TU WIEN Informatics

Hyperparameter Tuning for Quantum Machine Learning

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Daniel Suchan, BSc
Matrikelnummer 11810787

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Ivona Brandić
Mitwirkung: Dr. Vincenzo De Maio

Wien, 31. Juni 2024

Daniel Suchan	Ivona Brandić

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Hyperparameter Tuning for Quantum Machine Learning

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Daniel Suchan, BSc
Registration Number 11810787

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Ivona Brandić
Assistance: Dr. Vincenzo De Maio

Vienna, June 31, 2024

Daniel Suchan	Ivona Brandić

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Daniel Suchan, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 31. Juni 2024

Daniel Suchan

Danksagung

An erste Stelle möchte ich mich bei meinen Betreuern Prof. Ivona Brandić und Vincenzo De Maio für die großartige Möglichkeit bedanken, an diesem inspirierenden und spannenden Thema arbeiten zu können. Auch möchte ich mich für die zuverlässige und stets kompetente Zusammenarbeit über die gesamte Masterarbeit hinweg bedanken. Die hohe fachliche Kompetenz zusammen mit konstruktiven Ratschlägen haben mir maßgeblich bei meiner Arbeit geholfen.

Zudem möchte ich meinen Studienkollegen und Freunden für die unzähligen Erinnerungen danken, die ich mit ihnen während des Studiums sammeln durfte. Ein großes Dankeschön auch an meine Familie und insbesondere an meine Eltern, die während meiner gesamten Studienzeit an mich geglaubt haben und mich stets motiviert haben, mein Bestes zu geben. Abschließend möchte ich mich noch besonders bei Katja für ihr Verständnis und die mentale Unterstützung während schwieriger Phasen meines Studiums bedanken.

Ich bin unglaublich dankbar für diese großartige Unterstützung. Vielen Dank!

Acknowledgements

First and foremost, I want to thank my supervisors Prof. Ivona Brandić and Vincenzo De Maio for the opportunity to work on this inspiring topic as well as their continuous support and guidance throughout the whole thesis. Their expertise and constructive feedback guided me well through my research and supported me during the writing process of this thesis.

I would also like to thank my fellow students and friends for the positive memories I could share with them during my studies. Furthermore, I would like to thank my family and especially my parents for their support and constant belief that encouraged me to always give my best. Finally, I want to give special thanks to Katja for her mental support and understanding during the challenging stages of my studies that gave me confidence in the final success.

I am grateful for all the support. Thank you!

Kurzfassung

Der steigende Bedarf an Rechenleistung für Machine Learning (ML) Anwendungen stellt die Forschung vor die Herausforderung, Technologien zu finden, die diesen Anforderungen gerecht werden. In der derzeitigen Post-Moore Ära ist jedoch davon auszugehen, dass traditionelle von Neumann Computerarchitekturen schon bald an ihre physikalischen Grenzen stoßen und die gewohnten Leistungssteigerungen zukünftig ausbleiben. Da viele Probleme mit Quantenalgorithmen wesentlich effizienter gelöst werden können, haben sich Quantencomputer als vielversprechendes Paradigma für den zukünftigen Bedarf an Rechenleistung erwiesen. Das große Interesse, die Leistungsfähigkeit von Quantencomputern auch zur Beschleunigung von ML Anwendungen zu nutzen, hat Quantum Machine Learning (QML) zum Durchbruch verholfen.

Variational Quantum Algorithms (VQAs) haben sich als der vielversprechendste Ansatz für QML erwiesen, da sie klassische Computer verwenden, um die Limitierungen heutiger Quantencomputer zu minimieren. Der klassische Computer wird von VQAs dazu genutzt, einen Parameterized Quantum Circuit (PQC) zu optimieren, der auf dem Quantencomputer die Lösung berechnet. Die Leistung dieser VQAs ist jedoch sehr von den gewählten Hyperparametern abhängig, welche bereits vor der Ausführung festgelegt werden müssen und äußerst problemabhängig sind. Da es bereits eine große Auswahl an Hyperparametern gibt, ist es sehr zeit- und ressourcenaufwendig diese manuell zu testen. In klassischen ML Anwendungen werden daher automatisierte Lösungen eingesetzt, deren Anwendbarkeit für QML jedoch noch kaum für untersucht wurde. In dieser Arbeit untersuchen wir daher die Anwendbarkeit und Leistungsfähigkeit verschiedener automatisierter Hyperparameter-Tuning-Algorithmen für QML Klassifikationsprobleme.

Unsere Ergebnisse zeigen, dass die richtige Wahl des Hyperparameter-Tuning-Algorithmus einen signifikanten Einfluss auf die Leistung hat und es ermöglicht, nahezu optimale Konfigurationen zu finden. Jedoch zeigen unsere Experimente auch, dass das Barren Plateau Phänomen signifikante Auswirkungen auf die Laufzeit von QML Algorithmen hat und daher stets zu berücksichtigen ist. Insgesamt zeigt diese Arbeit, wie komplex die Suche nach optimalen Hyperparameter-Konfigurationen im Bereich von QML sein kann und bietet eine Orientierung für zukünftige Projekte.

Abstract

In recent years, the computational requirements of modern Machine Learning (ML) applications have increased significantly. The upcoming post-Moore era therefore forces scientists to search for alternative forms of computing that can meet computational demands beyond the capabilities of classical von Neumann architectures. Quantum computing emerged as a very promising paradigm for providing the necessary computational resources, as several quantum algorithms have proven to be more efficient for certain problems. The great interest in exploiting the capabilities of quantum hardware to speed up machine learning applications contributed to the rise of Quantum Machine Learning (QML).

The most promising approach for QML are Variational Quantum Algorithms (VQAs), that combine classical hardware to overcome the limitations of current quantum hardware. Variational Quantum Algorithms (VQAs) use an optimizer on classical hardware to train a Parameterized Quantum Circuit (PQC), that is used to find the quantum state containing the solution to the problem. However, the optimal choice of the optimizer, the structure of the PQC and other hyperparameters is problem-specific and has a major impact on the performance of VQAs. The large number of available options makes manual testing extremely time-consuming and therefore requires automated solutions. In classical ML, automated hyperparameter tuning is widely used, but there are only few studies on its application to QML. In this thesis, we therefore investigate the applicability and performance of different automated hyperparameter tuning algorithms for QML classification tasks.

Our results show that choosing the right hyperparameter tuning algorithm is essential and allows to reliably find near optimal configurations. Nevertheless, we also see that the barren plateau phenomenon significantly impacts the runtime of these algorithms and must be considered in future QML projects. Overall, our results highlight the complexity of hyperparameter tuning for QML applications and provide valuable insights for future projects.

Contents

CHAPTER 1

Introduction

Machine Learning (ML) has advanced rapidly in the last years and the availability of large datasets led to very powerful applications that are able to learn and detect complex patterns, e.g., natural language processing or computer vision [BWP+17, SS18]. However, in the upcoming post-Moore era, classical von Neumann architectures will not be able to scale with the increasing computing demands of machine learning applications due to physical limitations [DMAB22, OM20]. Quantum computing has gained huge popularity as a computational paradigm that is able to overcome the limitations of classical architectures [OM20, GKS+22]. Several quantum algorithms have been proven to solve certain problems more efficiently than classical algorithms and provide impressive speedups, e.g., Shor's factoring algorithm [OM20, GKS+22, CDMB+22]. Quantum algorithms are therefore being explored in a wide range of research areas including drug design, finance, secure communications and others. [BWP+17, GKS+22].

Quantum Machine Learning (QML) has emerged as a very promising approach to address the requirements of future ML workloads by exploiting the capabilities of quantum hardware [GKS+22, SAGB22, BWP+17]. Moreover, QML is known to handle counterintuitive statistical patterns very well and could improve their detection over classical computers [BWP+17]. QML therefore gained popularity and evolved into a very active and promising area of research that is believed to speed up ML tasks in the future [GKS+22, SAGB22, BWP+17].

To overcome the limited capabilities of recent quantum hardware, Variational Quantum Algorithms (VQAs) have emerged as the most promising approach for QML [CAB+21]. VQAs combine quantum and classical hardware to exploit the capabilities of quantum computing while minimizing the error rate through the use of classical hardware [CAB+21]. In VQAs, the classical hardware is used to optimize variational parameters of so called Parameterized Quantum Circuit (PQC) [CAB+21]. PQCs are a tunable way to explore multiple quantum states until the final solution state of the problem is reached. The optimizer iteratively tunes the parameters of a PQC after each execution of the quantum

circuit until a cost function is satisfied and an approximation of the optimal solution can be returned [MDTD23, CAB+21]. Generally, VQAs can be seen as the quantum equivalent of neural networks that only execute the relevant part of a problem on the quantum hardware and make use of classical optimization techniques [MDTD23, CAB+21].

Since VQAs are considered as the most promising strategy to utilize the power of current quantum hardware, they have already been proposed for many applications such as quantum chemistry, quantum meteorology or ML [CAB+21]. It has been shown that the performance of VQAs heavily depends on the selected hyperparameters, i.e., the used PQC, optimizer or cost function [HDMB24]. However, the hyperparameters of VQAs are highly problem dependent, making it hard to select them a priori [BMWV+23]. Moreover, there are already a lot of different optimizers and PQCs architectures available, making it extremely costly and time-consuming to exhaustively test them all. Nevertheless, the correct hyperparameter selection is crucial for efficiently executingVQAs and a necessity in any project.

In classical ML, hyperparameter tuning or Hyperparameter Optimization (HPO) is a common technique and describes the process of finding the optimal hyperparameters for a specific problem. Consequently, there already exist a lot of different algorithms that automatically search for optimal hyperparameters of a model during its development [YZ20]. These algorithms reduce the time to find well suited hyperparameters, make the choice more reproducible, and often results in more efficient and accurate model training [YZ20]. For classical ML, there are even approaches called Automated machine learning (AutoML) that automate all steps of applying an ML to an actual dataset. For QML on the other hand, there is currently only little knowledge on how to find the optimal hyperparameters [MPD+23]. This emphasizes the need for research in this direction, making hyperparameter tuning for QML an important open problem that needs to be investigated.

1.1 Aim of the Thesis

The aim of the thesis is to investigate the applicability and performance of hyperparameter tuning algorithms for QML classification tasks. The results of this work contribute to the problem described in the previous section and allow determining whether automated hyperparameter tuning of VQAs can be used to improve future QML workflows.

Manually selecting hyperparameters based on knowledge and experience is still a common approach [SKK+23]. However, manually or exhaustively searching for optimal hyperparameters has major drawbacks as it naturally limits the search space, is not reproducible and is very time-consuming due to the increased computational cost of QML [SKK+23, MPD+23]. A comprehensive evaluation of available automated hyperparameter tuning algorithms in terms of model accuracy and time consumption would help researchers to select an appropriate approach to reduce the time and cost of training QML models. To this extent, this thesis will answer the following research questions:

RQ 1 Which hyperparameters are most relevant for Quantum Machine Learning models?

Hyperparameters are the configuration parameters of a VQA such as the optimizer or the used PQC. Certain components of VQAs are expected to have a greater impact on model performance than others. Due to the computational expense of hyperparameter tuning, only a suitable selection of hyperparameters will be used for the experiment.

RQ 2 Which existing methods can be used for hyperparameter tuning of Quantum Machine Learning models?

For classical ML tasks, numerous automated hyperparameter tuning techniques are available in the literature [YZ20]. These existing methods need to be reviewed to evaluate whether they are suitable for typical QML problems or not.

RQ 3 How does automatic hyperparameter tuning perform compared to exhaustive search in terms of time and accuracy?

To establish a baseline for the performance benchmark, an exhaustive search over all available hyperparameters in the domain space will be conducted. The performance of automated hyperparameter tuning algorithms will then be compared to the results of the exhaustive search. As performance metrics, we use the accuracy of the best hyperparameter configuration and the running time of the hyperparameter tuning algorithm.

1.2 Contribution

In the following, we summarize the key contributions of this work:

- By reviewing existing literature, we identified the most relevant hyperparameters of VQAs for QML. We found multiple hyperparameters that are crucial for the performance of VQAs and need to be carefully selected for the best performance. Our findings therefore allow prioritizing these hyperparameters in future QML projects and form the theoretical foundation of applying hyperparameter tuning algorithms to VQAs.

- We systematically studied available hyperparameter tuning algorithms for their applicability to QML problems and assessed their suitability. We identified several promising tuning approaches, such as Random Search (RS) or variants of Bayesian optimization algorithms. However, we also explain why some approaches are not suited for typical VQAs projects. This thesis therefore provides valuable insights into suitable hyperparameter tuning algorithms for QML problems, and can help future researchers and experts in selecting appropriate tuning approaches.

- We adapted the most promising hyperparameter tuning algorithms to support QML classification tasks and evaluate their performance regarding accuracy and time. Our benchmark shows that some hyperparameter tuning approaches are able to consistently outperform the others across multiple dataset in terms of accuracy. The results of the runtime evaluation also indicate that certain approaches are more effective than others. However, we also see that some evaluation runs are heavily affected by the barren plateau phenomenon, which demonstrates that hyperparameter tuning for QML is not a trivial problem.

  In this thesis, we also provide a comprehensive analysis of multiple hyperparameter tuning algorithms regarding their parallelization capabilities and their ability to handle very large search spaces. Additionally, we evaluate the performance of the individual hyperparameters and show that the optimal hyperparameters are highly problem dependent and vary between different datasets.

1.3 Structure

The thesis is structured as follows. In Chapter 2 we elaborate the background of quantum computing and introduce the basics of QML and hyperparameter tuning that are needed throughout the thesis. We then review some of the most important related work before we explain the fundamental building blocks of VQAs in Chapter 3. In this chapter, we also highlight the challenges and limitations of VQAs before we finally describe the most important hyperparameters and answer RQ1. In Chapter 4 we review a variety of existing algorithms to evaluate their suitability for QML projects and answer RQ2. Chapter 5 then introduces the methodology of our benchmarks, including the selected tuning algorithms, datasets and the hyperparameter search space. The results of our experiments are then evaluated and discussed extensively in Chapter 6. Finally, in Chapter 7 we conclude the thesis and outline some possible future work.

CHAPTER 2

Background

This chapter introduces the basics that are needed throughout the thesis by providing the necessary background knowledge and presenting the most important related work at the end. First, we discuss the general concept of quantum computing and explain the most fundamental quantum mechanics principles. We then cover some limitations of current quantum computers before introducing QML and its promising capabilities. Furthermore, we introduce the basics of hyperparameter tuning and discuss its advantages that make it an essential tool for all state-of-the-art ML models. Finally, we give an overview of the most important related work in the field of QML and hyperparameter tuning for QML.

2.1 Quantum Computing

The first theoretical concepts of quantum computing were already introduced in 1982 by Richard Feynman as an idea to simulate quantum physics using a computing device based on quantum mechanics [GKS+22]. Controlling the underlying quantum mechanical processes to build a quantum computer poses many technical challenges and was therefore not feasible for a long period [GKS+22]. However, in recent years, significant progress has been made, and quantum computing has evolved to a very active area of research for both academic and industrial scientists [OM20]. Nearly all major tech companies and many new startups are competing to build the most powerful yet error-free quantum computer, leading to tremendous advances in both hardware and software development [GKS+22, OM20]. The great global interest in developing a universal quantum computer is driven by the assumption that classical Von-Neumann computing architectures reach their physical limits soon [CDMB+22, OM20]. Since electrical transistors experience problems when they are further reduces in size, it is uncertain whether these classical architectures can scale with future computing demands [OM20]. Consequently, research on alternative computing architectures becomes increasingly important [CDMB+22, OM20].

Quantum computers are based on quantum bits (qubits), which utilize some fundamental quantum mechanical principles for their calculations [OM20]. The fact that they are not based on classical transistors makes quantum computing a promising architecture for the post-Moore era that has the potential to surpass the capabilities of classical computers [OM20, CDMB+22]. For certain problems, there are already quantum algorithms that are much more efficient than their classical variant. One of the first and probably the most prominent example is Shor's prime factorization algorithm [Sho97]. It provides an almost exponential speedup for factoring large prime numbers and is considered as a possible method for invalidating the Rivest–Shamir–Adleman (RSA) crypto system [OM20]. Shor's prime factorization algorithm attracted much attention when it was proposed in 1994 as it has a huge impact on cybersecurity and further increased the global interest in developing a universal quantum computer [GI19]. Another prominent example of quantum speedup is Grover's search algorithm that can be used to find elements in an unsorted database using only $O(\sqrt{n})$ evaluations, while classical search algorithms need $O(n)$ evaluations [Gro96, BBBV97]. Since these fundamental quantum algorithms have been proposed, a lot of research has happened and many new quantum algorithms have been developed [Jor]. Another milestone was reached in 2019 when Google demonstrated *quantum supremacy* with one of its quantum computers [GKS+22, AAB+19]. Quantum supremacy describes the computational advantage of quantum computers to solve a certain problem with super-polynomial speedup that would otherwise be intractable for classical computers [Pre12]. However, the problem used to demonstrate quantum supremacy was specifically chosen for this purpose, and it remains to be shown that quantum computers can solve real-world problems better than classical computers [GKS+22].

Although, the advances of today's quantum computers are very promising, there are still many open challenges and limitations that need to be solved before quantum computers can be used for real-world problems [GKS+22]. Therefore, current quantum hardware is referred to as Noisy Intermediate-Scale Quantum technology due to the noisy and error-prone operation [Pre18, Bro19]. To better understand these limitations, we introduce the fundamental quantum mechanical principles that are needed for the remaining chapters in the following, with more detailed explanations being available in the respective references.

2.1.1 Superposition

The basic unit of information for quantum computers is a qubit. In contrast to classical binary bits that either represent the value '0' or '1', a qubit can additionally be in a *superposition* of both values simultaneously[CDMB+22]. The state of a qubit can mathematically be represented as follows [WBLS22]:

$$a|0\rangle + b|1\rangle \text{ with } a, b \in \mathbb{C} \text{ and } |a|^2 + |b|^2 = 1 \tag{2.1}$$

The two orthogonal basis vectors $|0\rangle$ and $|1\rangle$ are called the *computational basis* and represent a two-dimensional vector space [WBLS22]. The complex coefficients a and

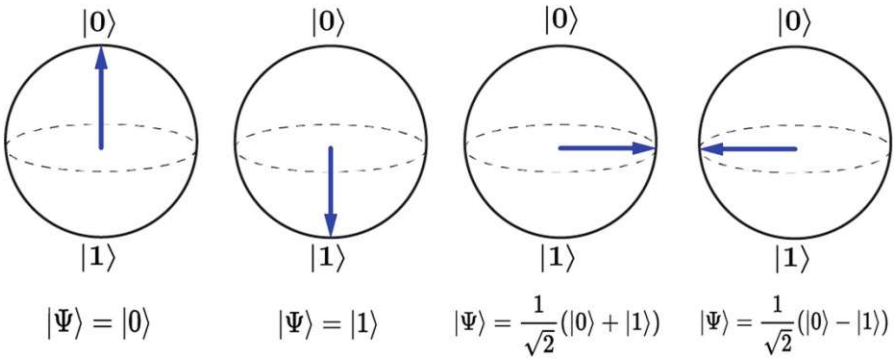


Figure 2.1: Different states of a qubit visualized in a Bloch sphere [HIP+21].

b are called *amplitudes* and can be interpreted as probabilities of the qubit being in either of the states $|0\rangle$ or $|1\rangle$ after measuring it [GKS+22, CDMB+22, WBLS22]. This means that with a probability of $|a|^2$ the qubit is in the $|0\rangle$ state and with a probability of $|b|^2$ in the $|1\rangle$ state [WBLS22]. As already stated in the equation above, the sum of these probabilities must be equal to 1 since there are only this two possible states that can be measured [WBLS22]. However, if neither a nor b is zero i.e., $a, b \neq 0$, the qubit is in *superposition* and the state is a linear combination of $|0\rangle$ and $|1\rangle$ [WBLS22]. Consequently, the *superposition* of a qubit allows various different linear combinations and opens up a very large computational space that can be used to solve complex problems very efficiently [GKS+22].

In Figure 2.1 four possible states of a single qubit are visualized using the Bloch sphere representation. The first two examples represent the measurable basis vectors $|0\rangle$ and $|1\rangle$ with the arrows showing straight up or down respectively. However, each other possible point in the Bloch sphere represents a *superposition* of the qubit [HIP+21]. As an example, the third state in Figure 2.1 shows the qubit being in the *superposition* $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ with the amplitudes being $a = b = \frac{1}{\sqrt{2}}$. The probability of measuring either of the two states $|0\rangle$ or $|1\rangle$ is $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$.

The Bloch sphere representation visualizes the large computational space that the *superposition* of a single qubit provides [GKS+22]. In a quantum computer, multiple qubits are combined to a *quantum register* whose state is a linear combination of the individual qubits [DMAB22]. Consequently, a *quantum register* of n qubits can process 2^n values simultaneously, which is called *quantum parallelism* [DMAB22]. By adding a single qubit, the computational space of a quantum computer can be doubled, while for a classical computer the number of bits needs to be doubled to achieve the same [GKS+22]. This exponential increase in computational space makes quantum computing a very promising approach for large problems such as drug design, molecular dynamics or ML [DMAB22, GKS+22]. However, more research is needed to find ways that allow the efficient encoding and loading of such big problem instances into a quantum state to fully exploit this advantage [GKS+22]. Moreover, the result of a quantum computation

is probabilistic and needs to be repeated multiple times to obtain the result, which negatively impacts the overall performance [CDMB+22].

2.1.2 Entanglement

In addition to *superposition*, quantum computers use another quantum mechanical principle called *entanglement*. It describes the phenomenon that quantum particles can have a special connection that lets them interact with each other regardless of their physical distance [OM20, KRK20]. This allows qubits in an entangled state to share their properties instantaneously over large physical distances [KRK20]. When the state of an entangled qubit is changed, e.g., by a measurement or manipulation, the properties of the remaining qubits of this entangled state also change without any delay [GKS+22]. This is sometimes referred to as quantum teleportation, meaning that an unknown quantum state is transferred over large distances without any delay [HIP+21].

An illustrative example for the behavior of quantum entanglement is given by Hughes *et al.* in [HIP+21]. They consider a fair coin toss with two coins that normally would have the possible outcomes of HH, HT, TH or TT each occurring with a probability of 25%. However, the probability of the possible outcomes can be changed by bringing the coins into an entangled state. A well-known entangled state is the so-called Bell state that provides a probability of 50% for two measurable states. Generally, there are four different Bell states that are possible for two qubits. In the example of two coins, all pairs of two different results could be used to generate a Bell state e.g, HH-HT or HH-TH. Hughes *et al.* used the following Bell state as an illustrative example [HIP+21]:

$$\frac{|HH\rangle + |TT\rangle}{\sqrt{2}} \tag{2.2}$$

This Bell state guarantees that after the entangled coin flip the only possible states are either HH or TT with a probability of 50% for each. Furthermore, entanglement allows separating the coins over a large physical distance and measure only one of them. By the characteristics of the above Bell state, the outcome of the other distant coin must be identical to the measured one, i.e., if head is measured on one coin, the other must be head too. This behavior happens instantaneously because the quantum particles share non-classical information already during the process of entanglement and no information is being transmitted during measurement [HIP+21]. Quantum entanglement is therefore a key advantage over classical computers and one of the reasons why some computational problems can be solved much faster [HIP+21].

