/1361-6633/ad7f69.
- [165] John Watrous. *The Theory of Quantum Information*. Cambridge University Press, 1 edition, April 2018. ISBN 978-1-316-84814-2 978-1-107-18056-7. doi: 10.1017/9781316848142.
- [166] Nathan Wiebe, Ashish Kapoor, and Krysta Svore. Quantum Algorithms for Nearest-Neighbor Methods for Supervised and Unsupervised Learning, July 2014.
- [167] Nathan Wiebe, Ashish Kapoor, and Krysta M. Svore. Quantum Deep Learning, May 2015.
- [168] Rui Xie and Wei Wei. *Distributionally Robust Optimization and Its Applications in Power System Energy Storage Sizing*. Springer Nature, Singapore, 2024. ISBN 978-981-97-2565-6 978-981-97-2566-3. doi: 10.1007/978-981-97-2566-3.
- [169] Hayata Yamasaki, Natsuto Isogai, and Mio Murao. Advantage of Quantum Machine Learning from General Computational Advantages, December 2023.
- [170] Shang Yu, Zhian Jia, Aonan Zhang, Ewan Mer, Zhenghao Li, Valerio Crescimanna, Kuan-Cheng Chen, Raj B. Patel, Ian A. Walmsley, and Dagomir Kaszlikowski. Shedding Light on the Future: Exploring Quantum Neural Networks through Optics, 2024.

- [171] Amine Zeguendry, Zahi Jarir, and Mohamed Quafafou. Quantum Machine Learning: A Review and Case Studies. *Entropy*, 25(2):287, February 2023. ISSN 1099-4300. doi: 10.3390/e25020287.
- [172] Harry Zhang. The Optimality of Naive Bayes.
- [173] Renxin Zhao and Shi Wang. A review of Quantum Neural Networks: Methods, Models, Dilemma, September 2021.
- [174] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D. Lukin. Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices. 2018. doi: 10.48550/ARXIV.1812.01041.