2.1.3 Computation

To take advantage of the *superposition* and *entanglement* principles during a quantum computation, many algorithms share a common structure [GKS+22]. First, a qubit is initialized to be in superposition with equal probabilities for all possible results [GKS+22]. Then, quantum operations based on entanglement and superposition principles are used

to generate a quantum state with high probabilities for the desired outcome [GKS+22]. Finally, the measurement causes the qubits to collapse into a measurable state with the previously set probabilities [GKS+22].

In the following, some of the most important limitations of quantum computers are introduced.

2.1.4 Limitations

Although quantum computers are very promising and efficient quantum algorithms have been proposed, there are still major limitations that need to be considered in current implementations. In the following, we introduce some of the main limitations that current quantum computers are facing.

Noise

A major limitation of current quantum computing is the available hardware. Quantum computers and especially the underlying qubits are expected to be reliable and error free to execute quantum algorithms efficiently [BCLK+22]. However, the quantum mechanical processes inside a qubit make them very fragile and vulnerable to effects from the surrounding environment. Environmental factors like changes in temperature, electromagnetic fields or energy dissipation can alter the state of a qubit and lead to incorrect signals known as *noise* [HIP+21]. Consequently, qubits need to be isolated from the environment to avoid that quantum *decoherence* destroys the encoded information [OM20]. Depending on the technology used, some qubits only have a very short coherence time, causing the encoded information to become invalid after a few nanoseconds [GKS+22]. Moreover, the engineering process is extremely challenging, since minimal defects in the used materials or components can introduce errors [GKS+22]. Effectively isolating qubit from the environment while interacting with them during the execution of quantum algorithms is still an open challenge [BCLK+22, GKS+22].

A possible solution to the noise problem is Quantum Error Correction (QEC), which reduces the effects of noise and allows fault-tolerant quantum computing [BCLK+22, CBB+23]. Like error correction protocols on classical computers, the idea is to correct errors immediately after they occur. A simple way of archiving this on classical hardware is to use redundancy to recognize errors on the normally deterministic result e.g., the repetition code [HIP+21, Got09]. On quantum hardware, the state of a qubit cannot be duplicated according to the No-Cloning theorem, since quantum states collapse into a basis state once they are measured [Got09, HIP+21]. An attempt to copy a superposition state would effectively destroy it [HIP+21]. Therefore, more sophisticated QEC protocols have been developed to correct the errors on quantum computers [BCLK+22]. With the threshold theorem [ABO97] such QEC approaches have been theoretical proven to solve the noise problem if the errors on a quantum system can be reduced below a certain threshold [CBB+23].

However, developing hardware with noise levels below these required thresholds is very challenging [CBB+23]. Moreover, these protocols introduce a high qubit overhead, requiring millions of qubits to effectively correct errors and enable fault-tolerant quantum computing for industrial-sized applications [BCLK+22, CBB+23]. Given that current quantum hardware provides on the order of 1000 physical qubits, this overhead makes it impossible to employ such QEC protocols [BCLK+22, Cas23]. Additionally, these qubits need to be tightly connected and integrated to provide the required performance [GKS+22]. Consequently, current hardware is sometimes referred to as Noisy Intermediate-Scale Quantum (NISQ) hardware, given their limited qubit count and the imperfect results due to noise [BCLK+22, Pre18].

While fault-tolerant quantum computing will only be available on more advanced hardware in the future, Quantum Error Mitigation (QEM) aims to reduce the effects of noise on current NISQ devices [CBB+23]. In contrast to QEC, the noise on the systems remains unchanged, but the errors are corrected using post-processing [CBB+23]. This implies that QEM can only correct errors for a limited circuit size dependent on the hardware specifications until the noise induced errors are too high to recover the information [CBB+23].

Repeated execution

As introduced in Section 2.1, the goal of quantum algorithms is to manipulate the state of qubits in a way that the probability of the desired result is as high as possible [HIP+21]. One way of accomplishing this is by using quantum gates, which conceptually can be compared to logical gates in classical computers [CDMB+22]. However, rather than being physical components, quantum gates are unitary matrices i.e., they are invertible, and their inverse is equal to their conjugate transpose [HIP+21]. For these gate-based models, it is necessary to repeatedly execute the same quantum circuit to receive a reliable and statistically significant final result [HIP+21]. This introduces an additional overhead and can negatively impact the performance of a quantum experiment.

2.2 Quantum Machine Learning

ML advanced rapidly in the last years and is used for a variety of applications such as image recognition or natural language processing [KRK20]. However, the ever-growing amount of data that is used to further improve these ML models also requires more computational power [KRK20]. As already discussed in Section 2.1, there is a need for alternative architectures since classical computers are not expected to scale with these computational needs in future [KRK20]. QML therefore emerged as a very promising approach with the potential to accelerate at least some of these tasks and evolved into a very active area of research for current NISQ devices [CVH+22, GKS+22, SAGB22].

Generally, QML is a broad term that describes the combination of classical ML theories and quantum algorithms with the goal to accelerate ML tasks by exploiting quantum

Algorithm	Speedup
Least squares fitting [WBL12]	$O(\log n)$
Quantum Boltzmann Machine [AAR+18, KW17]	$O(\log n)$
Quantum Support Vector Machine [RML14]	$O(\log n)$
Bayesian Inference [WG15, LYC14]	$O(\sqrt{n})$
Online Perceptron [KWS16]	$O(\sqrt{n})$
Classical Boltzmann Machine [WKS14]	$O(\sqrt{n})$
Quantum Principal Component Analysis [LMR14]	$O(\sqrt{n})$
Quantum Reinforcement Learning[DTB16]	$O(\sqrt{n})$

Table 2.1: Known speedups of QML algorithms ($O(\log n)$ = exponential, $O(\sqrt{n})$ = quadratic speedup over the respective classical approach) [KRK20].

mechanical principles [CVH+22]. It can be used to process classical binary data or directly work on quantum data as it occurs in many physical processes [CVH+22]. QML is therefore often categorized into four main types that distinguish whether the algorithm is executed on a classical or quantum system and whether the data is classical or quantum data [CVH+22]. The great potential of QML stems from various quantum algorithms which are not available on classical platforms and can be used to accelerate typical ML tasks such as clustering [OM20]. Examples are the asymptotically optimal Grover's search algorithm to find elements in unsorted databases or the HHL algorithm for solving linear equation systems [Gro96, BBBV97, OM20, BWP+17]. However, similar to classicalML, there is a wide variety of different quantum algorithms available and many new have been developed to utilize the advantages of currently accessible NISQ devices [GKS+22]. For some QML algorithms, a speedup over their classical counterparts has already been shown in literature. Table 2.1 shows some of these known speedups based on the work of Khan and Robles-Kelly, while the original table can be found in their work [KRK20].

This enables QML to be used in various different applications such as quantum simulations for chemistry [PMS+14] or quantum convolutional neural networks to find QEC schemes [CCL19, CVH+22]. Some other promising applications include meteorology, drug design or material science task as well as classical data analysis tasks [CVH+22]. For classical data analysis tasks, it is assumed that QML is able to detect statistical patterns that are too complex for classical computers due to the quantum mechanical principles they are based on [BWP+17, KRK20]. Furthermore, quantum superposition allows evaluating multiple states simultaneously, which is beneficial for data intensive tasks [KRK20]. However, instead of replacing classical computers, quantum computers will most likely be used as a complementary technology [KRK20]. Similar to Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs), quantum computers could be used as accelerators to run more complex ML models for suitable problems [KRK20].

2.3 Hyperparameter Tuning

ML is already being used in many applications and provides great improvements for computer vision, speech processing or natural language processing [HKV19, YS20]. However, ML models need to be individually configured for each of these applications to provide the best performance [HKV19]. Generally, there are two types of parameters that need to be distinguished for ML models. On the one hand, there are model parameters that are intrinsic to the ML model and get updated throughout the learning process after they are initialized at the beginning [YS20]. On the other hand, there are so-called *hyperparameters* which need to be selected before the learning process, since they define the architecture of the ML model [YS20]. Examples of model parameters are the internal neuron weights of a network, while hyperparameters could be the learning rate or the optimizer type [YS20].

The process of selecting the optimal hyperparameters is not trivial and requires experts to test several configurations until an acceptable configuration is found [HKV19]. This has inspired the research on automated hyperparameter tuning approaches to accelerate the process of finding suitable hyperparameters for any application [HKV19]. It is sometimes also referred to as HPO and will be used interchangeably in this thesis. Nevertheless, hyperparameter tuning is considered a hard problem with many challenges during its practical use [HKV19]. Depending on the size and complexity of a chosen model and dataset, the evaluations can be extremely time and resource consuming [HKV19]. Moreover, knowing which hyperparameter to tune and what value ranges to use can be difficult and is often not known in advance [HKV19]. In addition to this uncertainty, the typically large number of hyperparameters with different types result in a very complex search space that is hard to deal with [HKV19].

Automated hyperparameter tuning approaches have therefore been extensively studied in academia for years and are now a well-known technique for classical ML tasks [HKV19, YZ20]. Although they cannot solve all the aforementioned challenges, the usage of automated algorithms to search for optimal hyperparameters has multiple benefits [HKV19]:

- It minimizes the knowledge barrier for researchers to apply hyperparameter tuning, since it does not require in-depth knowledge for all the needed hyperparameters [HKV19].
- It generally lowers the human effort during the optimization process, especially for multiple hyperparameters with complex and time-consuming model evaluations [HKV19, YS20]. After an initial configuration, hyperparameter tuning algorithms do not require any user interaction and simply terminate if their budget is exhausted [YS20]. This reduces the time researchers need to actively spend on hyperparameter tuning [YS20].
- Using automated hyperparameter tuning techniques improves the performance of the respective ML model by adjusting the hyperparameters specifically for an

individual problem [HKV19]. Its usage has improved the overall performance of common ML models in various benchmarks [HKV19]. Moreover, it allows scaling the hyperparameter tuning effort by reusing a tuning pipeline for a specific model on different datasets to find the individual optimums for each dataset [YS20].

- The usage of automated hyperparameter tuning algorithms makes the performance of ML models more reproducible and comparable [HKV19]. In contrast to manual hyperparameter tuning, the used budget and resources can be better measured and properly adjusted to allow a fair comparison [HKV19]. Furthermore, this allows to identify the most effective ML model for a certain problem by using the same tuning approach and budget for different ML models [YS20].

To take advantage of these benefits, many hyperparameter tuning algorithms exist today. However, each of these approaches has different advantages and disadvantages, making it crucial to select an appropriate approach based on the specific problem and configuration space [YS20]. We therefore review various existing algorithms in Chapter 4 to evaluate their suitability for QML projects.

2.4 Related Work

As quantum hardware becomes more accessible to researchers, the number of feasible applications is increasing and QML is gaining popularity. In order to build fast and efficient applications, hyperparameter tuning is an important aspect during their development. Although hyperparameter tuning for QML is a relatively new topic, results of available literature are promising and emphasize the importance of research in this direction. In the following, we introduce the most relevant existing work for this thesis. We start by discussing research that covers the fundamentals of QML, VQAs and hyperparameter tuning as well as some promising applications before introducing work with a focus on hyperparameter tuning for QML.

A comprehensive overview over the most fundamental algorithms used for QML is given by Biamonte *et al.* [BWP+17]. They discuss why QML is a promising approach for ML problems and highlight what is needed to outperform classical techniques in the future. Furthermore, they discuss different algorithmic approaches to QML and provide an overview of methods for which quantum speedups have already been found.

Cerezo *et al.* provide a more recent review over the current methods of QML and focus more on the challenges and opportunities in their work [CVH+22]. They give an extensive overview of promising applications, with a focus on quantum neural networks and quantum deep learning. Moreover, they extensively discuss the challenges that current QML methods face, including architecture design, noise and barren plateaus.

A promising application of QML is demonstrated by Cong, Choi and Lukin in [CCL19] where they propose a new quantum convolutional neural network. They use the quantum

convolutional neural network to efficiently find QEC codes. The resulting QEC scheme is able to significantly outperform known quantum codes with comparable complexity.

Another promising application of QML is quantum-inspired hyperparameter optimization, as presented by Sagingalieva *et al.* in [SKK+23]. They benchmark their approach with a car image classification problem and were able to reduce the expected runtimes and fitness compared to other existing approaches. Additionally, they present a hybrid quantum neural network model that combines a classical ResNet model with a quantum circuit. This hybrid model is able to outperform comparable classical ML models and demonstrates a promising practical application for QML methods.

The hyperparameters of quantum neural networks, a special type of QML, are studied by Moussa *et al.* in [MPD+23]. In their work, they analyze the performance impact of different hyperparameters for classification problems. During the benchmark, they use seven small open source datasets with less than 20 features to not exceed a 20 qubit budget and allow for efficient benchmarking. Their results show that the performance impact varies considerable between the different hyperparameters. The learning rate of the optimizer could be identified as the most important hyperparameter, followed by the depth and date encoding strategy, while the entanglement was least important. However, this study focuses on specific quantum neural network models and is limited to a specific optimizer. The experiments are based on the Cirq [Dev24] framework and a specific statistical model.

Gomez *et al.* [BGOC+22] proposed a cloud-based framework to automatically search for optimal circuit architectures and other hyperparameters. The proposed framework is demonstrated with a quantum generative adversarial neural network for energy prices. However, their framework is currently optimized to support generative adversarial neural networks and supports the respective hyperparameters. Moreover, the framework currently uses a brute force approach for model selection, while more advanced approaches such as Bayesian optimization are future work.

Taking advantage of QML is challenging on current quantum hardware. Bharti *et al.* [BCLK+22] therefore provide a great overview of state-of-the-art approaches that can be used on today's quantum hardware for various different applications. Their extensive review of current algorithms and paradigms shows that most of the current approaches utilize hybrid architectures based on classical and quantum hardware, which are referred to as VQAs. After introducing the fundamentals of VQAs, they discuss challenges, future applications and provide some insights into benchmarking.

Similarly, the work of Cerezo *et al.* [CAB+21] reviews the most important aspects of VQAs and outlines why they are considered as the most promising approach to take advantage of current quantum hardware. They also introduce the basic concepts and building blocks of VQAs and review the most promising applications such as quantum simulations or QML. Furthermore, they extensively discuss the challenges that affect the trainability, efficiency, and accuracy of VQAs, including some proposed solutions. Finally, they describe how VQAs could improve the performance of ML applications and

highlight recent advances in quantum neural networks.

Although VQAs are a very promising concept for QML, hyperparameter tuning is a crucial aspect for archiving the best performance. A very comprehensive and detailed overview of hyperparameter tuning approaches for classical ML is given by Hutter *et al.* in [HKV19]. They cover the fundamental tuning techniques and review well-known systems that can be used to improve the performance of ML models. Furthermore, the work covers some well known competitions where teams challenge each other to find the most performant ML model by applying different hyperparameter tuning approaches.

The work of Yang and Shami [YS20] also gives a great overview of existing hyperparameter tuning algorithms for classical ML models. They cover a range of different optimization algorithms and discuss frameworks and libraries to apply them. Additionally, they provide valuable insights into which techniques are best suited for different search spaces and hyperparameter types. These insights allow assessing the suitability of these optimization algorithms for different problems and help researcher to choose an appropriate approach. Furthermore, this work provides practical results from experiments conducted on benchmarking datasets to compare the performance of various hyperparameter tuning algorithms.

Bonet-Monroig *et al.* [BMWV$^+$23] study the performance of VQAs by focusing on the optimizer, which is one of the basic building blocks of VQAs. They chose four different optimizers and compare their performance on some small chemistry and material science problems. They additionally used hyperparameter tuning on two of these optimizers for a large range of models and were able to significantly improve the performance for one of them. Their results demonstrate the importance of hyperparameter tuning for VQAs and show that the configurations of VQAs are highly problem dependent. However, their work is limited to only one component of VQAs, while other possible hyperparameters are left to their defaults.

The PQC which is another component of VQAs is studied by Du *et al.* in [DHY$^+$22]. In their work, they propose a quantum architecture search scheme to automatically find a suitable circuit architecture for a given problem. The goal is to find a good balance between expressively and noise sensitivity by selecting a not too large but still expressive PQC. Their proposed scheme is benchmarked with data classification and quantum chemistry tasks. The results show that for the considered problems, the automated architecture search was able to outperform default PQCs while minimizing the effects of noise and barren plateaus. However, the other components of VQAs are not considered and presumably left to their defaults.

Another basic building block of VQAs is called feature map and studied by Hossain *et al.* in [HAS$^+$21]. The feature map is a way to encode the input data into quantum states and necessary for all QML tasks. In the study, the authors use a Quantum Support Vector Machines (QSVMs) for data classification tasks to assess the performance of different feature maps based on the resulting accuracy and time consumption. In the benchmark, they compare different feature map types with varying depths to find the best

configuration for four different datasets. Their results show that the time consumption increases linearly with higher depths, and that the best feature map selection varies across different datasets. Although the used feature maps are also commonly used in VQAs, the results may vary. Furthermore, the study is again limited to only a single hyperparameter that is tuned.

Cranganore *et al.* studied the performance of VQAs by comparing different PQCs and optimizers for molecular dynamics simulation. For the benchmark, they used Qiskit [Qis23a] to test different PQCs with various width, repetition, and entanglement configurations as well as different optimizers. Their results show that for the tested applications, the ansatz has a higher impact on the overall performance than the optimizer. Moreover, it can be seen that the overall performance could be significantly improved by hyperparameter tuning for the individual applications. Although the study already covers two relevant hyperparameters, there are other components such as the feature map that are left to their defaults. Moreover, they use exhaustive search to determine the best hyperparameter configuration instead of more sophisticated approaches.

Overall it can be seen that the field of QML and in particular hyperparameter tuning for QML is advancing rapidly. Although a lot of work has already been done, it is still uncertain which hyperparameter tuning approach works best for QML and in particular VQAs since comprehensive evaluations are missing. Existing papers either focus on individual hyperparameters or use very basic hyperparameter tuning algorithms. Moreover, it is difficult to compare the results of different papers since no standardized benchmark is established so far. We therefore try to fill this gap with our thesis by providing a comprehensive hyperparameter tuning study for VQAs. The hyperparameters considered in this thesis include the most important components of a VQA rather than just a selection thereof. Additionally, we provide an in depth review of the most common hyperparameter tuning algorithms and their suitability for QML and evaluate the performance of the most promising approaches by applying them to well-known classification problems.

CHAPTER 3

Variational Quantum Algorithms

As already mentioned in Section 2.1.4, current NISQ hardware has various constraints such as the limited number of available qubits or noise that leads to errors and limits the circuit depth [CAB+21]. To overcome these limitations, quantum hardware is combined with classical computers to operate as a hybrid quantum system [DMAB22, CDMB+22]. A problem then gets split into a quantum part that is processed on the Quantum Processing Unit (QPU), while the remaining part of the problem is executed on a classical computing architecture [BCLK+22]. This allows to utilize the advantages of quantum computers for the complex portions of an algorithm while maintaining low error rates and small circuits.

VQAs have emerged as the most promising approach to exploit the power of such hybrid systems [CDMB+22, CAB+21, BMWV+23]. In the following, the basic concepts and fundamental building blocks of VQAs are introduced and described. Furthermore, we highlight the most important challenges and limitations that that need to be considered. Finally, we answer our first research question by reviewing available literature to determine the most important hyperparameters of VQAs.

3.1 Basic Concepts

VQAs are special algorithms that operate on classical and quantum computers simultaneously to overcome the constraints of current quantum hardware [CAB+21]. Until fault-tolerant quantum computers are available, VQAs are considered as the most promising approach to utilize the power of quantum devices [CAB+21]. Their modular structure can be adapted for a wide range of problems, and their expressive power even supports universal quantum computing [CAB+21, Bia21]. Example applications are quantum meteorology, simulations, mathematical applications such as factoring or principal component analysis but also the search of ground states in quantum chemistry or molecular dynamics [CAB+21]. Out of the various possible applications, QML is considered to

be one of the most likely areas of research to achieve practical advantage [SAGB22]. Therefore, the focus of the rest of the thesis is on applications related to QML, especially on QML classification problems.

The key concept of VQAs is an optimization-/learning-based approach similar to well-known classical ML methods [CAB+21]. As illustrated in Figure 3.1, the PQC is executed on the quantum computer and an optimizer on the classical computer is used to optimize its parameters repeatedly, forming a hybrid loop [CAB+21]. This process allows overcoming the limitations of current NISQ devices [CAB+21]. This structure can be adapted for various applications, resulting in different types of VQAs. Variational Quantum Eigensolver (VQE) for example, is considered as one of the first examples that is specifically designed to solve quantum chemistry problems [BCLK+22]. Another early variant is called Quantum Approximate Optimization Algorithm (QAOA), which is used to solve combinatorial optimization problems [BCLK+22].

Figure 3.1 also contains the basic building blocks of VQAs which are described in more detail in the following sections.

3.1.1 Problem Description

The first necessity to use VQAs for QML is a proper problem description and a dataset to train the PQC [CAB+21]. Similar to classical ML, such datasets contain instances of the problem that need to be solved e.g., images for classification, weather data for prediction or quantum data for simulation. To assess the performance of the resulting VQA, a subset of the given data is used for training the PQC and the remaining data is used to measure the performance, just like for classical ML models.

The cost function encodes the problem into a measurable function [CAB+21]. Similar to classical ML application, the cost function is used to track the performance of intermediate states with the ultimate goal of finding the global minima [CAB+21]. A general cost function proposed by Cerezo *et al.* in [CAB+21] is given in the following:

$$C(\theta) = f(\{\rho_k\}, \{O_k\}, U(\theta)) \tag{3.1}$$

In the above equation, θ are the variational optimizable discrete and continuous parameters, f is some function, $U(\theta)$ stands for a parameterized unitary, $\{\rho_k\}$ is a set of input states and $\{O_k\}$ is a set of observables.

Besides the formal specification, there are also some properties that a cost function should satisfy. The required properties as proposed by Cerezo *et al.* are summarized in Table 3.1 [CAB+21]. Additionally, the quantum circuits used to estimate $C(\theta)$ need to remain shallow and small to comply with the restrictions of current NISQ devices [CAB+21].

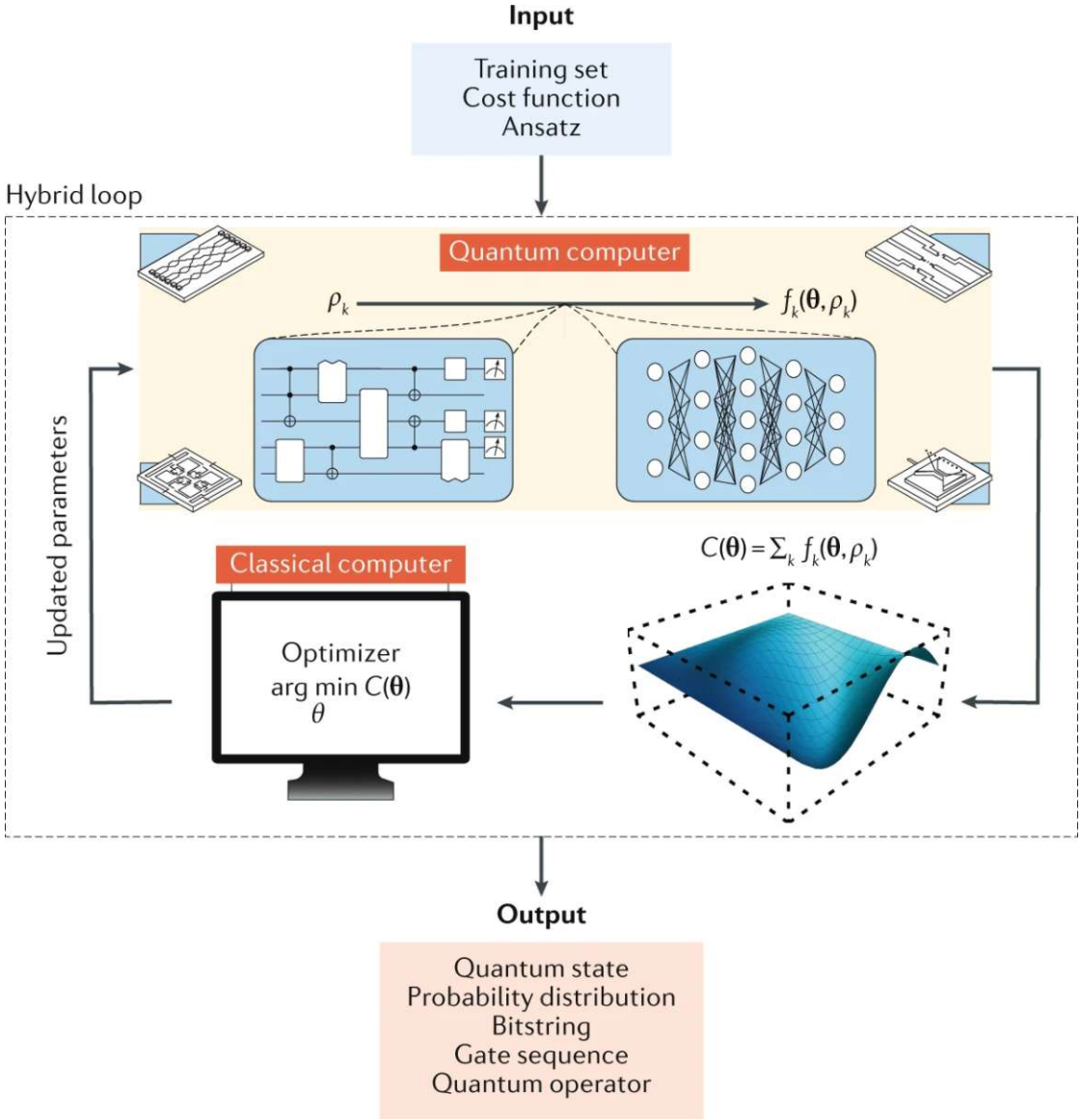


Figure 3.1: Typical architecture of Variational Quantum Algorithms [CAB+21].

Property	Description
Faithful	Problem solution corresponds to minimum of $C(\theta)$
Efficient estimation	$C(\theta)$ is efficiently measurable on a quantum computer (but not on a classical computer)
Meaningful operation	Smaller values of $C(\theta)$ denote better solution
Trainable	Parameters of θ are efficiently optimizable

Table 3.1: Required properties of a cost function [CAB+21].

3.1.2 Feature Map

In classical ML applications, it is common practice to first embed the input data into a higher dimensional feature space before it is further processed [SK19]. The idea is to find a feature space that offers some beneficial properties to make the data analysis more efficient. An example thereof are support vector machines that map the input data to a feature space where the data becomes linearly separable [SK19].

Quantum systems operate in a very similar way. Quantum algorithms require the data to be encoded into quantum states before they can process it [WBLS22, LC20, SK19]. This is an important step since a great portion of possible quantum speedups relies on the entanglement and superposition modification in the exponentially large quantum state space [HCT$^+$19]. The chosen encoding strategy has a significant impact on the runtime of quantum algorithms and therefore needs to be well considered [WBLS22]. Generally, there are various different encoding patterns that can be used to load the data into qubits [WBLS22, LC20]. A well known encoding strategy is called *basis encoding*, which uses the computational basis to encode the data [WBLS22]. The numerical value 2 for example would be transformed into the binary format 10 and then be encoded into the basis state $|10\rangle$ [WBLS22]. Another prominent encoding scheme is *amplitude encoding*, which uses the amplitudes of a quantum state to create a very compact encoding and only needs $\log n$ qubits for an input vector of length n [WBLS22]. More details and additional encoding schemes can be found in the work of LaRose and Coyle [LC20] as well as in the work of Weigold *et al.* [WBLS22].

However, for QML applications the most prominent encoding strategy are quantum *feature maps*, which we will therefore use in the rest of this thesis [SK19]. Feature maps generally use a PQC to map classical data into quantum states [HAS$^+$21, WBLS22]. The encoding process defines a quantum kernel and is comparable to feature encoding in classical ML applications, except that the target feature space is the Hilbert space of the quantum system [SK19]. This analogy described by Schuld and Killoran is a key concept of VQAs for QML, as it allows to generically use currently available quantum hardware to recognize patterns in data [SK19].

Formally, a feature map is a map $\phi : \mathcal{X} \to \mathcal{F}$ that maps an input x_m from a set of input data $\mathcal{X} = \{x_1, ..., x_n\}$ to a feature vector in the Hilbert space $\phi(x_m) \in \mathcal{F}$. To modify the feature map to a quantum feature map, only the target vector $|\phi(x_m)\rangle$ is adapted to a quantum state that lies in the Hilbert space of the quantum system [SK19]. Since a quantum feature map $x \to |\phi(x)\rangle$ is often realized through a PQC, it can also be seen as a state preparation circuit $U_\phi(x)$ that modifies a ground state of the Hilbert space $U_\phi(x)|0...0\rangle = |\phi(x)\rangle$ [SK19, HAS$^+$21]. Some example implementations and their evaluations can be found in the work of Schuld and Killoran [SK19] and Havlicek *et al.* [HCT$^+$19].

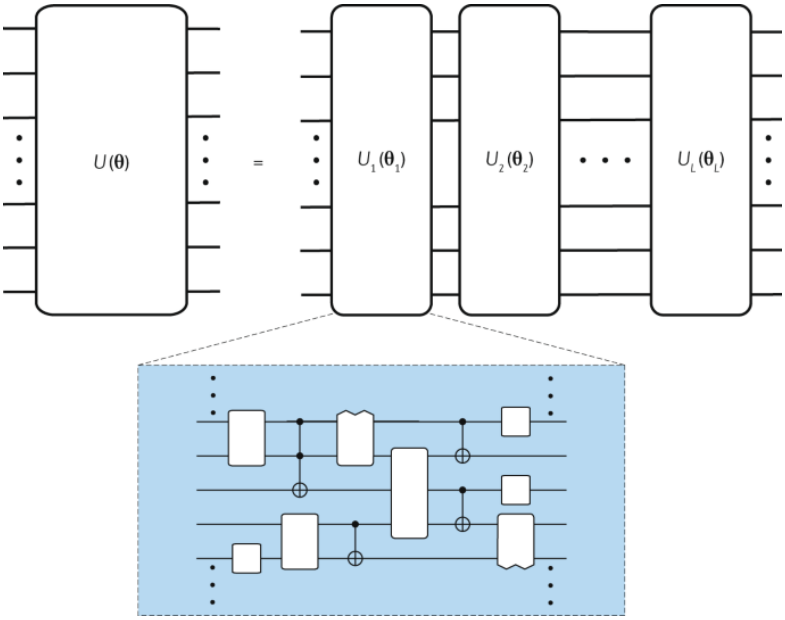


Figure 3.2: Illustration of an ansatz as the product of multiple PQCs [CAB+21].

3.1.3 Ansatz

Another important element of VQAs is the so-called ansatz, or sometimes also called PQC. It defines what the variational parameters θ are and consequently determines how they can be optimized during execution [CAB+21]. There are different structures available that can be used for an ansatz. *Problem-inspired ansatzes* are specifically designed for a given problem, while *problem-agnostic ansatzes* are generic and can be used even when a detailed problem description is not available [CAB+21].

Formally, the structure of an ansatz for a cost function $C(\theta)$ can be described by a unitary $U(\theta)$ that encodes the parameters θ and is applied to an input state [CAB+21]. The unitary $U(\theta)$ can then be seen as a product of multiple unitaries that are applied sequentially to the input state, where each unitary itself consists of parameterized and non-parameterized gates [CAB+21]. This concept is schematically illustrated in Figure 3.2.

As already briefly mentioned above, there are various different ansatz types, each with individual advantages and disadvantages. In the following, some well-known ansatz categories are briefly described based on the work of Cerezo *et al.* in [CAB+21]:

Hardware-efficient ansatzes *Hardware-efficient ansatzes* are specifically tailored to the underlying hardware with the goal of minimizing the required circuit depth of $U(\theta)$ on a certain quantum architecture. This is accomplished by implementing the unitaries with an optimized subset of quantum gates that are especially well interconnected and efficiently available on the given quantum hardware. This allows

to mitigate overheads in circuit construction as encoding symmetries and correlated qubits can be considered, resulting in an efficient ansatz design. A drawback of this ansatz type is that initialization needs to be done very carefully, since random initialization can cause trainability issues.

Unitary coupled clustered ansatz The problem-inspired *unitary coupled clustered ansatz* is mainly used in quantum chemistry to find the ground-state energy of molecules and is available in several variations.

Quantum alternating operator ansatz This ansatz type, as the name suggests, is mainly used in QAOAs. It employs an alternating structure that has been shown to be computationally universal.

Hybrid ansatzes Since current NISQ devices have various constraints, *hybrid ansatzes* integrate classical strategies into a quantum ansatz by offloading some work to a classical computer. This can be done by using classically optimizable coefficients, processing quantum circuits with existing tensor network techniques or dividing the whole ansatz into multiple smaller parts and solving them individually.

For further ansatz variants and more detailed descriptions, the interested reader is referred to the original work of Cerezo *et al.* [CAB+21].

The large number of different ansatzes makes it hard to fairly compare their performance and evaluate whether they are able to create the desired target state after the optimization [CAB+21]. Determining the expressibility of an ansatz is still an active are of research, but there are two different properties that can be used to compare ansatzes [CAB+21]. One is the *expressibility* which indicates whether an ansatz is able to uniformly explore the complete space of quantum states [CAB+21]. The other is the *entangling capability* that indicates the average entanglement of a state resulting from randomly sampled parameters θ.

3.1.4 Optimizers

The optimizer is a very important part of VQAs, as it is used to optimize the cost function and to update the variational parameters. The typical execution of a VQA as illustrated in Figure 3.1 starts by executing the ansatz with an initial set of parameters θ on the quantum computer [CDMB+22]. Then the quantum state is measured, and the parameters θ are updated by the optimizer and used as the new input parameters of the next execution step until a termination condition is met [CDMB+22]. The formal optimization task of a quantum-classical loop can be described by the following equation [CAB+21]:

$$\theta^* = \arg\min_{\theta} C(\theta) \tag{3.2}$$

The goal of the optimization task is therefore to find the set of parameters θ^* that minimize the cost function $C(\theta)$ [CDMB+22].

The optimization problem of VQAs is generally believed to be NP-hard and the correct choice of the optimizer is very important [BK21, CAB+21]. Although the optimizer is executed on a classical computer, there are special quantum-aware optimizers that can handle the noise and other issues that are present on current NISQ devices [CAB+21]. However, the optimal optimizer selection is still an active area of research since there are many options available [CAB+21]. In the following, some of the most common optimizer categories are introduced based on the work of Cerezo *et al.* [CAB+21]:

Gradient-based approaches This kind of optimizer iteratively proceeds in the direction indicated by the gradient and is the most common approach used for VQAs. Since on quantum hardware the gradients can only be statistically estimated through measurements, they are often referred to as Stochastic Gradient Descent (SGD) approaches. However, this concept is not new, as it is also used in many classical ML approaches. Therefore, available implementations are often just quantum adapted versions of well-known classical ML approaches, e.g., Adam [KB17].

Meta-learning This approach is also based on gradients but utilizes a neural network to determine the steps towards the goal [WSW+19]. This allows to also use the optimization history and knowledge of similar cost functions to efficiently optimize a given problem.

Simultaneous perturbation stochastic approximation An approach that rather uses an approximation of the gradients is called Simultaneous Perturbation Stochastic Approximation (SPSA) [Spa92]. Since these methods avoid complex gradient calculations at each iteration, it has been shown to allow fast theoretical convergence rates for some problem instances.

Sequential minimal optimization Another gradient- but also hyperparameter free approach has been specifically developed for VQAs with cost functions that are expressible as a sum of trigonometric functions [NFT20]. This allows to sequentially update only a few local parameters.

3.2 Challenges

Although VQAs are a very promising approach to take advantage of current quantum devices in the NISQ era, there are challenges and limitations that researchers should be aware of. In the following, some of the most important limitations of VQAs are discussed.

The purpose of training the internal parameters of a QML model is usually to minimize a loss function, with the ultimate goal to find the global minimum [CVH+22]. The loss function maps the parameters of the model to real values of the problem, e.g., the error of a task [CVH+22]. During training, the loss function landscape is traversed by iteratively

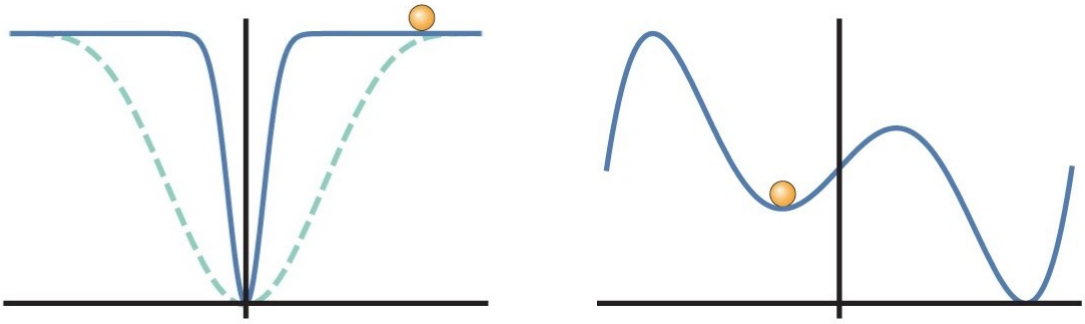
updating the model parameters. However, research on quantum landscape theory has shown that there are multiple challenges that can severely impact the performance of QML models [CVH+22].

One of these challenges is the phenomenon of multiple local minima in the loss function, which is also well known from classical ML tasks [CVH+22]. The existence of multiple local minima in the loss function is schematically illustrated in Figure 3.3b. These local minima can lead to suboptimal solutions, since optimization algorithms converge too far from the global optimum [BK21]. The optimization problem of VQAs has therefore been proved to be NP-hard [BK21, CVH+22]. To address this issue, various methods have been proposed, including methods which are able to turn local minima into saddle points [CVH+22].

Another related and very prominent limitation of VQAs are *barren plateaus* and *narrow gorges* [CVH+22, AHCC22]. The Barren plateau phenomenon was first described by McClean *et al.* in [MBS+18] and occurs during the search for the global minimum [CAB+21]. A typical loss function affected by the barren plateau phenomenon is illustrated in Figure 3.3a [CVH+22]. As indicated by the dashed line, the loss landscape is becoming exponentially flat on average with an increasing number of qubits used in a quantum system [MBS+18, CVH+22]. As a result, the area around the global minimum becomes narrower and decreases exponentially with the problem size, which is known as a narrow gorge [CVH+22, AHCC22].

It has been proven by Arrasmith *et al.* [AHCC22] that these two phenomenons are actually connected, and barren plateaus always occur along with narrow gorges. For their proof, Arrasmith *et al.* define the narrow gorges phenomenon as a probabilistic concentration of the cost function values around the mean [AHCC22]. Consequently, the presence of a barren plateau is logically equivalent, with the cost values being on average exponentially concentrated about the mean value [AHCC22]. If a loss functions has a well-defined minimum, this also implies that it has a narrow gorge [AHCC22].

Overall, these phenomenons make it exponentially hard for the optimizer to traverse the loss landscape, and thereby exponentially increase the required resources to find the global minimum during the execution of a VQA [CVH+22]. The performance impact is therefore significant, as it can severely decrease the performance of QML algorithms and can even invalidate quantum speedups [CVH+22]. Although the results of Arrasmith *et al.* [AHCC22] help to simplify the detection of barren plateaus, it is still hard to overcome their negative effects. However, a lot of research has been done to develop mitigation strategies. It has for example been shown that clever initialization strategies, pretraining, and parameter correlation help to address the issue of barren plateaus and result in more promising architecture designs than unbiased designs [CAB+21, CVH+22]. Another important observation regarding loss functions is that local observables that only compare quantum states at a single-qubit level are superior to global observables that measure multiple or all qubits when it comes to barren plateaus [CVH+22]. Moreover, it has also been shown that QML architectures with a high degree of entanglement are more prone to barren plateaus [CVH+22]. However, more detailed explanations and further



(a) Barren plateau (b) Local minima

Figure 3.3: Technical challenges of QML adapted from [CVH+22].

mitigation strategies can be found in the work of Cerezo *et al.* in [CAB+21, CVH+22] and in the work of McClean *et al.* in [MBS+18] as it goes beyond the scope of this thesis.

3.3 Hyperparameter Analysis

When working with VQAs, researchers are faced with the practical challenge of selecting the right hyperparameters for the given task. This includes basically all building blocks that were introduced in Section 3.1 and are depicted in Figure 3.1 such as optimizers, ansatzes or feature maps. These components can be combined individually for a given task and need to be manually selected before the parameters of a VQA can be trained.

Given the rapid pace of development and the plethora of different components available in literature and quantum programming frameworks, the selection of the correct hyperparameters can become quite cumbersome. Furthermore, it has been shown that the execution time and accuracy of VQAs heavily depends on the selected hyperparameters [CDMB+22]. Therefore, it is a necessity for any project to carefully select them to develop an efficient and accurate VQA for the task at hand. However, whether selected hyperparameters perform well depends significantly on the specific problem, which makes it hard to give a general recommendation or selecting them a priori [BMWV+23]. Consequently, it is important to know which hyperparameters are impacting the performance of VQAs to guide possible decisions in the selection process.

Hence, we want to answer *RQ 1* by reviewing existing literature regarding the most important hyperparameters. Although the available literature is still very limited, the following contains an overview of components that have already been identified as being relevant for the performance of VQAs. However, the following review is not exhaustive, but rather a selection of the most important findings that are relevant for this thesis.

Ansatz and Optimizer Cranganore *et al.* used VQAs for molecular dynamics simulations and analyzed the impact of different hyperparameters in [CDMB+22].

Specifically, they used the Mean Square Error (MSE) to compare the performance of different ansatzes, optimizers and application specific cost functions. However, during the experiments they especially focused on the ansatz and the optimizer, which were implemented using the Qiskit framework [Con]. For the ansatz they used the *EfficientSU2*, *RealAmplitudes*, *PauliTwo* and the *ExcitationPreserving* circuit and tried each with a different number of qubits, repetitions and entanglement strategies. Furthermore, the *Constrained Optimization BY Linear Approximations (COBYLA)*, *SPSA* and *Gradient Descent* optimizers were used to assess the impact of the optimizer.

According to the authors, the experiments showed that the choice of the optimizer only has a small effect on the performance, with the ansatz being more important. Moreover, they showed that linear entanglement is the best overall choice, while the number of repetitions showing no correlations with the MSE. Consequently, the ansatz could be identified as the most important among the considered hyperparameters. Finally, the authors were able to improve the MSE by up to 31% by applying insights about favorable hyperparameters to the target molecular dynamics workflow. This once again shows the importance of hyperparameter optimization in general and especially for VQAs.

Optimizer Bonet-Monroig *et al.* used VQAs for various different chemistry and material science problems in [BMWV+23] and state that the performance of VQAs greatly depends on the used optimizer. In their experiments they therefore focused on studying the performance of *SPSA, COBYLA, Sequential Least Squares Quadratic Programming (SLSQP)* and the *Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)* optimizers. They later in the experiment also hyperparameter tuned the *CMA-ES* and *SPSA* optimizer to compare them with their default settings and used the Cirq framework [Dev] to simulate quantum circuits.

The experiments showed mixed results for the different optimizers across the chosen problems, once again highlighting that the performance is heavily problem dependent. Furthermore, the hyperparameter tuning showed that *CMA-ES* is competitive with *SPSA* when fully optimized. In conclusion, the experiments showed that the performance of VQAs is highly problem dependent and hyperparameter tuning is necessary to achieve the best performance.

Ansatz Du *et al.* studied the performance impact of the ansatz in VQAs using data classification and quantum chemistry tasks [DHY+22]. The authors highlight that, based on empirical and theoretical findings, the ansatz has a major impact on the overall performance of VQAs. They therefore propose a way to automatically find near-optimal ansatz structures called quantum architecture search and demonstrated it on numerical simulators as well as quantum hardware through the IBM cloud. However, finding the optimal ansatz is difficult since ansatzes with more quantum gates provide stronger expressivity but may be less trainable due to accumulated noise. For the implementation, Qiskit [Con] and PennyLane [BIS+22] were used.

The results show that individually configured ansatzes can reduce the impact of noise and barren plateaus and are able to outperform VQAs with pre-selected ansatzes. This again emphasizes the importance of tailoring a VQA for a specific problem and shows the need for optimal hyperparameters.

Feature map For feature maps, we were not able to find data about their performance impact on VQAs in the currently available literature. However, Hossain *et al.* studied the effect of different feature maps on QSVMs for QML in [HAS+21]. Given that they used feature maps that are also commonly used in VQAs, the insights of their work are of great value for this thesis too. They analyzed the Second-Order Expansion (SOE) feature map, Z feature map and the ZZ feature map with repetition configurations using the Qiskit library [Con]. To assess the performance, they used the accuracy and the execution time of the QSVMs for different classification datasets.

The results show that higher repetition counts increase the time consumption linearly, with the SOE feature map being the fastest and the ZZ feature map being the slowest. Regarding the accuracy, there are mixed results dependent on the individual dataset, with both the SOE feature map and the Z feature map being among the best performing. However, the authors highlight that the performance of any supervised QML model significantly depends on the correct selection of the feature map and repetition count. Since feature maps are similarly used in VQAs to encode the input data into quantum states, it is likely that the choice of the feature map also affects the performance of VQAs.

This literature review shows that there is no clear evidence of one particular hyperparameter being exclusively important for the performance of VQAs yet. It rather suggests that there are multiple hyperparameters that need to be considered and carefully selected to achieve the best performance for a given task.

However, the *optimizer*, *ansatz* and *feature map* could be identified as being the most relevant hyperparameters for the performance of VQAs. Based on the findings of Cranganore *et al.* in [CDMB+22] we conclude that the choice of the ansatz might be more important than the optimizer. Moreover, the findings also suggest that the hyperparameter selection is highly problem dependent, as there are multiple experiments that show individual results for different datasets. This underlines the importance of well-considered hyperparameters and highlights the need for automated hyperparameter tuning algorithms.

CHAPTER 4

Hyperparameter Tuning for QML

The findings of Chapter 3 show that VQAs have various hyperparameters that are highly problem dependent and significantly impact their performance. Similar to classical ML projects, it is therefore a necessity to select appropriate hyperparameters in any new QML project. From the classical ML literature, the process of finding the optimal hyperparameters is called *hyperparameter tuning* and a common technique for building efficient ML models [YS20]. However, given the wide variety of different hyperparameter types that exist in today's ML models, there are also various different hyperparameter tuning approaches to choose from [YS20]. Unfortunately, there is only limited knowledge about hyperparameter tuning approaches for QML and in particular VQAs.

We therefore investigate existing hyperparameter tuning approaches and analyze their applicability for quantum use cases in the following. First, we highlight the importance of hyperparameter tuning algorithms and discuss the limitations of manual hyperparameter tuning. We then answer *RQ 2* from a theoretical perspective by reviewing existing tuning algorithms and evaluating their suitability for QML projects. Finally, we summarize the findings to conclude this chapter.

4.1 Applicability Analysis

A popular hyperparameter tuning approach is called *manual testing*, where a domain expert manually tries different configurations and evaluates their performance. However, manual testing has many drawbacks as it usually requires good understanding of the specific algorithm and can become very time-consuming with a larger number of hyperparameters, complex and long-lasting model evaluations or non-linear hyperparameter interactions [YS20]. The main advantages of using automated hyperparameter tuning techniques over manual testing were already introduced in Section 2.3.

Hyperparameter	Type	Subparameters	Type
Feature Map	Categorical	Repetitions, Entanglement	Conditional - Discrete, Categorical
Ansatz	Categorical	Repetitions, Entanglement	Conditional - Discrete, Categorical
Optimizer	Categorical	Max. Iterations, Tolerance	Conditional - Discrete, Continuous

Table 4.1: Common hyperparameter types of VQAs for QML.

To benefit from the advantages of automated hyperparameter tuning, various hyperparameter tuning algorithms for classical ML applications have been proposed over the years, each with its own advantages and disadvantages [YS20]. The main criteria to select appropriate tuning approaches are the types of hyperparameters that need to be optimized [YS20]. In general, hyperparameters are divided into *continuous*, *discrete*, *categorical* or *conditional* types [YS20]. Based on the findings in Chapter 3, we therefore categorized the most important hyperparameters of VQAs in Table 4.1. Additionally, in Table 4.1 we also included common subparameters of the respective hyperparameters and their types as they need to be considered in large scale hyperparameter tuning as well.

Based on the above categorization, we can review existing hyperparameter tuning algorithms and evaluate their suitability for QML use cases. This evaluation contributes to *RQ 2* and determines which tuning algorithms are evaluated during the experimental benchmarks. In the following, we therefore summarize the most common hyperparameter tuning approaches and asses their suitability for QML, based on the work of Yang and Shami in [YS20]:

Grid Search A relatively easy to implement and therefore very popular hyperparameter tuning method is called Grid Search (GS). As the name suggests, a finite subset of possible hyperparameters is manually defined in a *grid* of configurations by the user before the execution. GS then performs an exhaustive search over this defined search space by executing the model with each individual hyperparameter setting. Consequently, it cannot further explore promising regions of the search space on its own but needs to be repeated with a redefined search space by the user itself. Therefore, GS is unable to detect global optimums for discrete or continuous hyperparameters in most cases, especially if they cover large range of values and is rather recommended for categorical hyperparameters. Since GS is an exhaustive approach, the computational complexity for k hyperparameters with n distinct values grows exponentially at a rate of $O(n^k)$. This is also referred to as the *curse of dimensionality*, as the resource consumption increases exponentially with more hyperparameters being included in the search space. It is therefore considered inefficient for large search spaces with a high number of multidimensional hyperparameters. However, since GS can easily be implemented and is highly

parallelizable, it is still a very popular choice for hyperparameter tuning and often used as a baseline hyperparameter tuning approach.

Given the good support for categorical hyperparameters and the capability to execute it in parallel makes GS also a valid choice for an exhaustive QML baseline.

Random Search Another very popular and yet simple hyperparameter tuning approach is called RS. Initially developed to overcome the limitations of GS for large search spaces, RS randomly selects samples from the search space until a user defined budget is exhausted. This predetermined budget makes RS more efficient for large search spaces since the independent evaluations are distributed randomly and less time is wasted on poor-performing regions. Therefore, the computational complexity is $O(n)$ where n is the number of evaluations defined as the budget before the execution is started. Moreover, RS is also highly parallelizable and even able to detect a (near-)global optimum when the budget is chosen large enough. However, since the evaluations are chosen randomly, there are still a lot of poorly performing evaluations that could be avoided with more sophisticated hyperparameter selection that.

Overall, RS is a valid choice to explore the search space of QML models as it supports all hyperparameter types, is highly parallelizable and allows defining a fixed resource budget.

Gradient descent For some ML algorithms, gradient descent is a well-known optimization approach that locates the optimum by calculating the gradient of a variable and moving towards it. This allows to find a local optimum, which in the case of convex functions is also the global optimum. However, for non-convex functions this is a major drawback as gradient descent may only be able to find a local instead of a global optimum. Moreover, it does not support discrete, categorical or conditional hyperparameters since only continuous hyperparameters have gradient directions. This is a major drawback and makes gradient descent unsuitable for the mainly categorical hyperparameter landscape of VQAs.

Bayesian Optimization A more sophisticated but also very popular approach for hyperparameter tuning is called Bayesian Optimization (BO). The main advantage of BO is that the performance of previous evaluations is used to select the next hyperparameter setting from the search space. This is achieved by using a surrogate model that considers already explored settings and provides a predictive distribution. Then an acquisition function is used to decide between exploring new areas of the search space (exploration) or exploiting promising regions (exploitation). The balance between exploration and exploitation allows BO approaches to prioritize good hyperparameter configurations without missing promising configurations in the unexplored space. The surrogate model is updated once new evaluation results are available and used to determine the next hyperparameter settings iteratively until a user defined budget is exhausted.

BO is more efficient than GS and RS since previous results are considered, and evaluating the surrogate model is computationally less intensive than actual evaluations on the test data. Although it is hard to parallelize since the individual evaluations depend on each other, BO is expected to find near-optimal results very fast, making it a very promising approach for QML.

For the surrogate model, the Gaussian Process (GP), Random forest (RF) and Tree-structured Parzen Estimator (TPE) are the most common models for BO algorithms. However, each model has its own advantages and limitations:

BO-GP GP is a well-known surrogate model for BO algorithms. However, BO-GP is mainly used for continuous hyperparameters, does not support conditional hyperparameters and is hard to parallelize due to its cubic time complexity. It is therefore not a good choice for the mainly categorical hyperparameter space of QML workloads.

BO-RF This algorithm based on regression trees and sometimes also called Sequential Model-based Algorithm Configuration (SMAC). It supports all types of hyperparameters and has an overall time complexity of $O(n \log n)$ which is much lower than the previously introduced BO-GP variant. Although the parallelization capabilities are not great, the theoretical specifications suggest that this could be a valid approach for QML projects.

BO-TPE Another popular BO approach is BO-TPE which, as the name suggests, is based on Parzen estimators organized in a tree structure. The main building blocks are two density functions that contain good and bad evaluation results respectively, which are then used by the acquisition to determine the next evaluation configuration. BO-TPE has a time complexity of $O(nlogn)$ and is especially well suited for conditional hyperparameters by its tree structure, although it supports all hyperparameter types. It is therefore an appropriate hyperparameter tuning choice for typical QML applications with multiple categorical hyperparameters.

In conclusion, BO-RF and BO-TPE can be considered as valid approaches for the QML use case in this thesis.

Multi-fidelity Optimization Algorithms To overcome the issue of long execution times and high resource consumption, multi-fidelity optimization algorithms have been proposed. By combining low-fidelity evaluations that only use a subset of the original dataset and high-fidelity evaluations on the whole dataset, they try to find a balance between low resource consumption and effective hyperparameter tuning. The goal is to eliminate poor performing configurations already during the cheap low-fidelity evaluations. Popular approaches are *successive halving* and *Hyperband* that both implement this multi-fidelity idea. However, successive halving is hard to use since it is difficult to determine the correct budget for high-fidelity and low-fidelity runs. Hyperband overcomes this problem by dynamically allocating the budgets

and has good support for parallelization. However, it is considered inefficient for conditional hyperparameters and has the limitation of requiring subsets of the original problem to be representative. Another prominent hyperparameter tuning algorithm is called Bayesian Optimization HyperBand (BOHB), which combines BO with Hyperband. BOHB has therefore been shown to outperform many other tuning approaches for certain problems, since it provides high efficiency by taking advantage of two popular techniques. However, just as Hyperband, also BOHB requires that subsets of the problem are representative for the whole dataset while only using a small budget.

Overall, the use of multi-fidelity optimization algorithms requires low-fidelity evaluations to be representative for the problem at hand and do not support conditional hyperparameters very well. Since the hyperparameters of QML include conditional types, it can be assumed that there are better hyperparameter tuning approaches for this use case.

Metaheuristic algorithms Hyperparameter tuning methods of this kind are a popular choice for many optimization problems. Most of the available approaches are population-based and use generation-like concepts to evaluate the individual configurations. The individual algorithms mostly differ in how these generations are initialized and updated. Two of the most well-known algorithms are Genetic algorithm (GA) and Particle swarm optimization (PSO). GA is based on evolutionary theory and considers hyperparameter configurations as individuals that need to adapt in each generation circle to finally find the global optimum. It is a good choice when there is not much prior knowledge about the initial search space. However, GA introduces additional hyperparameters that need to be configured such as the fitness function or mutation rate which require additional knowledge. Furthermore, it has a runtime complexity of $O(n^2)$ and is hard to parallelize.

Similarly, PSO is also an evolutionary algorithm that is based on the individual behavior of particle swarms. The key idea is to let individual particles work through the configuration space in a semi-random manner, while sharing information about the performance of configurations throughout the swam. This tight cooperation allows finding the global optimum of the search space efficiently and results in a computational complexity of $O(n \log n)$. Compared to GA, it therefore convergences faster in most cases and additionally has great parallelization support since the individual particles run independently. A major challenge when working with PSO is to find a suitable population initialization, since it determines whether a global or only a local optimum is found. However, selecting suitable population initialization usually requires prior experience or sophisticated initialization techniques.

Overall, metaheuristic tuning approaches are considered more complex and challenging to use, especially if there is only limited initial knowledge about the search space. Although both above approaches support all types of hyperparameters, their limitations potentially make them not the best choice for a QML use case. GA introduces additional hyperparameters such as the fitness function or population

size that needs to be configured, which can be hard to determine for computationally intensive quantum workloads. PSO requires a suitable population initialization, which requires comprehensive knowledge about the hyperparameter space, which is currently not the case for QML.

More details on the technical and statistical principles of the above hyperparameter tuning algorithms can be found in the original work of Yang and Shami in [YS20] which serves as the basis of the above summary.

Overall, there are multiple hyperparameter tuning approaches that can be considered suitable for typical VQA problems. GS is a very prominent approach and the only exhaustive hyperparameter tuning method of the above, and therefore often used as a baseline approach. However, due to the curse of dimensionality it is not suited for large search spaces as the resource consumption increases exponentially [YZ20]. RS is also very popular because of its simplicity and through its budget limit the resource consumption can be regulated. Especially the Bayesian optimization based approaches SMAC and BO-TPE offer promising characteristics and seem to be a good choice for QML project because of their good support for categorical hyperparameters [YS20]. On the other hand, BO-GP, Hyperband and gradient descent cannot be recommended as they are not suitable for conditional hyperparameters [YS20]. Metaheuristic algorithms could theoretically be well suited as they are able to handle complex hyperparameter spaces but require comprehensible knowledge about the search space and its properties in advance, which might be a limitation in the case of QML.

CHAPTER 5

Methodology

In Chapter 4 we reviewed the most prominent hyperparameter tuning approaches and found multiple that could theoretically also be suited for QML problems. However, to verify their practical suitability, we adapt the most promising approaches and apply them to a typical QML use case. This allows us to investigate their performance and finally answer our third research question, *RQ 3* by comparing their performance. In this chapter, we therefore introduce the methodology used for the experiments conducted in this thesis.

First, we discuss the selected hyperparameter tuning methods and explain why we chose them for the performance evaluation. Next, we introduce the datasets that we used for the evaluation and briefly analyze the characteristics, such as the class distribution. We then extensively discuss the implemented preprocessing steps, including the used dimensionality reduction algorithms, and evaluate how well they are able to contain the original variance. Finally, we specify the used hyperparameter configuration space and discuss each individual hyperparameter and the corresponding sub parameters.

5.1 Hyperparameter Tuning Algorithms

To also evaluate the suitability of hyperparameter tuning algorithms from a practical perspective, we benchmark the most promising approaches from our review in Chapter 4 to get insights into their usability and performance. In the following, we discuss the selected approaches that are later used during our experiment:

Gridsearch As already discussed in Chapter 4, GS is a very prominent and widely used approach. Because it is an exhaustive approach that evaluates the whole search space, it can give valuable insights into the hyperparameter search space and is often used as a baseline to compare other tuning approaches against [SKK+23].

Similar to Yang and Shami in their performance comparison of hyperparameter tuning algorithms for classical ML models [YS20], we include GS as an exhaustive baseline model in our experiment. Moreover, GS is also recommended by Yang and Shami in [YS20] for categorical hyperparameters, which the majority of the hyperparameters of VQAs are.

Randomsearch RS is another very popular hyperparameter tuning approach that supports all types of hyperparameters and is highly parallelizable. However, other than GS, it allows specifying a budget of iterations independent of the search space size before the evaluation is started. This allows to plan the resource consumption in advance and makes it a promising approach for the computationally intensive evaluations of QML problems. Moreover, is a relatively simple tuning approach and is therefore also considered a baseline approach for budget based hyperparameter tuning algorithms by Yang and Shami in [YS20]. Consequently, RS could be a promising alternative for projects that would otherwise use manual hyperparameter tuning or exhaustive search, as the technical barrier is relatively low. We therefore select it as our second approach.

BO-TPE The third approach for our experiment is BO-TPE. We selected this approach based on the expected hyperparameters for our VQA use-case, as listed in Table 4.1 and the findings of Chapter 4. Consequently, we identified BO-TPE to be the most suitable hyperparameter tuning approach for three categorical hyperparameters with two optional conditional hyperparameters each. Moreover, Yang and Shami recommend this approach explicitly in their study [YS20] for search spaces mostly consisting of categorical and some conditional hyperparameters. Additionally, it supports all types of hyperparameters, including discrete and continuous hyperparameters, which are needed for the individual conditional hyperparameters. Although the individual evaluations are not independent, BO-TPE offers support for parallelization. Overall, we consider it therefore as a very promising hyperparameter tuning approach for our experiment.

5.2 Data

To study the performance of the selected hyperparameter tuning approaches, we focused on different classification problems. Classification problems have already been extensively studied for classical ML and are used in various research areas. Also, they are of great interest for the quantum community and are commonly used in quantum literature [SKK+23]. Insights on hyperparameter tuning techniques for these problems are therefore important and could be used in future research to speed up the tuning process. Moreover, there are plenty of well known datasets available that can be used for our experiment. Given the still limited number of qubits of current NISQ devices, we focused on low dimensional datasets. In the following, we discuss the selected datasets in more detail.

	Min	Max	Mean	SD	Class Correlation
Sepal length	4.3	7.9	5.84	0.83	0.7826
Sepal width	2.0	4.4	3.05	0.43	-0.4194
Petal length	1.0	6.9	3.76	1.76	0.9490 (high!)
Petal width	0.1	2.5	1.20	0.76	0.9565 (high!)

Table 5.1: Dataset statistics for the iris plants dataset [lDa].

5.2.1 Iris

The iris dataset [lDa] was first used in 1936 by R.A. Fisher [Fis36] and is one of the best-known datasets for classification today [UK21]. It consists of 3 classes, evenly represented by 50 instances per class, for a total of 150 instances. Each class represents a type of iris plant and is described by four numeric features, which are the length and width of the sepal and petal in centimeters. Some of the most important statistics of the dataset are shown in Table 5.1 which is based on the dataset description from Scikit-learn [lDa]. Overall, there are no missing values, and it is known that one class is linearly separable from the other 2 [lDa] as it can be seen in Figure 5.1 from the corresponding Qiskit example [Qis23b].

Because of the low dimensionality, it is a popular choice for QML as well and has for example been used by Hossain *et al.* in their study about feature mapping techniques [HAS+21]. However, due to its relatively small size and few instances it may not be representative for real world problems which are often much more complex and consist of more instances.

5.2.2 Real World Datasets

To also test the performance of our selected hyperparameter tuning methods with more realistic data, we used the Scikit-learn real world dataset collection [1] to find realistic datasets. We searched for datasets that have low dimensionality to ensure they can be efficiently processed on restricted NISQ devices with only a few qubits. In the following, we describe the selected datasets in more detail.

Forest Covertypes

The first dataset that matches our criteria is the Forest Covertype dataset [Bla98] that consists of seven classes which are described through 54 integer based features. The overall goal of the classification problem is to predict forest cover types based on cartographic variables that were originally determined by the US Forest Service Region 2 Resource Information System [Bla98]. The features of the dataset are described in more detail in Table 5.2.

[1] `https://scikit-learn.org/stable/datasets/real_world.html`, last accessed April 03, 2024

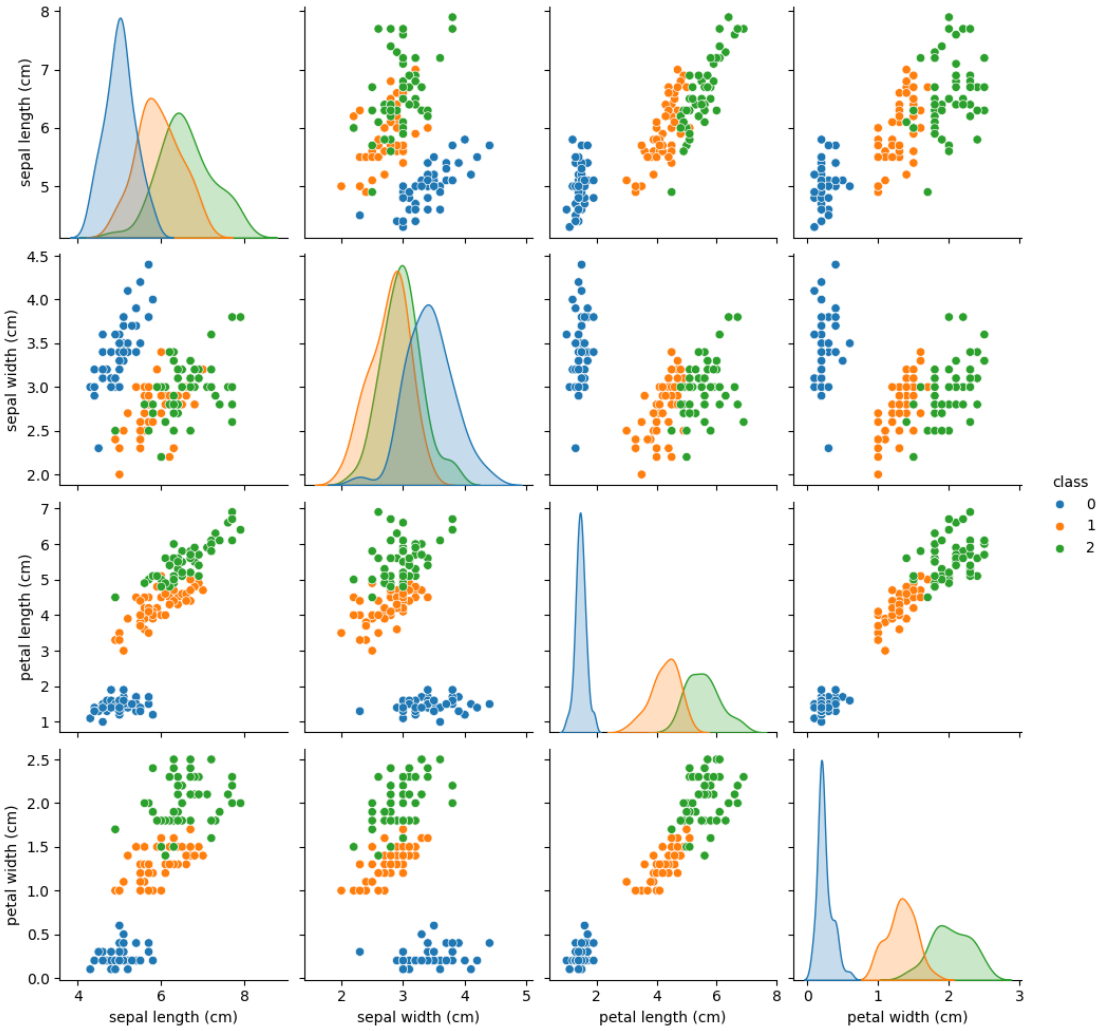


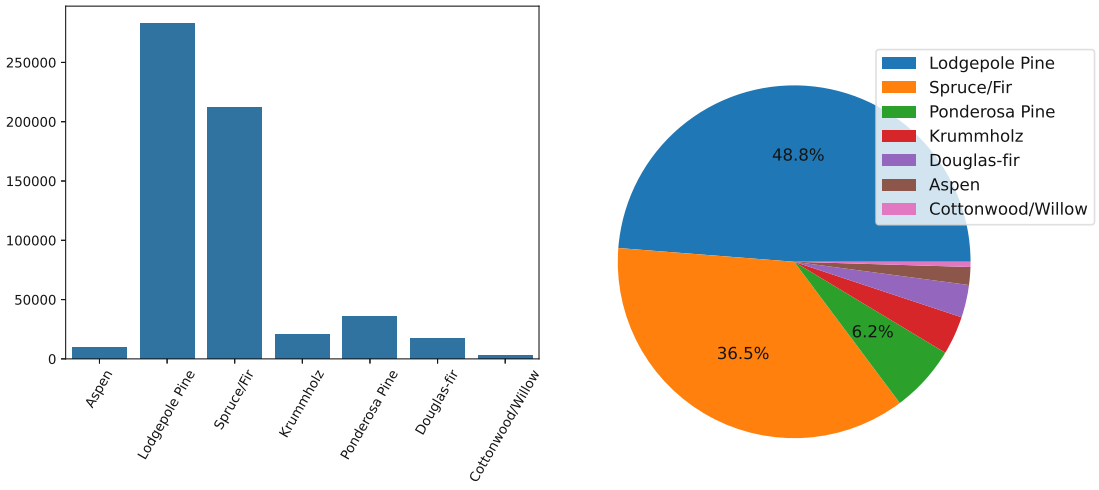
Figure 5.1: Correlation of features in the iris dataset [Qis23b].

The seven covertype class labels are *Spruce/Fir, Lodgepole Pine, Ponderosa Pine, Cottonwood/Willow, Aspen, Douglas-fir* and *Krummholz* [Bla98]. In contrast to the iris dataset, the classes are not distributed equally within the dataset. The absolute distribution of the 581.012 individual samples is illustrated in Figure 5.2a. In Figure 5.2b it can be seen that with more than 48% the most common target instance is *Lodgepole Pine* closely followed by *Spruce/Fir* with over 36% of the whole dataset. The other five instances together represent only about 15% of the total dataset and are therefore expected to be harder to predict.

Feature Name	Type	Description
Elevation	Integer	Elevation in meters
Aspect	Integer	Aspect in degrees azimuth
Slope	Integer	Slope in degrees
Horz_Dist_To_Hydrology	Integer	Horz Dist to nearest surface water features
Vert_Dist_To_Hydrology	Integer	Vert Dist to nearest surface water features
Horz_Dist_To_Roadways	Integer	Horz Dist to nearest roadway
Hillshade_9am	Integer	Hillshade index at 9am, summer solstice
Hillshade_Noon	Integer	Hillshade index at noon, summer soltice
Hillshade_3pm	Integer	Hillshade index at 3pm, summer solstice
Horz_Dist_To_Fire_Points	Integer	Horz Dist to nearest wildfire ignition points
Wilderness_Area (1-4)	Integer	Wilderness area designation
Soil_Type (1-40)	Integer	Soil Type designation

Horz = Horizontal, Vert = Vertical and Dist = Distance.

Table 5.2: Feature description of the Forest Covertypes dataset [Bla98].



(a) Absolute distribution.

(b) Relative distribution.

Figure 5.2: Class distribution of the Forest Covertype dataset.

KDD Cup 1999

The second dataset is the KDD Cup 1999 [LHF+00] which contains data about network connections from an intrusion detection system [2]. The overall goal of the classification problem is to detect attacks by distinguishing bad connections from the normal network traffic [lDb]. It was specifically designed to improve research in intrusion detection systems, and a version of it was used in the 1999 KDD intrusion detection contest [SFLP]. The data was created by simulating typical network traffic and manually injecting different kind of attacks [lDb]. Each instance of the dataset is described by 41 features that are explained in the task description [3] and listed in Table 5.3. Note that the nine features that are computed using a two-second time window, which are described in the lower third of Table 5.3, also exist for the destination host. However, we have omitted them in the table for readability reasons, as the descriptions would have been redundant. Consequently, there are nine similar features with a *dst_host_* prefix in the dataset and an additional *dst_host_same_src_port_rate* feature, summing up to 41 features in total.

Based on this features, the goal of the classification problem is to predict the correct class of traffic. Overall, there are 23 classes available in the dataset that have been divided into four main attack categories by Stolfo *et al.* in [SFLP] and the normal background traffic:

- Normal: *normal*
- Denial-of-Service (DOS):
  - *back, land, neptune, pod, smurf, teardrop*
- Unauthorized access from a remote machine (R2L):
  - *ftp_write, guess_passwd, imap, multihop, phf, spy, warezclient, warezmaster*
- Unauthorized access to local superuser (root) privileges (U2R):
  - *buffer_overflow, loadmodule, perl, rootkit*
- Surveillance and probing (PROBING):
  - *ipsweep, nmap, portsweep, satan*

Figure 5.3a shows the absolute count of the individual classes using a logarithmic scale to fit all classes in the figure. Again, the distribution of the classes is not equal, as it can be seen in Figure 5.3b. With over 56% the denial-of-service attack *smurf* is the most frequent attack, closely followed by *neptune* with more than 21%. However, the *normal*

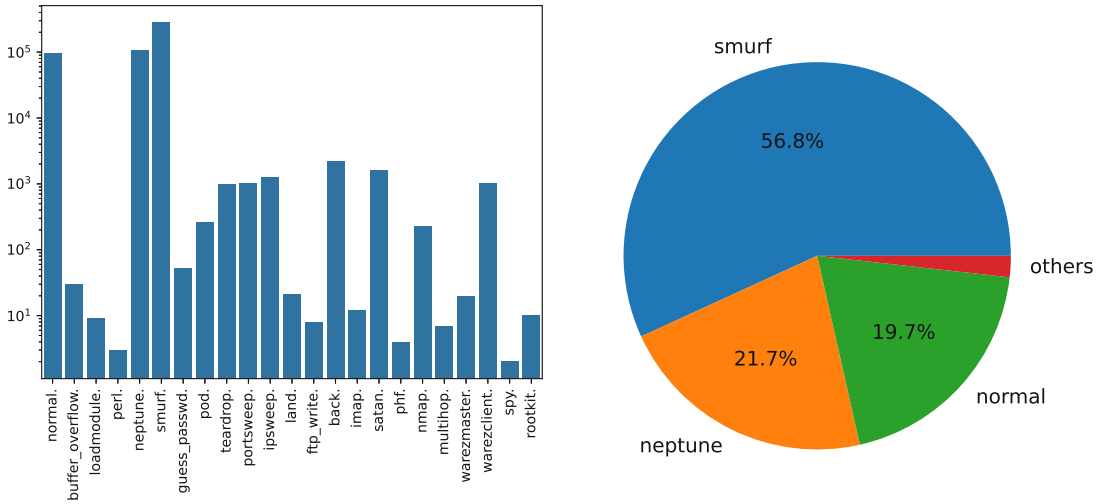
[2] `https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, last accessed April 03, 2024

[3] `https://kdd.ics.uci.edu/databases/kddcup99/task.html`, last accessed April 03, 2024

Feature Name	Type	Description
duration	continuous	length (number of seconds) of the connection
protocol_type	discrete	type of the protocol, e.g. tcp, udp, etc.
service	discrete	network service on the destination
src_bytes	continuous	number of data bytes from source to destination
dst_bytes	continuous	number of data bytes from destination to source
flag	discrete	normal or error status of the connection
land	discrete	1 if connection is from/to the same host/port
wrong_fragment	continuous	number of "wrong" fragments
urgent	continuous	number of urgent packets
Content features within a connection suggested by domain knowledge.		
hot	continuous	number of "hot" indicators
num_failed_logins	continuous	number of failed login attempts
logged_in	discrete	1 if successfully logged in
num_compromised	continuous	number of "compromised" conditions
root_shell	discrete	1 if root shell is obtained
su_attempted	discrete	1 if "su root" command attempted
num_root	continuous	number of "root" accesses
num_file_creations	continuous	number of file creation operations
num_shells	continuous	number of shell prompts
num_access_files	continuous	number of operations on access control files
num_outbound_cmds	continuous	number of outbound commands in an ftp session
is_hot_login	discrete	1 if the login belongs to the "hot" list
is_guest_login	discrete	1 if the login is a "guest"login
Traffic features computed using a two-second time window.		
count	continuous	number of connections to the same host as the current connection in the past two seconds
serror_rate	continuous	% of connections that have "SYN" errors
rerror_rate	continuous	% of connections that have "REJ" errors
same_srv_rate	continuous	% of connections to the same service
diff_srv_rate	continuous	% of connections to different services
srv_count	continuous	number of connections to the same service as the current connection in the past two seconds
srv_serror_rate	continuous	% of connections that have "SYN" errors
srv_rerror_rate	continuous	% of connections that have "REJ" errors
srv_diff_host_rate	continuous	% of connections to different hosts

Some descriptions have been shortened.

Table 5.3: Feature description of the KDD Cup 1999 dataset [SFLP].



(a) Absolute distribution. (b) Relative distribution.

Figure 5.3: Class distribution of the KDD Cup 1999 dataset.

traffic only makes up roughly 20% of the data, resulting in an unrealistic high proportion of attacks in the dataset [lDb]. Moreover, for some attack classes like *spy*, *perl* or *phf* there exist only 2, 3 and 4 instances in the whole dataset which makes them extremely hard to predict within the 4.898.431 total instances.

5.3 Preprocessing

To evaluate the performance of the selected hyperparameter tuning methods with the different datasets, some preprocessing was necessary to ensure the data is consistent and can be efficiently processed in our experiment. We used the Scikit-learn framework [PVG+11] to load and preprocess the datasets. In particular, we used the *load_iris*, *fetch_covtype* and *fetch_kddcup99* utilities of Scikit-learn to load the respective datasets.

We then analyzed the individual datasets and checked that there are no missing values and the data has the correct format. The features and labels of the Iris and Covertype dataset were all numeric and did not require further preparation. However, the *protocol_type, service* and *flag* features as well as the target labels of the KDDCup99 dataset are contained as categorical features and need to be encoded as numeric values for our pipeline. We therefore used the *LabelEncoder* for the target labels, the *OneHotEncoder* to encode the non-numerical features and the *StandardScaler* to normalize numeric features. All of the above functions can be found in the Scikit-learn preprocessing library [4].

[4] `https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing`, last accessed April 06, 2024

5.3.1 Dimensionality Reduction

Large real world datasets often contain many features that are irrelevant or redundant to correctly predict the target label [YS20]. Dimensionality reduction algorithms are therefore a well-known technique for classical ML applications to eliminate unimportant features [YS20]. However, especially for QML, it is important to efficiently reduce the number of features since current NISQ devices only offer a limited number of qubits [MP22]. It is therefore common to reduce the dimensionality of a given dataset before it is used for QML classification tasks.

Two of the most well-known approaches are called Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) [YS20].

PCA PCA is a linear dimensionality reduction approach that uses the eigenvectors of the covariance matrix to create new features of the original data [YS20]. The fewer new features are a linear combination of multiple original features, with the first components containing the most information [MP22]. To calculate the eigenvectors of the covariance matrix, the most common method is Singular value decomposition (SVD).

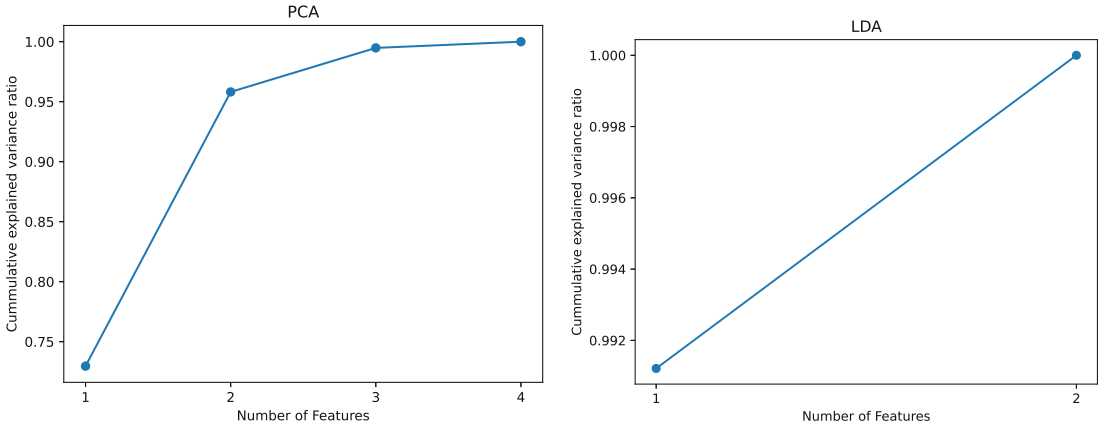
LDA LDA is also a linear dimensionality reduction approach that additionally uses the target classes to maximize class separation [MP22]. The goal is to achieve a minimal variance inside classes but maximize the variance between different classes [YS20]. The number of reduced features k is limited by the number of original features n with $k < n - 1$ [MP22].

While both approaches are well suited for QML, the results of Mancilla and Pere show that LDA performed better than PCA in their experiment [MP22]. They evaluated the performance for two datasets by reducing their features to only two from originally 25 and 114 respectively. However, they state that further investigation is needed to fully understand the prevalence of LDA and plan to study for more datasets.

Since for the datasets in our experiment it is uncertain which approach performs better, we employ both approaches. To fairly compare them, we use the same number of qubits for PCA and LDA. We used the *PCA* [5] and *LDA* [6] implementations of the Scikit-learn framework [PVG+11] that offer the *n_components* parameter to define the number of components to keep after reduction. In the following, we shortly discuss how we applied them to the individual datasets.

[5] `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`, last accessed April 06, 2024

[6] `https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html`, last accessed April 06, 2024



(a) Explained variance ratio for PCA. (b) Explained variance ratio for LDA.

Figure 5.4: Explained variance ratio for the Iris dataset.

Iris

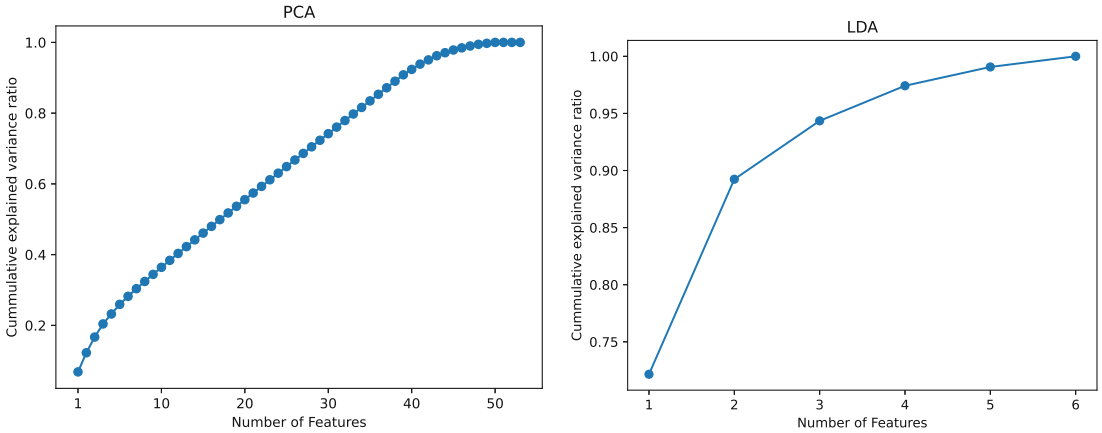
As already discussed above, the maximum number of reduced features for LDA is limited by the number of target classes in the original dataset. Since the Iris dataset only has 3 target classes, we set the *n_components* parameter for both approaches to two. To get insights of how well the reduced features can keep the variance of the original dataset, Scikit-learn offers the *explained_variance_ratio_* function. We illustrated the cumulative explained variance for the Iris dataset in Figure 5.4. As it can be seen, for two components PCA can keep more than 95% of the variance from the original dataset while LDA is able to keep 100%. We therefore expect LDA to slightly perform better than PCA in our experiment for this dataset.

Forest Covertypes

The Forest Covertype dataset has seven target classes and 54 features. With the class limitation of LDA, we reduced them to six using PCA and LDA. We again evaluated the cumulative explained variance and illustrated it in Figure 5.5. As it can be seen, for six components PCA can only keep roughly 30% of the variance from the original dataset while LDA is still able to keep 100%. For PCA to keep 99% of the original variance, it would need 48 features.

KDD Cup 1999

The KDD Cup 1999 dataset has 23 target classes and therefore by far the most out of the three datasets. However, the number of qubits is very limited on current NISQ devices and the running time increases drastically with the number of used qubits for simulators. To allow a fair comparison of our two real world datasets, we decided to allocate the same amount of qubits for each of them. This ensures comparable runtimes and resource



(a) Explained variance ratio for PCA. (b) Explained variance ratio for LDA.

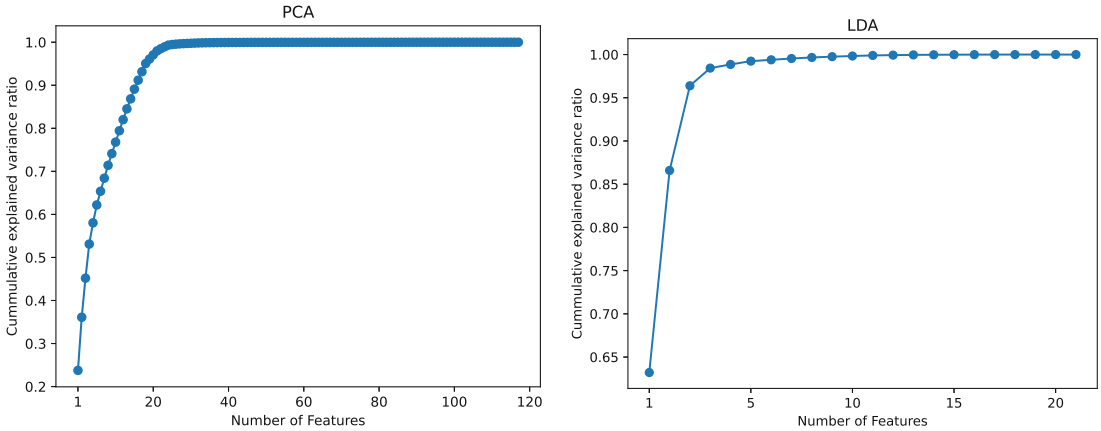
Figure 5.5: Explained variance ratio for the Forest Covertype dataset.

usage, as well as comparable accuracy results. We therefore reduce the 41 features of this dataset to six, just as for the Forest Covertypes dataset. Moreover, we are able to keep 99% of the original variance with these six features when using LDA as described in the following.

The cumulative explained variance for PCA and LDA is illustrated in Figure 5.6. Because we used the *OneHotEncoder* to encode non-numeric features, we have in total 118 individual features since there are three different *protocol_type* instances, 66 different *service* types and eleven different *flag* instances. Figure 5.6a shows that the first six components of PCA contain a little over 65% of the original variance, while Figure 5.6b shows that LDA is able to keep over 99% of the original variance. For PCA to keep 99% it would need 24 features which is only half of the required features compared to the Forest Covertype dataset.

5.4 Hyperparameters

To evaluate the selected hyperparameter tuning methods, we need to define an appropriate search space. We focused on finding a reasonably large search space that contains enough variables to extensively evaluate the performance of the budget based tuning methods, while not being too large for the exhaustive GS evaluation. Generally, we use the same hyperparameter configuration space for all tuning methods to allow a fair comparison, as suggested by Yang and Shami in [YS20]. The complete hyperparameter configuration space is specified in Table 5.4 and contains 504 individual hyperparameter combinations. In the following, we discuss each of the used hyperparameters in detail.



(a) Explained variance ratio for PCA. (b) Explained variance ratio for LDA.

Figure 5.6: Explained variance ratio for the KDD Cup 1999 dataset.

Hyperparameter	Configuration	Subparameter	Values
Feature reduction	PCA LDA	/	/
Feature Map	PauliFeatureMap ZFeatureMap ZZFeatureMap	Repetitions	[1,5]
Ansatz	PauliTwoDesign RealAmplitudes EfficientSU2	/ Entanglement	/ [full, linear, sca]
Optimizer	COBYLA SLSQP NELDER_MEAD	Max. Iterations	[100, 1000]

Table 5.4: Hyperparameter configuration space used for experiment.

5.4.1 Feature Map

As already discussed in Chapter 3.1.2, quantum feature maps are used to encode the classical data in quantum states. In the literature, there are already various feature maps available, such as the Z feature map or the ZZ feature map [HAS+21]. For our experiment, we chose the three well-known implementations named *Pauli Feature Map*, *Z Feature Map* and the *ZZ Feature Map* that are available in the Qiskit circuit library [7] and can be used out of the box. The Pauli feature map was originally proposed in [HCT+19] and is used as the basis for the Z Feature Map which has no entangling gates since the

[7] https://docs.quantum.ibm.com/api/qiskit/circuit_library, last accessed April 07, 2024

Pauli strings are fixed as ['Z'] [8]. The ZZ Feature Map is also based on the Pauli feature map, but in contrast to the Z Feature Map offers support for entanglement.

As studied by Hossain *et al.* in [HAS+21], a common parameter for feature maps is the number of repetitions that allows to specify the circuit depth by repeating the respective feature map circuit multiple times. However, the results show that the best circuit depth configurations varies between different datasets is therefore heavily problem depended. We therefore include the *reps* parameter in our hyperparameter search space to find the best configuration for our datasets.

5.4.2 Ansatz

As discussed in Section 3.1.3, the ansatz defines the variational parameters of the VQA and is therefore an essential building block. In our experiments we use three well-known ansatz types that are available in the Qiskit circuit library and can be used out of the box: *PauliTwoDesign*, *RealAmplitudes* and *EfficientSU2*. The authors of [CDMB+22] studied the performance of all of these three ansatzes with different repetitions and entanglement strategies. Their results show that the *linear* entanglement strategy performs better than *full* and *sca* throughout their experiments, while the optimal number of repetitions varies depending on the concrete use case [CDMB+22]. However, we decided to include the entanglement strategy as another conditional categorical hyperparameter, since we already included the discrete repetitions parameter for the feature maps. The *RealAmplitudes* and *EfficientSU2* ansatz both support this parameter, while the *PauliTwoDesign* does not offer it. More details on the individual ansatz types can be found in [CDMB+22].

5.4.3 Optimizer

The last remaining hyperparameter of the VQA that we need to tune is the optimizer, which we already discussed in Section 3.1.4. Qiskit [9] offers numerous optimizers to choose from. For our experiment we choose the two common gradient free optimizes *COBYLA* and *Nelder-Mead* as well as the gradient-based *SLSQP* optimizer. In the study [SMM23] of Singh *et al.* where they evaluated a large range of different optimizers for a quantum chemistry use-case, the *COBYLA* and *SLSQP* optimizers are consistently among the top performing optimizers. Moreover, we run some preliminary benchmarks on our own, where these two optimizers also showed promising results. We therefore include the *COBYLA* and *SLSQP* optimizers in our hyperparameter selection. Additionally, we included the *Nelder-Mead* optimizer, as it surprisingly also showed promising accuracy results in our preliminary benchmarks. More details on the individual optimizers can be found in the corresponding Qiskit documentation [9] or in the work of Singh *et al.* [SMM23].

[8] `https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.ZFeatureMap`, last accessed April 07, 2024

[9] `https://docs.quantum.ibm.com/api/qiskit/0.46/qiskit.algorithms.optimizers`, last accessed April 07, 2024

Singh *et al.* suggests setting the number of maximum iterations to a fixed value for all optimizers to allow a fair comparison by allocating the same computational resources [SMM23]. In their experiment, they set this number to 100 for all optimizers. We adopted this setting for our experiment but also test the optimizer with an additional value of 1000 which is the default value for the COBYLA optimizer in the Qiskit implementation. This allows us to get insights on how this parameter influences the performance.

CHAPTER 6

Results

In this chapter, we evaluate the results of our experiments and briefly discuss them. First, we introduce the experimental setup and hardware used throughout our experiments. We then extensively evaluate the accuracy and runtime of Grid Search (GS), Random Search (RS) and Bayesian Optimization (BO)-Tree-structured Parzen Estimator (TPE) for each of the datasets before we analyze different parallelization configurations. Next, we share some insights on how the actual hyperparameters perform for each dataset and evaluate the most promising choices. Finally, we analyze how budget based hyperparameter tuning algorithms RS and BO-TPE can handle larger search spaces.

6.1 Experimental Setup

To implement our experiments, we use Qiskit [Qis23a], a Python based open-source quantum framework that provides the required VQA components in its circuit library [1]. Furthermore, we use the Python based scikit-learn [PVG+11] library for loading and preprocessing the data, as described in Chapter 5. To evaluate the performance of GS, we use the GridSearchCV [2] implementation from the scikit-learn [PVG+11] library. For RS and BO-TPE we use Hyperopt [BYC13], a well known open source Python library that supports parallel evaluations using MongoDB [YS20]. The exact versions of the software used in our experiments can be found in Table 6.1.

6.1.1 Hardware

Qiskit supports different computing backends and allows to easily specify whether the quantum calculations should be run on a dedicated quantum computer or a simulator.

[1] https://docs.quantum.ibm.com/api/qiskit/circuit_library, last accessed April 07, 2024

[2] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html, last accessed April 19, 2024

Software	Version
hyperopt	0.2.7
mongodb	3.6.8
pymongo	3.12.0
python	3.8.10
qiskit	0.45.1
qiskit-machine-learning	0.7.1
scikit-learn	1.3.2

Table 6.1: Software versions used in experiment.

Public access to dedicated quantum computers is still very costly, and long waiting queues make larger experiments very inefficient. IBM for example offers access to quantum systems with more than 100 qubits [3] in their quantum cloud for free but limits it to 10 minutes of runtime per month. However, since our experiments exceed this limit by far, we decided to run Qiskit locally on a regular server to simulate a perfect quantum computer for our evaluations. We used an Ubuntu 20.04.6 LTS server with two AMD EPYC 7452 32-Core processors that have a base frequency of 2.35 GHz and 64 threads each. With 1 TB of RAM and 128 threads in total, this system offers great parallelization support with.

6.2 Performance Evaluation

In our experiment we evaluate the performance of GS, RS and BO-TPE for each dataset. We use the same hyperparameter configuration space that is defined in Table 5.4 for each hyperparameter tuning approach to allow a fair comparison. Each dataset is split in a train and test subset [4] using stratified sampling to preserve the relative class frequency of the target labels. The extensive runtime of the individual evaluations made it necessary to limit the train split to 450 samples and the test split to 150 for the real-world datasets. Larger splits resulted in significantly longer runtimes, which for the scale of these experiments would have exceeded the time budget of this thesis. As the classification performance metric, we use the mean accuracy of 3-fold cross validation. Additionally, we track the total time that each hyperparameter tuning algorithm needs to evaluate the hyperparameter search space and use it as an efficiency metric, similar to Yang and Shami [YS20].

With GS, we exhaustively evaluate the whole configuration space and use these results as our performance baseline. Because of the great parallelization support, we utilize all 128 threads at the same time for the GS approach. For the budget based approaches RS and BO-TPE we set the maximum evaluations to 100 as recommended by Yang

[3] `https://www.ibm.com/quantum/pricing`, last accessed April 19, 2024

[4] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`, last accessed April 10, 2024

Dataset	Tuning Algorithm	Accuracy	SD	Time (hh:mm:ss)
Iris	GS	59.18%	3.33%	01:02:30
	RS	57.82%	3.85%	00:27:04
	BO-TPE	61.22%	1.67%	00:28:09
Forest Covertype	GS	54.89%	6.58%	77:13:23
	RS	55.78%	7.93%	59:51:48
	BO-TPE	56.00%	5.36%	37:06:35
KDD Cup 1999	GS	91.33%	2.83%	50:06:29
	RS	84.22%	2.79%	58:20:19
	BO-TPE	87.33%	0.54%	57:24:05

SD = Standard Deviation

Table 6.2: Performance of different hyperparameter tuning algorithms for each dataset.

and Shami [YS20] to fairly compare their performance. Similar to GS, we use full parallelization for RS and use 100 threads simultaneously. However, for BO-TPE this is not the best configuration since all 100 configurations would be randomly selected at the beginning of the search without considering previous results or taking advantage of the surrogate model. We analyzed the parallelization configuration in Section 6.3 and identified 20 threads to perform best in terms of accuracy. We therefore only use 20 threads for BO-TPE while leaving the evaluation budget unchanged at 100 runs. The results of our experiment can be found in Table 6.2.

Overall, it can be seen that the measured accuracy and time varies between the individual datasets. However, when looking at the individual datasets, all three hyperparameter tuning approaches were able to achieve a similar accuracy with great differences in the time needed to complete the run. As expected, the evaluation of the Iris dataset could be completed the fastest, as it is a low-dimensional dataset with only a few instances to classify. The results of the Iris and Forest Covertype dataset are in line with our expectations and show that the execution time of GS is by far the longest. BO-TPE shows very promising results for these two datasets and was able to achieve the highest accuracy for both datasets while requiring only half the time compared to GS. Since all hyperparameter tuning approaches use the exact same hyperparameter search space, the differences in accuracy are most likely a result of the internal randomness of the optimizer. For the KDD Cup 1999 dataset we observe different results with GS being the fastest approach and the highest accuracy. To better understand these results, we provide additional insights for each dataset in the following sections and analyze their performance in more detail.

6.2.1 Iris

In this section, we further analyze the results for the Iris dataset. We first provide more insights into the accuracy and analyze the median and accumulated maximum accuracy

as well as the best configurations before we also analyze the runtime.

Accuracy

Table 6.2 shows that the best accuracy of all three hyperparameter tuning methods for the Iris dataset is almost equal. With GS we evaluated the whole hyperparameter configuration space and found the best accuracy to be 59.18% while RS reports a slightly worse accuracy of 57.82%. Interestingly, BO-TPE reports an even better accuracy than GS of 61.22%, although all approaches use the exact same hyperparameter search space. We find these accuracy results to be consistent with the results from the Qiskit tutorial [5] that also uses the Iris dataset and dimensionality reduction with two features.

To further analyze these differences, we have listed the corresponding hyperparameter configurations of these accuracy values for each hyperparameter tuning approach in Table 6.3. As it can be seen, there are two configurations for each approach that led to the exact same accuracy value. For GS these two configurations only differ in the optimizer while for BO-TPE only the entanglement strategy is different. RS apparently evaluated the same configuration twice, as there are no differences. Moreover, Table 6.3 shows that all tuning approaches report the same hyperparameter configuration as the best performing one, with the only difference being the maximum iterations of the optimizer. We therefore assume that the difference in accuracy between the tuning approaches is a result of the random initialization used for the optimizer and the deviation that naturally occurs from cross-validation. The initial value of the optimizer affects how the optimizer converges and is set randomly at the beginning of a run. Consequently, even identical hyperparameter configurations can lead to slightly different results as the number of maximum iterations is fixed. Furthermore, we use the mean accuracy of 3-fold cross validation as our performance metric. The corresponding standard deviation can be found in Table 6.2 and may also be a reason for the slight differences in accuracy between the tuning approaches.

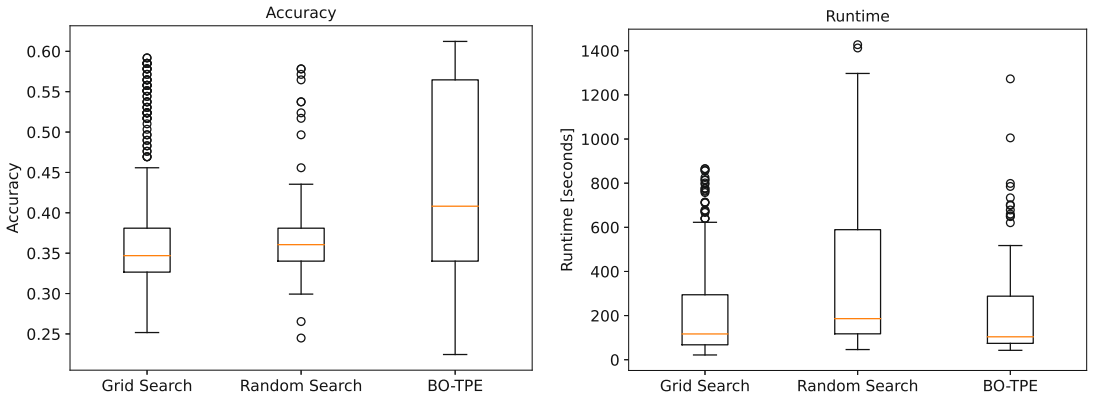
To further analyze the performance of the different hyperparameter tuning approaches, we visualize the accuracy results of the individual runs using a box plot in Figure 6.1a. For GS there are 504 individual evaluations representing the complete hyperparameter configuration space, while for RS and BO-TPE there are 100 evaluations due to our budget restriction. It can clearly be seen that the median accuracy for BO-TPE is significantly higher than for the other approaches. This shows that the guided search of BO-TPE helped to find more promising hyperparameter configurations with higher accuracy while spending less time evaluating poor configurations. The box plot of GS and RS looks almost identical, with the median accuracy of RS being slightly better. However, accuracy values greater than 0.5 are considered outliers for both approaches, indicating that the majority of evaluations performed poorly.

[5] https://qiskit-community.github.io/qiskit-machine-learning/tutorials/02a_training_a_quantum_model_on_a_real_dataset.html, last accessed April 10, 2024

Algo.	Dim. Red.	Feature Map	Reps	Ansatz	Ent.	Optimizer	Max. Iter.	Time
GS	PCA	ZFeatureMap	1	EfficientSU2	full	COBYLA	1000	468s
	PCA	ZFeatureMap	1	EfficientSU2	sca	SLSQP	1000	262s
RS	PCA	ZFeatureMap	1	EfficientSU2	full	COBYLA	1000	565s
	PCA	ZFeatureMap	1	EfficientSU2	full	COBYLA	1000	655s
TPE	PCA	ZFeatureMap	1	EfficientSU2	full	COBYLA	100	75s
	PCA	ZFeatureMap	1	EfficientSU2	linear	COBYLA	100	71s

Algo. = Hyperparameter Tuning Algorithm, Dim. Red. = Dimensionality Reduction Method, Ent. = Entanglement Strategy, Max. Iter. = Maximum Number of Iterations

Table 6.3: Best Hyperparameter Configurations for the Iris Dataset.



(a) Box plot of the Accuracy.

(b) Box plot of the Runtime.

Figure 6.1: Accuracy and Runtime of the Iris dataset.

To get insights into the search progress of the different approaches, we illustrated the accumulated maximum accuracy over the iterations in Figure 6.2. Note that for GS only the first 100 iterations are shown to match the defined evaluation budget of RS and BO-TPE. The maximum accuracy of GS is found in a later iteration but showing all 504 iterations would reduce the readability of the illustration. It can be seen that RS found the maximum relatively early and did not further improve its result after about 15 iterations. BO-TPE also found a relatively good result within the first 15 iterations, but was able to slightly improve the accuracy after about 70 iterations. Overall, this shows that randomly selecting hyperparameter configurations can lead to very promising results with a very limited budget of only 20 runs. Especially since BO-TPE also randomly selects the first candidates of the search before using the results for the next iteration. Since we use 20 threads in parallel for BO-TPE, the first 20 iterations are essentially randomly selected.

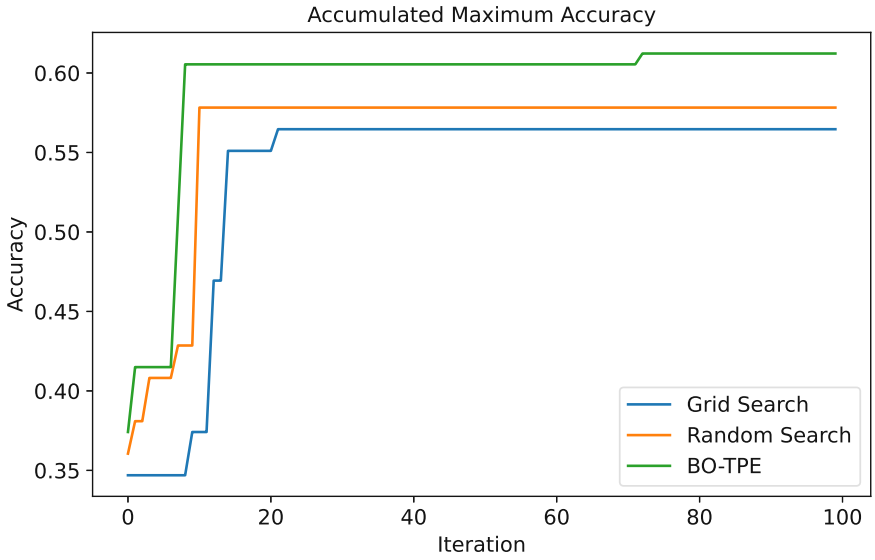


Figure 6.2: Accumulated maximum accuracy for the Iris dataset.

Runtime

In Figure 6.1b we visualize the runtime of the individual evaluations using a box plot. It can be seen that the median runtime for all hyperparameter tuning approaches is around 200 seconds, with RS being slightly higher than the others. Furthermore, the upper quartile of the box plot for RS is larger, indicating a greater number of long evaluations. However, is can also be seen that there are outliers across all hyperparameter tuning approaches, with a significantly higher runtime compared to the median. We analyzed these runs and could not find any correlations between particular hyperparameters since all different feature maps, ansatzes, optimizers and dimensionality reduction algorithms are among the 10 longest runs. We therefore believe that the significantly increased runtimes are a result of the barren plateau phenomenon. Since only a small portion of the evaluations are affected by the phenomenon, we assume that it could be related to the random initialization of the optimizer. However, further analysis of different initialization strategies or distributions would be beyond the scope of this thesis.

6.2.2 Forest Covertypes

In this section, we further analyze the results for the Forest Covertypes dataset. Similar to the previous section, we start by analyzing the accuracy before evaluating the runtime.

Accuracy

In Table 6.2 it can be seen that the best accuracy of all three hyperparameter tuning methods for the Forest Covertypes dataset is almost equal. Interestingly, GS achieved the lowest accuracy of 54.89%, although it evaluated the whole hyperparameter configuration

Algo.	Dim. Red.	Feature Map	Reps	Ansatz	Ent.	Optimizer	Max. Iter.	Time
GS	LDA	ZFeatureMap	5	EfficientSU2	full	SLSQP	100	29656s
	LDA	ZFeatureMap	1	EfficientSU2	full	SLSQP	1000	28582s
RS	LDA	ZFeatureMap	1	EfficientSU2	linear	SLSQP	1000	42136s
TPE	LDA	ZFeatureMap	1	RealAmplitudes	full	SLSQP	1000	10275s

Algo. = Hyperparameter Tuning Algorithm, Dim. Red. = Dimensionality Reduction Method, Ent. = Entanglement Strategy, Max. Iter. = Maximum Number of Iterations

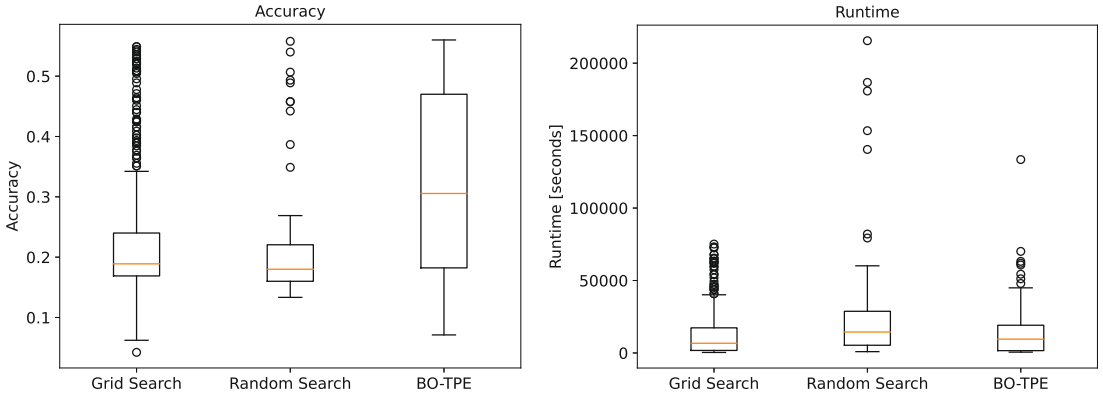
Table 6.4: Best Hyperparameter Configurations for the Forest Covertypes Dataset.

space. The best accuracy of RS is about 1% higher at 55.78% and BO-TPE achieved an even better accuracy of 56.00%. We again listed the corresponding hyperparameter configurations of these accuracy values for each tuning tuning approach in Table 6.4. As it can be seen, there are two configurations for GS that achieved the same accuracy and one configuration for RS and BO-TPE respectively. For GS these two configurations only differ in the repetitions of the feature map and the maximum iterations of the optimizers. Generally, it can be seen that all tuning approaches report a similar hyperparameter configuration as the best performing one, with the only difference being the ansatz for BO-TPE and the entanglement strategy for RS. We therefore assume that the different accuracy values between the hyperparameter tuning approaches can again be explained by the relatively high standard deviation from the used cross-validation and the random initialization used for the optimizer.

We again visualize the accuracy results of the individual runs using a box plot in Figure 6.3a to analyze the performance of the different hyperparameter tuning approaches. Similar to the Iris dataset, it can be seen that the median accuracy for BO-TPE is significantly higher than for the other approaches. This again shows that the guided search of BO-TPE helped to find more promising hyperparameter configurations within the same budget of 100 evaluations as RS. For GS, the median accuracy is significantly lower than for BO-TPE but slightly higher than for RS. The box plot of GS and RS look very similar, and evaluations with an accuracy over 0.4 are considered as outliers for both approaches. However, we can see that there are more outliers for GS than for RS due to the higher number of evaluations.

In Figure 6.4 we illustrate the accumulated maximum accuracy over the iterations for the Forest Covertypes dataset to get insights into the search progress of the different hyperparameter tuning approaches. We only show the first 100 iterations of GS although the maximum accuracy is found after more than 400 iterations to improve the readability. This also allows us to better analyze the progress of the budget based approaches, RS and BO-TPE. For RS it can be seen that a relatively good configuration is already found within the first 10 iterations with a slight improvement after about 80 iterations.

For BO-TPE keep in mind that the results of previous iterations are used to select the



(a) Box plot of the Accuracy.

(b) Box plot of the Runtime.

Figure 6.3: Accuracy and Runtime of the Forest Covertypes dataset.

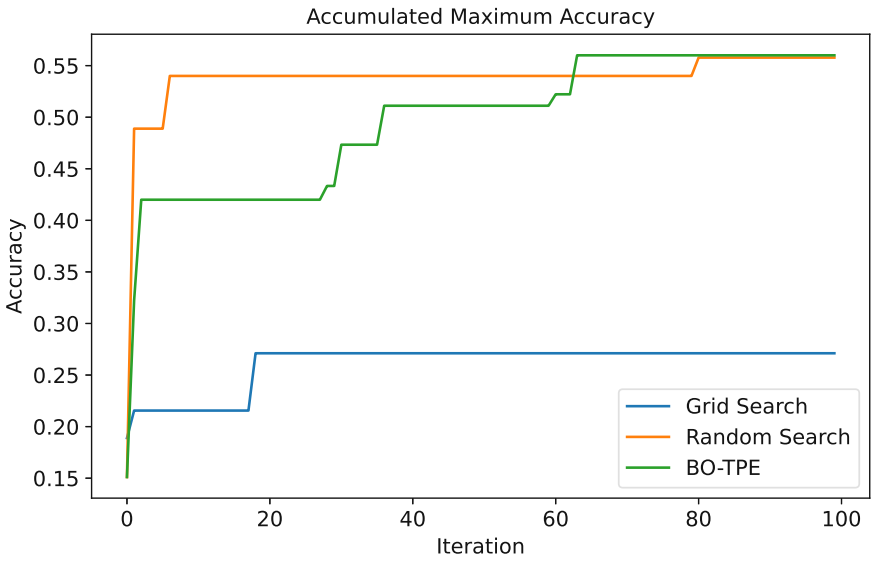


Figure 6.4: Accumulated maximum accuracy for the Forest Covertypes dataset.

next configurations. Given that we use 20 threads in parallel, we would therefore expect to see improvements of the maximum accuracy roughly every 20 iterations. Figure 6.4 confirms this hypothesis and shows how the informed search of BO-TPE is able to continuously improve the accuracy over the iterations. It can be seen that a relatively good configuration is found at the beginning and the first improvements happened between iteration 20 and 40. The search then continued until the best configuration is found after 60 iterations.

Overall, it can be seen that, similar to the Iris dataset, BO-TPE was even able to find the highest accuracy while being able to continuously improve the maximum accuracy.

Runtime

In Figure 6.3b we visualize the runtime of each hyperparameter tuning approach for the Forest Covertypes dataset using a box plot. It can be seen that the median runtime for all tuning approaches is almost similar, with RS being slightly higher. However, is can also be seen that there are outliers across all tuning approaches, with a significantly higher runtime compared to the median. Especially for RS these outliers show very log runtimes that are five to ten times higher than the median runtime. Similar to the outliers for the Iris dataset, we analyzed these runs to find some correlations among the hyperparameters. Among the 10 longest runs, we found that all runs used the SLSQP optimizer and the EfficientSU2 ansatz. However, as it can be seen in Chapter 6.4, both the SLSQP optimizer and the EfficientSU2 ansatz show the longest overall runtime. We could not find any other correlations and therefore believe that the significantly increased runtimes could be again a result of the barren plateau phenomenon.

Given the significantly higher runtime of the outliers, we additionally analyze their impact on the overall runtime. The different parallelization configurations make it hard to deduce the total runtime from the individual runtimes because the evaluations are randomly allocated to the available threads. To still analyze the impact of the outliers, we kept track of the total runtime after each finished evaluation. For RS, it can be seen in Figure 6.3b that there are five outliers with significantly higher runtimes. The evaluation of the fastest 95 configurations without the outliers lasted 22 hours and 51 minutes. The fastest 96 configurations already lasted over 39 hours, with the total runtime increasing to nearly 60 hours until all 100 configurations are finishes. Consequently, the outliers increased the total runtime of RS by over 160%, resulting in a total time consumption that is more than two and a half times higher than without outliers. In the case of BO-TPE there is a single outlier that increased the runtime from 25 hours and 30 minutes to 37 hours for the complete evaluation. This is an increase in time of over 45% which is significantly less than for RS but still a considerable increase.

6.2.3 KDD Cup 1999

In this section, we further analyze the results for the KDD Cup 1999 dataset. We again start by analyzing the accuracy and finish with the runtime evaluation.

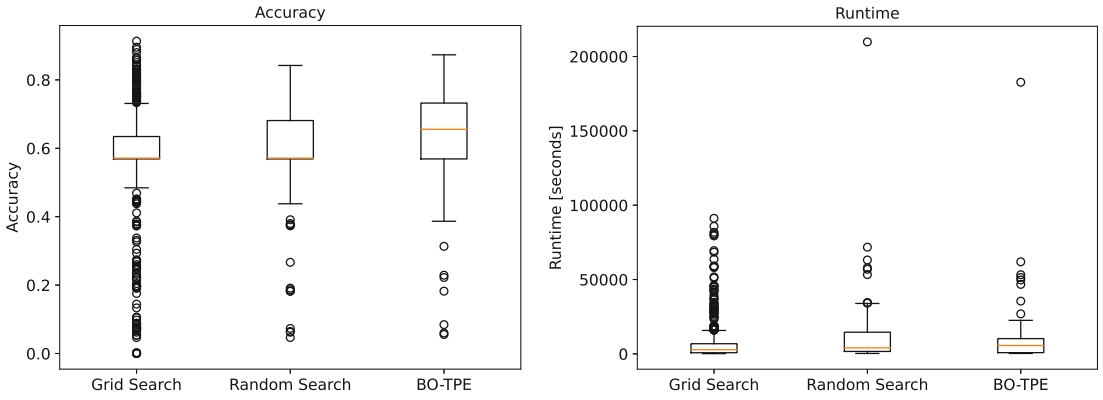
Accuracy

Table 6.2 shows that for the KDD Cup 1999 dataset, the accuracy across the different hyperparameter tuning approaches varies more than for the other datasets. GS was able to achieve the highest accuracy of 91.33% followed by BO-TPE with 87.33% and RS with 84.22%. In Table 6.5 we again listed the hyperparameter configuration of the runs with the best accuracy. This time there is only a single configuration for each tuning approach which vary only in the selected number of repetitions for the feature map and the entanglement strategy of the ansatz. Since the found configurations are almost identical,

Algo.	Dim. Red.	Feature Map	Reps	Ansatz	Ent.	Optimizer	Max. Iter.	Time
GS	PCA	ZFeatureMap	1	EfficientSU2	linear	SLSQP	100	17778s
RS	PCA	ZFeatureMap	5	EfficientSU2	linear	SLSQP	100	56876s
TPE	PCA	ZFeatureMap	1	EfficientSU2	full	SLSQP	100	51352s

Algo. = Hyperparameter Tuning Algorithm, Dim. Red. = Dimensionality Reduction Method, Ent. = Entanglement Strategy, Max. Iter. = Maximum Number of Iterations

Table 6.5: Best Hyperparameter Configurations for the KDD Cup 1999 Dataset.



(a) Box plot of the Accuracy.

(b) Box plot of the Runtime.

Figure 6.5: Accuracy and Runtime of the KDD Cup 1999 dataset.

we again assume that the difference in accuracy stems from the random initialization used for the optimizer and the deviation that naturally occurs from cross-validation.

Figure 6.5a shows the box plot of the accuracy for the KDD Cup 1999 dataset. It can be seen that the media accuracy is much higher than for the previous datasets and once again BO-TPE was able to achieve the highest median accuracy out of all hyperparameter tuning approaches. The overall boxes look very similar, with GS having the most outliers due to the higher number of individual evaluations.

In Figure 6.6 we illustrated the accumulated maximum accuracy over the iterations for the KDD Cup 1999 dataset. We again focus on the first 100 iterations for GS and show all 100 evaluations for the RS and BO-TPE approach. In contrast to the Forest Covertype dataset where RS found a good configuration very early, we can see that for this dataset RS continuously improved over the 100 iterations. BO-TPE found a better configuration at the start, but we again see that improvements happened in the first 20 iterations, between 40, 60 and finally after 80 iterations. This again aligns with the parallelization configuration of 20 threads and shows that BO-TPE is able to continuously improve the accuracy also for the KDD Cup 1999 dataset. However, also GS was able to continuously

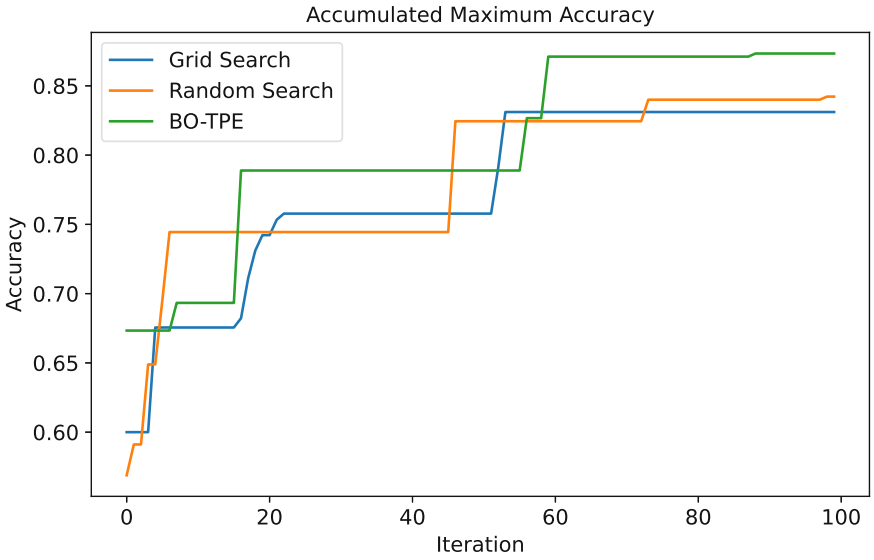


Figure 6.6: Accumulated maximum accuracy for the KDD Cup 1999 dataset.

improve over the first 100 iterations and found its maximum accuracy after about 200 iterations.

Overall, it can be seen that both of the budget based approaches were able to find very good configurations that are almost identical with the best configuration found by GS but without evaluating the complete search space.

Runtime

Figure 6.5b shows the runtime of each hyperparameter tuning approach for the KDD Cup 1999 dataset using a box plot. The median runtime across all approaches is almost equal, with the median of BO-TPE being slightly higher. However, similar to the Forest Covertypes dataset, there are outliers with a significantly higher runtime compared to the median. Especially for RS and BO-TPE there is one outlier that that has a much longer runtime than all the others. We again analyzed these outliers for correlations among the hyperparameters, and could only find the optimizer to be SLSQP for the top 10 longest runs. However, as it can be seen in Chapter 6.4, the SLSQP optimizer generally takes the longest among all optimizers in our selection. The other hyperparameters are different and show a similar distribution as the remaining runs. As already explained for the Iris and Forest Covertype dataset, we believe that the significantly increased runtimes are a result of the barren plateau phenomenon.

Since there are again outliers with a significantly higher runtime for RS and BO-TPE, we again analyze their impact on the overall runtime. In Figure 6.5b it can be seen that there is a single outlier for RS and BO-TPE each. For RS, the evaluation of the fastest 99 configurations without the outlier lasted 20 hours but well over 58 hours until all

Dataset	Tuning Algorithm	Threads	Accuracy	Time (hh:mm:ss)
Iris	BO-TPE	100	59.86%	00:21:11
		20	61.22%	00:28:09
Forest Covertype	BO-TPE	100	49.11%	47:56:01
		20	56.00%	37:06:35
KDD Cup 1999	BO-TPE	100	80.44%	38:23:17
		20	87.33%	57:24:05

Table 6.6: Performance of different parallelization Configurations for BO-TPE.

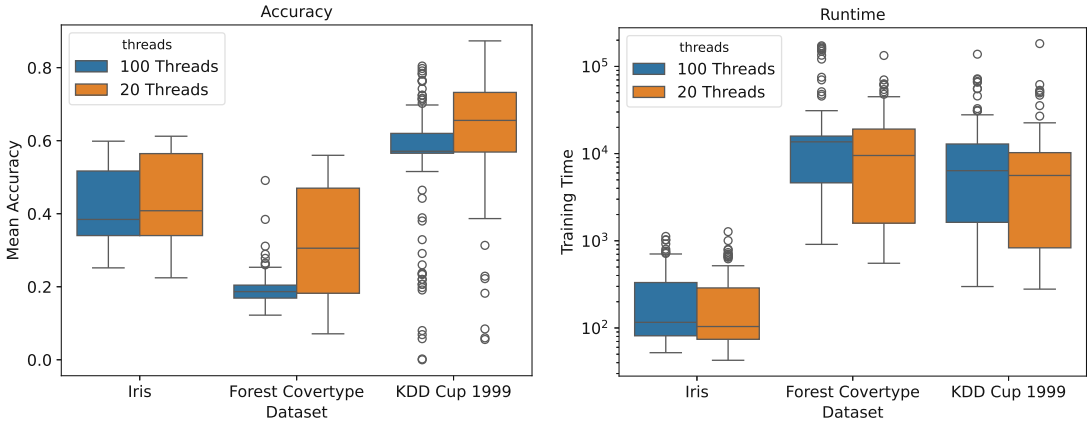
configurations finished. This is a very drastic increase of more than 190% resulting in a nearly three times higher time consumption for just a single evaluation without an increase in accuracy. A similar increase can be seen for BO-TPE where the single outlier increased the runtime from more than 21 hours to over 57 hours. This is an increase in time of over 170% without improving the accuracy.

6.3 Parallelization Analysis

For the performance evaluation in Section 6.2 we used 20 threads for BO-TPE while we used 100 threads for RS. In the following, we want to justify this decision by providing additional insights into the parallelization configuration. However, because of the extensive resource consumption and runtimes, we could not test multiple parallelization configurations in a large scale for all our datasets. We therefore tested them in a smaller scale and found 20 threads to provide a good overall performance.

To justify our selection, we show the performance improvement by comparing our configuration of 20 threads with a full parallelization using 100 threads. In Table 6.6 the accuracy and runtimes of these two configurations are listed for all the three datasets. It can be seen that the accuracy for 20 threads is significantly better across all datasets. In Figure 6.7a we additionally visualize the accuracy for both configurations using a box plot. The illustration shows that also the median accuracy is significantly higher for 20 threads, with the median for the Forest Covertypes and KDD Cup 1999 dataset even being beyond the box for the 100 thread configuration. However, this is not surprising, as it only proofs that the informed search of BO-TPE needs multiple iterations to suggest promising configurations based on the previous results.

In Table 6.6 we also list the overall runtime for each configuration. We can see that for the Forest Covertype dataset the configuration using 20 threads is faster, but for the KDD Cup 1999 dataset the 100 threads configurations is much faster. These runtimes appear to be inconsistent, which could again be caused by outliers with significantly longer runtimes that negatively affect the overall runtime. To further analyze this behavior, we show the individual runtimes in the box plot of Figure 6.7b using a logarithmic scale



(a) Box plot of the Accuracy.

(b) Box plot of the Runtime.

Figure 6.7: Comparison of different parallelization Configurations for all three Datasets.

to also fit the much lower runtimes of the Iris dataset. It can be seen that the median runtimes of the different configurations are very similar, and tend to be slightly lower for the 20-thread configuration. The figure therefore confirms our hypothesis of outliers being the reason for the inconsistent overall runtimes.

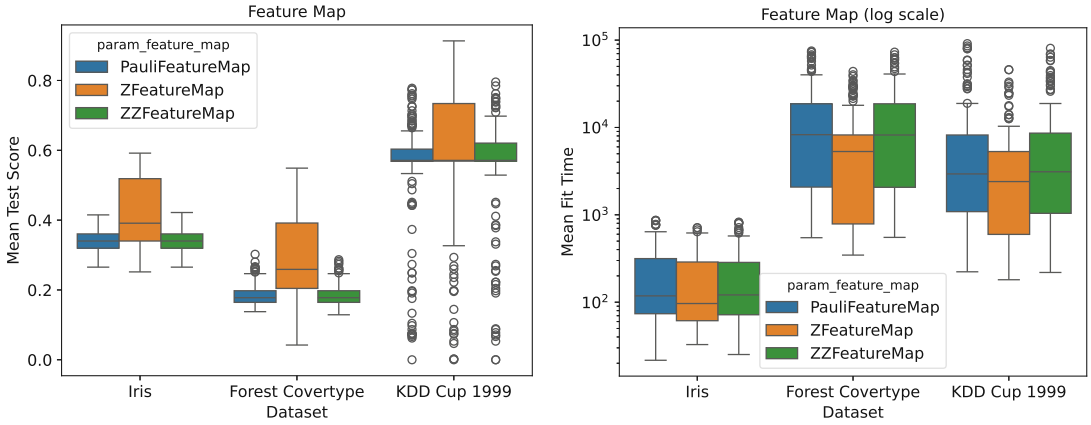
Overall, it can be seen that 20 threads provide a much higher accuracy without a significant negative impact on the runtime.

6.4 Hyperparameter Evaluation

Finally, we would also like to gain some insight into the performance impact of the different hyperparameters. In the following we therefore analyze which feature map, ansatz, optimizer and dimensionality reduction method has impact on the accuracy and runtime across the three datasets. We use the data from the GS runs to cover the complete hyperparameter space.

6.4.1 Feature Map

We start by analyzing the different feature map types. To evaluate their impact on the accuracy, we visualize the runs with the corresponding feature map types using a box plot in Figure 6.8a. We combined the evaluation of all three datasets into a single illustration and highlight the feature map types with different colors. We find that the Z feature map performs best across all datasets, with the median accuracy being significantly higher for the Iris and Forest Covertype dataset and almost equal for the KDD Cup 1999 dataset. Although the median accuracy of the Z feature map is not significantly higher for the KDD Cup 1999 dataset, it can be seen that there are more evaluations with high accuracy compared to the other feature maps types. Furthermore, this is consistent with the findings of Section 6.2 where the Z feature map is listed in all the top configurations with



(a) Box plot of the Accuracy.

(b) Box plot of the Runtime.

Figure 6.8: Performance of different Feature Map Types for all three Datasets.

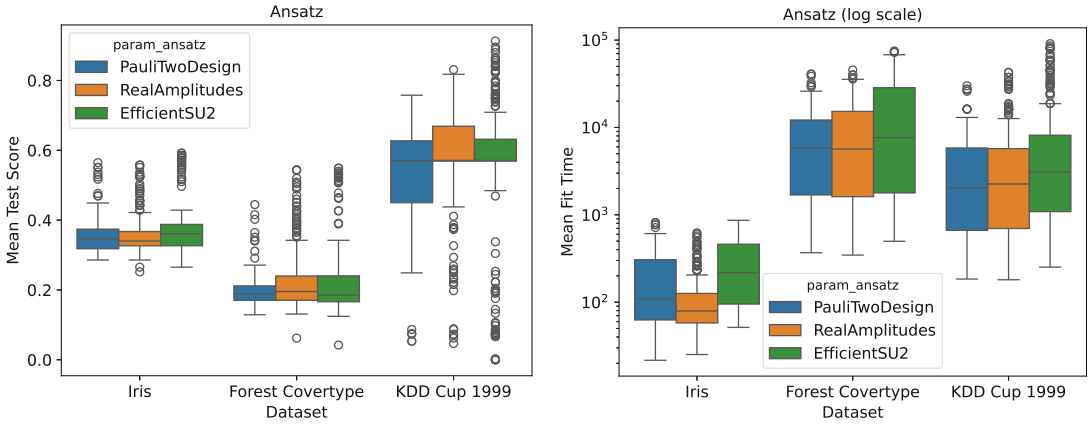
the best accuracy. The Pauli and ZZ feature maps perform almost identically, and we see no difference in accuracy for them, with the box plots also looking almost identical.

To analyze the runtime, we use again a box plot with a logarithmic scale to visualize the runtime of evaluations using a particular feature map in Figure 6.8b. Overall, the runtimes seem to be consistent across all feature map types and datasets. However, it can be seen that the median runtime and the interquartile ranges of the Z feature map is slightly lower for all datasets, indicating better general runtimes. We therefore find the Z feature map to be the best choice, providing lower runtimes while achieving the highest accuracy.

6.4.2 Ansatz

We continue by analyzing the performance of the different ansatzes in our hyperparameter selection. In Figure 6.9a we represent the accuracy for the runs with the corresponding ansatz. The box plot shows that there is no significant difference in accuracy for either of the ansatzes with the median accuracy being almost equal for all types across the three datasets. From the top configurations discussed in Section 6.2, we know that most of the top configurations used the EfficientSU2 ansatz. In the box plot, this is confirmed by the outliers of the EfficientSU2 ansatz that show a better accuracy for the Iris and the KDD Cup 1999 dataset. For the Forest Covertype dataset, the RealAmplitudes ansatz performs equally to the EfficientSU2 ansatz and can therefore also be found among the top configurations in Table 6.4.

Figure 6.9b shows that there is no significant difference in the runtimes of the respective ansatzes. However, it can be observed that the median runtime of the EfficientSU2 ansatz is slightly higher for all datasets. The RealAmplitudes ansatz is the fastest for the Iris and Forest Covertype dataset, while the PauliTwoDesign ansatz is the fastest for the



(a) Box plot of the Accuracy.

(b) Box plot of the Runtime.

Figure 6.9: Performance of different Ansatz Types for all three Datasets.

KDD Cup 1999 dataset.

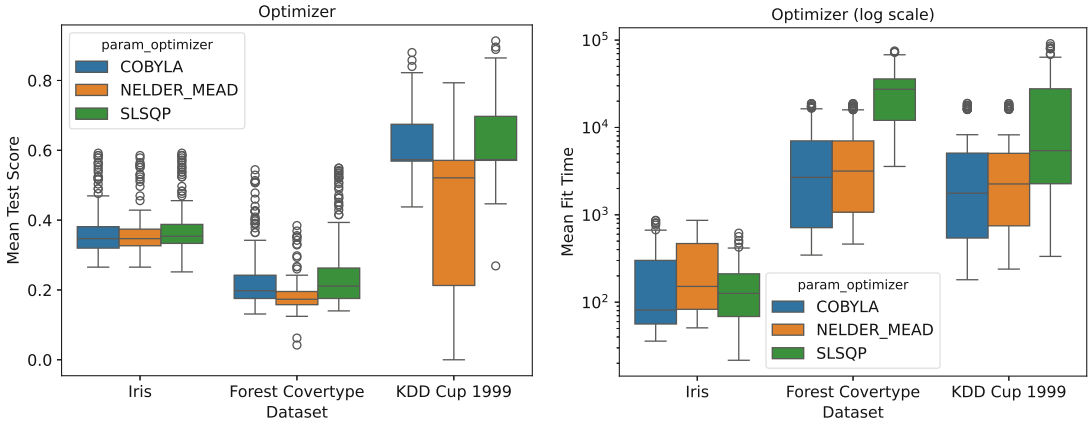
Overall, the performance impact of the ansatz is rather small and there are no significant difference between the ansatz types for our datasets. The EfficientSU2 ansatz provides the highest accuracy but also the highest runtimes in our experiment.

6.4.3 Optimizer

For the optimizer, we illustrate the accuracy in Figure 6.10a. The median accuracy is very similar for all optimizers, and we could not find any significant difference for the Iris and Forest Covertype dataset. However, for the KDD Cup 1999 dataset it can be seen that the lower quartile for the Nelder Mead optimizer is significantly larger, indicating a higher number of low accuracy runs. Moreover, the median accuracy for Nelder Mead is the lowest for the Forest Covertype and KDD Cup 1999 dataset. Regarding the accuracy, the Nelder Mead optimizer therefore seems to be the worst choice for our use cases. When we look at the top configurations listed in Section 6.2, we find the SLSQP optimizer to be the best choice for the Forest Covertype and KDD Cup 1999 dataset and the COBYLA optimizer for the Iris dataset.

The box plot of the runtime for each optimizer can be found in Figure 6.10b. For the Forest Covertype and KDD Cup 1999 datasets, the SLSQP optimizer shows significantly higher runtimes than the other optimizers. This can be seen by the higher median runtime and the box of SLSQP surpassing the others. Interestingly, for the Iris dataset the SLSQP optimizer shows good runtimes with the median being lower than for Nelder Mead. However, we find the COBYLA optimizer to be consistently the fastest optimizer across all datasets.

Considering the almost equal accuracy with the SLSQP optimizer and the best runtime performance, the COBYLA optimizer seems to be the most promising optimizer to choose



(a) Box plot of the Accuracy.

(b) Box plot of the Runtime.

Figure 6.10: Performance of different Optimizers for all three Datasets.

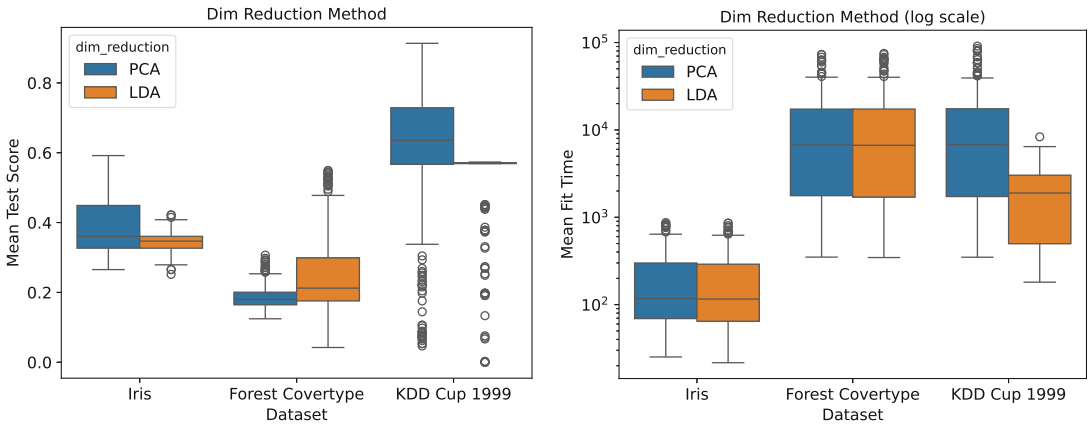
for our use cases. However, if the runtime is not that important, the SLSQP optimizer is also a valid choice.

6.4.4 Dimensionality Reduction

To analyze the impact of the dimensionality reduction method on the accuracy, we again use a box plot to visualize the corresponding runs in Figure 6.11a. We find LDA to provide significantly higher accuracy for the Forest Covertype dataset, while PCA achieves significantly higher accuracy for the Iris and KDD Cup 1999 dataset. These findings also correspond to the top configurations listed in Section 6.2, where LDA can be found among all top configurations for the Forest Covertype dataset, while PCA is used for the other two datasets. Interestingly, this is contrary to the results of Mancilla and Pere in [MP22] where they found LDA to perform better across two datasets in the finance domain. This shows how problem dependent the hyperparameters, including the dimensionality reduction, are for QML.

In Figure 6.11b we show the impact of the dimensionality reduction method on the runtime. For the Iris and Forest Covertype dataset, both methods perform nearly identical but for the KDD Cup 1999 dataset LDA seems to achieve shorter runtimes. However, since the accuracy of LDA was relatively poor for the KDD Cup 1999 dataset, it is still not the best choice.

Overall, we find PCA to be the best option for the Iris and KDD Cup 1999 dataset and LDA to work best for the Forest Covertype dataset.



(a) Box plot of the Accuracy.

(b) Box plot of the Runtime.

Figure 6.11: Performance of different Dimensionality Reduction Approaches.

Hyperparameter	**Configuration**	**Subparameter**	**Values**
Feature reduction	PCA LDA	/	/
Feature Map	PauliFeatureMap ZFeatureMap ZZFeatureMap	Repetitions	[1-5]
Ansatz	PauliTwoDesign RealAmplitudes EfficientSU2	/ Entanglement	/ [full, linear, sca, reverse_linear, circular]
Optimizer	COBYLA SLSQP NELDER_MEAD	Max. Iterations	[100-1000]

Table 6.7: Extended Hyperparameter Configuration Space using discrete variables.

Dataset	Tuning Algorithm	Accuracy	Accuracy (large)	Time	Time (large)
Iris	RS	57.82%	58.50%	00:27:04	38:28
	BO-TPE	61.22%	59.86%	00:28:09	29:44
Forest Covertype	RS	55.78%	60.44%	59:51:48	51:54:36
	BO-TPE	56.00%	58.66%	37:06:35	24:00:17
KDD Cup 1999	RS	84.22%	89.11%	58:20:19	22:53:56
	BO-TPE	87.33%	92.66%	57:24:05	44:20:03

Table 6.8: Accuracy and Runtime of the Normal and Extended Search Space.

6.5 Extended Search Space Evaluation

One of the main advantages of RS and BO-TPE is that they can be used for very large search spaces because of their fixed budget of evaluations. We therefore extended the search space to analyze how these hyperparameter tuning approaches perform for our QML use case. In Table 6.7 we show the extended hyperparameter search space where we modeled the repetitions of the feature map and the maximum iterations of the optimizer as discrete variables. Additionally, we added the entanglement strategies *reverse_linear* and *circular* for the ansatz. These relatively small adjustments increase the hyperparameter search space from 504 to 891.000 possible configurations. Consequently, it is not possible to exhaustively evaluate this search space with GS. However, with RS and BO-TPE we are still able to evaluate this hyperparameter search space for all datasets.

6.5.1 Accuracy

Table 6.8 shows the accuracy and runtime of the normal and extended search space. It can be seen that the maximum accuracy for the larger search space is higher across all dataset and hyperparameter tuning combinations, except for BO-TPE with the Iris dataset. For the Iris dataset, the maximum accuracy could be improved from 57.82% to 58.50% for RS but BO-TPE performed worse with the larger search space showing an accuracy of 59.86% instead of 61.22%. However, for the Forest Covertype dataset both tuning approaches achieved a higher maximum accuracy with RS showing an accuracy of 60.4% which is the highest we achieved for this dataset across all our experiments. Compared to the normal search space, this is an increase of more than 4%. Similarly, for the KDD Cup 1999 dataset, both tuning approaches could increase the maximum accuracy by using the larger search space. Here BO-TPE achieves the highers accuracy of 92.66% which is more than 5% higher than the maximum accuracy of the normal search space.

In Figure 6.12 we visually compare the accuracy results of the extended search space to the normal search space and show the mean accuracy as well as the maximum accuracy. We use a bar plot instead of a box plot to improve readability, since we compare multiple

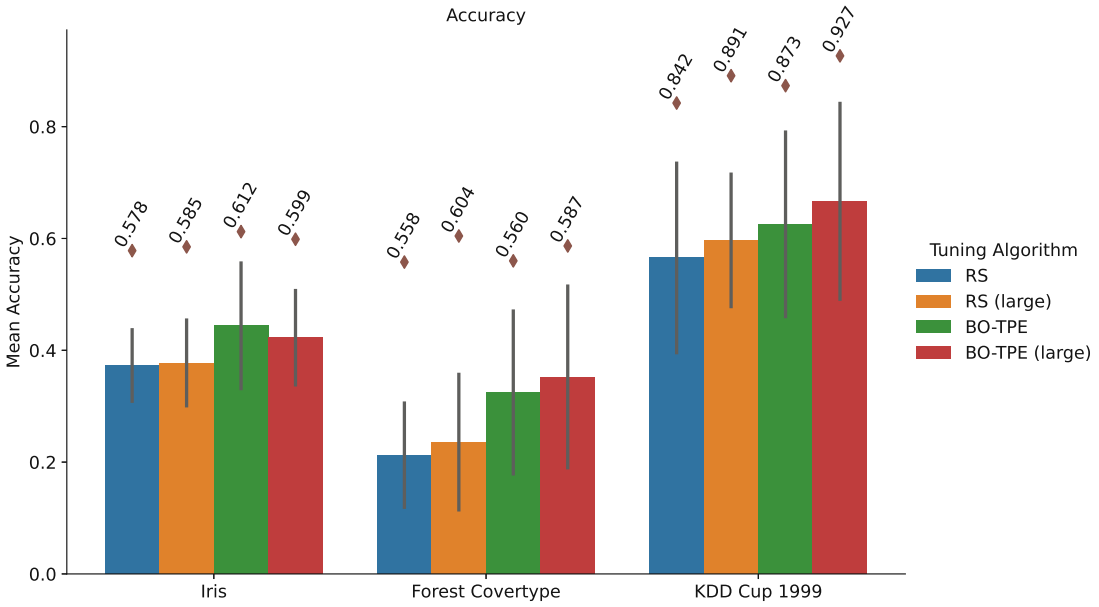


Figure 6.12: Accuracy for the extended Hyperparameter Search Space.

results inside a single figure. The bars show the mean accuracy and the black line indicates the standard deviation. Additionally, we added the maximum accuracy for each category, indicated by the labeled diamond on top of each bar.

Overall, we can see that the results for the mean accuracy are similar to those for the maximum accuracy. The large search space consistently shows a higher mean accuracy for RS across all datasets. For BO-TPE we again find the Iris dataset to be an exception where the larger hyperparameter space could not improve the mean accuracy similar to the maximum accuracy. However, for the Forest Covertype and KDD Cup 1999 dataset, the larger search space provides a higher mean accuracy.

6.5.2 Runtime

In Table 6.8 we also compare the runtime of the normal and the extended search space. The results are again a bit inconsistent, since the runtime of the extended search space is higher for the Iris dataset but lower for the Forest Covertype and KDD Cup 1999 dataset. For the Iris dataset, the runtime of both hyperparameter tuning approaches is higher compared to the normal search space, although the increase for BO-TPE with less than two minutes is very small. The runtimes of the Forest Covertype dataset are both lower for the extended search space, making BO-TPE for the larger search space the fastest evaluation of this dataset with a runtime of 24 hours. Similarly, the runtimes for the KDD Cup 1999 dataset are also lower than the normal search space for both tuning approaches. However, for this dataset RS is actually the faster approach taking only 22 hours which is half the time that BO-TPE needs. When it comes to the runtime, we

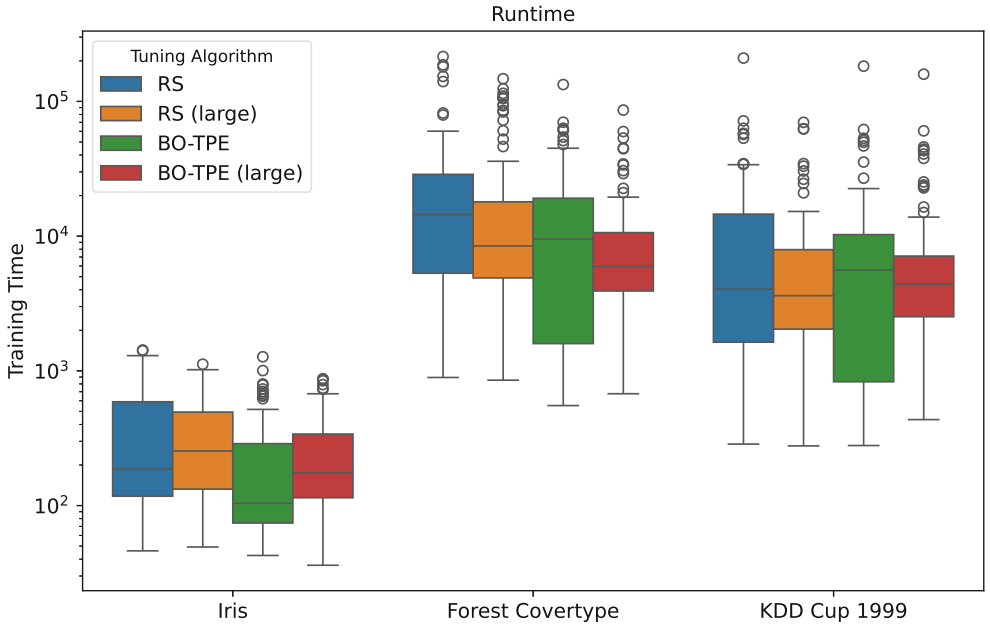


Figure 6.13: Runtimes for the extended Hyperparameter Search Space.

therefore find BO-TPE to be the better choice for the Iris and Forest Covertype dataset but RS performs better for the KDD Cup 1999 dataset.

We believe that the differences in runtimes are again a result of outliers introduced by the barren plateau phenomenon, as discussed in Section 6.2. In Figure 6.13 we therefore illustrate the runtimes of the individual evaluations using a box plot to analyze and compare the larger with the normal search space.

We find that similar to the overall runtime also the median runtime of the Iris dataset is higher for the larger search space across both hyperparameter tuning approaches. For the Forest Covertype and KDD Cup 1999 dataset on the other hand, we see that the median runtimes are lower for both tuning approaches, which again corresponds to the total runtimes. Moreover, it can be seen that the upper whiskers and outliers for the Forest Covertype and KDD Cup 1999 dataset are consistently lower for the larger search space. The lower total runtime can therefore be explained by the lower median runtime and less significant outliers.

Overall, we find that using a larger hyperparameter space improves the accuracy results of RS and BO-TPE without increasing the runtime. For our experiments, most of the runtimes were even lower than for the regular search space. However, we find that the barren plateau phenomenon impacts all our experiments and introduces outliers with a significantly higher runtime.

CHAPTER 7

Conclusion

In this thesis, we investigated the applicability and performance of hyperparameter tuning algorithms for QML classification tasks. Based on a literature review, we first determined the most important hyperparameters for VQAs to answer *RQ 1*. We could identify the optimizer, ansatz and feature map as being the main hyperparameters impacting the performance of VQAs. In our hyperparameter evaluation, we then analyzed the performance impact of the individual hyperparameters in our experiments. Generally, it can be seen that the best hyperparameter configurations are very problem dependent and vary across the different datasets. The feature map is an exception where we found the Z feature map to be the best choice for all datasets, providing a significant higher accuracy and lower runtimes compared to the other feature maps. For the ansatz, we found the EfficientSU2 to provide good accuracy, while the RealAmplitudes ansatz shows lower runtimes with comparable accuracy. The same is true for the optimizer, where we found the SLSQP optimizer to provide the highest accuracy with higher runtimes than the COBYLA optimizer that offers similar accuracy. The dimensionality reduction method on the other hand seems to be very dataset dependent since PCA provides significantly better accuracy for the Iris and KDD Cup 1999 dataset while LDA is best suited for the Forest Covertype dataset. The fact that these hyperparameters are highly problem dependent highlights the great need for automatic hyperparameter tuning to fully take advantage of the underlying VQA.

RQ 2 was answered by reviewing existing hyperparameter tuning approaches and assessing their applicability for quantum use cases. We found SMAC and BO-TPE to be especially well suited for the mostly categorical hyperparameter space of VQAs. RS is also a very promising approach because of its simplicity and excellent parallelization capabilities. We therefore focused on studying the performance of BO-TPE and RS for different QML classification tasks and included GS as an exhaustive baseline.

To evaluate the performance of these automatic hyperparameter tuning algorithms and answer *RQ 3*, we applied BO-TPE and RS to three different datasets and compared them

with the exhaustive results of GS. We found that BO-TPE consistently outperformed RS in terms of accuracy across all datasets. For the Iris and Forest Covertype dataset, BO-TPE even showed higher accuracy results than the exhaustive GS approach. We attribute this to the standard deviation from the cross validation, since the found configurations are nearly identical. Additionally, the higher median accuracy of BO-TPE for all datasets indicates that the guided search of BO-TPE helped to find more promising hyperparameter configurations with higher accuracy while spending less time evaluating poor configurations.

Surprisingly, the results for the runtime are very inconsistent, even though we would have expected the budget based tuning approaches BO-TPE and RS to significantly outperform the exhaustive GS approach across all datasets. However, due to outliers, the budget based approaches were only faster for the Iris and Forest Covertype dataset while taking more time than GS for the KDD Cup 1999 dataset. We could not find any correlations between the configurations that led to these outliers, and therefore attribute the significantly increased runtimes to the barren plateau phenomenon. Given that it only affects a small portion of the evaluations, we assume that its occurrence is related to the random initialization of the optimizer. However, we kept track of the total runtime after each finished evaluation and find the budget based approaches to be significantly faster than GS when the outliers are excluded. For the Forest Covertype dataset, the runtime is increased by over 160% for RS and over 45% for BO-TPE. In the case of the KDD Cup 1999 dataset, a single outlier increased the runtime of RS by over 190% and for BO-TPE by over 170%. It therefore makes sense to implement some sort of early stopping mechanism in any QML project that only has a certain time budget to cut out these outliers that do not increase accuracy. However, to the best of our knowledge, none of the tuning approaches in this thesis supports such an early stopping mechanism out of the box.

All the three hyperparameter tuning approaches studied in this thesis support parallelization. For GS and RS full parallelization can be used since the individual configurations can be evaluated independently of each other. However, since BO-TPE uses the result of previous evaluations to select subsequent hyperparameter configurations, we analyzed different parallelization configurations. We found a parallelization ratio of 20% of the evaluation budget to provide significantly better results than full parallelization. Consequently, if the evaluation budget is set to 100 it makes sense to use up to 20 threads in parallel for BO-TPE.

Finally, we also studied the performance of the budget based tuning approaches for larger search spaces and found that both, RS and BO-TPE, handle larger search spaces very well. Increasing the search space by modeling numerical hyperparameters as discrete variables and adding additional categorical options improved the overall accuracy without increasing the total runtime. For the experiments in this thesis, the observed total runtime was even better for the larger than for the normal search space. This shows that RS and BO-TPE are both capable of handling very large search spaces and indicates that including all available hyperparameter options can increase accuracy.

7.1 Future Work

The findings of this thesis already give good insights into how automatic hyperparameter tuning algorithms can be applied to QML problems. However, during our research, we also identified opportunities to extend the results of our work. In the following, we give a brief summary of possible future work:

Hyperparameter Tuning Algorithms We have seen that budget based approaches are well suited and especially BO-TPE was able to consistently outperform the other approaches in our experiments. It would therefore be interesting to benchmark additional algorithms and compare their performance to BO-TPE. A very promising algorithm for future experiments would be SMAC, which is also based on BO and uses regression trees to predict promising configuration. Moreover, it would be interesting to see if metaheuristic algorithms such as GA or PSO can also be used for QML problems and what initialization strategies are well suited.

Barren Plateau Analysis During our experiments we also found that outliers, most likely caused by the barren plateau phenomenon, have a significant impact on runtime. It would therefore be interesting to further analyze this phenomenon and evaluate different initialization strategies or distributions. By manually specifying the initialization parameters, the evaluations that lead to extensive runtimes could be analyzed and searched for correlations. This could provide insights which initializations parameters are more likely to result in a barren plateaus.

Better Hardware In this thesis we simulated quantum computers since access to real quantum computers is very expensive and free execution plans offer only very limited computation time that is not sufficient for our experiments. However, we had to drastically reduce the features of our datasets and limit the training data because simulating a quantum computer is computationally very intensive. More powerful and reliable quantum computers that are accessible for research would eliminate these limitations and allow including more training data and features in future experiments.

Hyperparameters In this thesis, we identified the most important hyperparameters of VQAs and focused on tuning them. However, our findings also show that budgets based hyperparameter tuning algorithms can handle larger search spaces well. In future work, additional hyperparameters or further subparameters, such as the entanglement strategy of supported feature maps or the repetition count of the ansatz, could be included in the search space.

Datasets In our thesis, we used three different datasets to evaluate the performance of the different hyperparameter tuning algorithms. However, we have seen that the performance of these algorithms vary across different datasets and the optimal hyperparameters are very problem dependent. Evaluating the performance on additional datasets would therefore provide more insights and allow assessing the

performance of hyperparameter tuning algorithms in a more general manner. Future work could also include to adapt a more general benchmarking framework such as MQT Bench [QBW23] to evaluate the performance. Having such a general benchmarking framework for QML would increase the comparability of performance results.

List of Figures

List of Tables

Acronyms

AutoML Automated machine learning. 2

BO Bayesian Optimization. 31–34, 36, 49–53, 55–60, 66–71

BOHB Bayesian Optimization HyperBand. 33

CMA-ES Covariance Matrix Adaptation Evolutionary Strategy. 26

COBYLA Constrained Optimization BY Linear Approximations. 26, 47, 48, 63, 69

GA Genetic algorithm. 33, 71

GP Gaussian Process. 32, 34

GPU Graphics Processing Unit. 11

GS Grid Search. 30–32, 34–36, 45, 49–55, 57–59, 61, 66, 69, 70

HPO Hyperparameter Optimization. 2, 12

LDA Linear Discriminant Analysis. 43–45, 64, 69

ML Machine Learning. xi, xiii, 1–3, 5, 7, 10–15, 18, 20, 23, 24, 29–31, 36, 43

MSE Mean Square Error. 26

NISQ Noisy Intermediate-Scale Quantum. 6, 10, 11, 17, 18, 22, 23, 36, 37, 43, 44

PCA Principal Component Analysis. 43–45, 64, 69

PQC Parameterized Quantum Circuit. xi, xiii, 1–3, 15, 16, 18, 20, 21, 73

PSO Particle swarm optimization. 33, 34, 71

QAOA Quantum Approximate Optimization Algorithm. 18, 22

QEC Quantum Error Correction. 9–11, 14

QEM Quantum Error Mitigation. 10

QML Quantum Machine Learning. xi, xiii, 1–5, 10, 11, 13–18, 20, 23–25, 27, 29–37, 43, 64, 66, 69–71, 73, 75

QPU Quantum Processing Unit. 17

QSVM Quantum Support Vector Machine. 15, 27

qubit quantum bit. 6–10, 14, 17, 20, 22, 24, 26, 36, 37, 43, 44, 50, 73

RF Random forest. 32

RS Random Search. 3, 31, 32, 34, 36, 49–55, 57–60, 66–70

RSA Rivest–Shamir–Adleman. 6

SGD Stochastic Gradient Descent. 23

SLSQP Sequential Least Squares Quadratic Programming. 26, 47, 57, 59, 63, 64, 69

SMAC Sequential Model-based Algorithm Configuration. 32, 34, 69, 71

SOE Second-Order Expansion. 27

SPSA Simultaneous Perturbation Stochastic Approximation. 23, 26

SVD Singular value decomposition. 43

TPE Tree-structured Parzen Estimator. 32, 34, 36, 49–53, 55–60, 66–71

TPU Tensor Processing Unit. 11

VQA Variational Quantum Algorithm. xi, xiii, 1–4, 13–18, 20–27, 29–31, 34, 36, 47, 49, 69, 71, 75

VQE Variational Quantum Eigensolver. 18

Bibliography

[AAB+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, October 2019.

[AAR+18] Mohammad H. Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum boltzmann machine. *Phys. Rev. X*, 8:021050, May 2018.

[ABO97] D. Aharonov and M. Ben-Or. Fault-tolerant quantum computation with constant error. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC '97*, pages 176–188, El Paso, Texas, United States, 1997. ACM Press.

[AHCC22] Andrew Arrasmith, Zoë Holmes, M. Cerezo, and Patrick J. Coles. Equivalence of quantum barren plateaus to cost concentration and narrow gorges. *Quantum Science and Technology*, 7(4):045015, October 2022. arXiv:2104.05868 [quant-ph].

[BBBV97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, October 1997.

[BCLK+22] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S. Kottmann, Tim Menke, Wai-Keong Mok, Sukin Sim, Leong-Chuan Kwek, and Alán Aspuru-Guzik. Noisy intermediate-scale quantum (NISQ) algorithms. *Reviews of Modern Physics*, 94(1):015004, February 2022. arXiv:2101.08448 [cond-mat, physics:quant-ph].

[BGOC+22] Raúl Berganza Gómez, Corey O'Meara, Giorgio Cortiana, Christian B. Mendl, and Juan Bernabé-Moreno. Towards autoqml: A cloud-based automated circuit architecture search framework. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, pages 129–136, 2022.

[Bia21] Jacob Biamonte. Universal variational quantum computation. *Phys. Rev. A*, 103:L030401, Mar 2021.

[BIS+22] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M. Sohaib Alam, Guillermo Alonso-Linaje, B. Akash-Narayanan, Ali Asadi, Juan Miguel Arrazola, Utkarsh Azad, Sam Banning, Carsten Blank, Thomas R Bromley, Benjamin A. Cordier, Jack Ceroni, Alain Delgado, Olivia Di Matteo, Amintor Dusko, Tanya Garg, Diego Guala, Anthony Hayes, Ryan Hill, Aroosa Ijaz, Theodor Isacsson, David Ittah, Soran Jahangiri, Prateek Jain, Edward Jiang, Ankit Khandelwal, Korbinian Kottmann, Robert A. Lang, Christina Lee, Thomas Loke, Angus Lowe, Keri McKiernan, Johannes Jakob Meyer, J. A. Montañez-Barrera, Romain Moyard, Zeyue Niu, Lee James O'Riordan, Steven Oud, Ashish Panigrahi, Chae-Yeun Park, Daniel Polatajko, Nicolás Quesada, Chase Roberts, Nahum Sá, Isidor Schoch, Borun Shi, Shuli Shu, Sukin Sim, Arshpreet Singh, Ingrid Strandberg, Jay Soni, Antal Száva, Slimane Thabet, Rodrigo A. Vargas-Hernández, Trevor Vincent, Nicola Vitucci, Maurice Weber, David Wierichs, Roeland Wiersema, Moritz Willmann, Vincent Wong, Shaoming Zhang, and Nathan Killoran. Pennylane: Automatic differentiation of hybrid quantum-classical computations. `https://pennylane.ai/`, 2022. Last accessed March 09, 2024.

[BK21] Lennart Bittel and Martin Kliesch. Training variational quantum algorithms is NP-hard. *Physical Review Letters*, 127(12):120502, September 2021. arXiv:2101.07267 [quant-ph].

[Bla98] Jock Blackard. Covertype. UCI Machine Learning Repository, 1998. DOI: https://doi.org/10.24432/C50K5N.

[BMWV+23] Xavier Bonet-Monroig, Hao Wang, Diederick Vermetten, Bruno Senjean, Charles Moussa, Thomas Bäck, Vedran Dunjko, and Thomas E. O'Brien. Performance comparison of optimization methods on variational quantum algorithms. *Physical Review A*, 107(3), mar 2023.

[Bro19] Michael Brooks. Beyond quantum supremacy: the hunt for useful quantum computers. *Nature*, 574(7776):19–21, October 2019.

[BWP+17] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, sep 2017.

[BYC13] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 115–123, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[CAB+21] 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. arXiv:2012.09265 [quant-ph, stat].

[Cas23] Davide Castelvecchi. IBM releases first-ever 1,000-qubit quantum chip. *Nature*, 624(7991):238–238, December 2023.

[CBB+23] Zhenyu Cai, Ryan Babbush, Simon C. Benjamin, Suguru Endo, William J. Huggins, Ying Li, Jarrod R. McClean, and Thomas E. O'Brien. Quantum Error Mitigation. *Reviews of Modern Physics*, 95(4):045005, December 2023. arXiv:2210.00921 [quant-ph].

[CCL19] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, August 2019.

[CDMB+22] Sandeep Suresh Cranganore, Vincenzo De Maio, Ivona Brandic, Tu Mai Anh Do, and Ewa Deelman. Molecular dynamics workflow decomposition for hybrid classic/quantum systems. In *2022 IEEE 18th International Conference on e-Science (e-Science)*, pages 346–356, 2022.

[Con] Qiskit Contributors. Qiskit circuit library. `https://qiskit.org/documentation/apidoc/circuit_library.html`. Last accessed October 27, 2023.

[CVH+22] M. Cerezo, Guillaume Verdon, Hsin-Yuan Huang, Lukasz Cincio, and Patrick J. Coles. Challenges and opportunities in quantum machine learning. *Nature Computational Science*, 2(9):567–576, September 2022.

[Dev] Cirq Developers. Cirq google quantum ai. `https://quantumai.google/cirq`. Last accessed March 09, 2024.

[Dev24] Cirq Developers. Cirq, May 2024.

[DHY+22] 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, 2022.

[DMAB22] Vincenzo De Maio, Atakan Aral, and Ivona Brandic. A roadmap to post-moore era for distributed systems. In *Proceedings of the 2022 Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating Algorithms for Distributed Systems*, ApPLIED '22, page 30–34, New York, NY, USA, 2022. Association for Computing Machinery.

[DTB16] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Quantum-enhanced machine learning. *Phys. Rev. Lett.*, 117:130501, Sep 2016.

[Fis36] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[GI19] Laszlo Gyongyosi and Sandor Imre. A survey on quantum computing technology. *Computer Science Review*, 31:51–71, 2019.

[GKS+22] Sukhpal Singh Gill, Adarsh Kumar, Harvinder Singh, Manmeet Singh, Kamalpreet Kaur, Muhammad Usman, and Rajkumar Buyya. Quantum computing: A taxonomy, systematic review and future directions. *Software: Practice and Experience*, 52(1):66–114, 2022.

[Got09] Daniel Gottesman. An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation, April 2009. arXiv:0904.2557 [quant-ph].

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[HAS+21] Muhammad Minoar Hossain, Mohammed Sowket Ali, Reshma Ahmed Swarna, Md. Mahmodul Hasan, Nahida Habib, Md. Wahidur Rahman, Mir Mohammad Azad, and Mohammad Motiur Rahman. Analyzing the effect of feature mapping techniques along with the circuit depth in quantum supervised learning by utilizing quantum support vector machine. In *2021 24th International Conference on Computer and Information Technology (ICCIT)*, pages 1–5, Dhaka, Bangladesh, December 2021. IEEE.

[HCT+19] 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. arXiv:1804.11326 [quant-ph, stat].

[HDMB24] Sabrina Herbst, Vincenzo De Maio, and Ivona Brandic. On Optimizing Hyperparameters for Quantum Neural Networks, March 2024. arXiv:2403.18579 [cs].

[HIP+21] Ciaran Hughes, Joshua Isaacson, Anastasia Perry, Ranbel F. Sun, and Jessica Turner. *Quantum Computing for the Quantum Curious*. Springer International Publishing, Cham, 2021.

[HKV19] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer International Publishing, Cham, 2019.

[Jor] Stephen Jordan. Quantum algorithm zoo. `https://quantumalgorithmzoo.org/`. Last accessed January 29, 2024.

[KB17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2017.

[KRK20] Tariq M. Khan and Antonio Robles-Kelly. Machine learning: Quantum vs classical. *IEEE Access*, 8:219275–219294, 2020.

[KW17] Mária Kieferová and Nathan Wiebe. Tomography and generative training with quantum boltzmann machines. *Phys. Rev. A*, 96:062327, Dec 2017.

[KWS16] Ashish Kapoor, Nathan Wiebe, and Krysta Svore. Quantum perceptron models. *Advances in neural information processing systems*, 29, 2016.

[LC20] Ryan LaRose and Brian Coyle. Robust data encodings for quantum classifiers. *Physical Review A*, 102(3):032420, September 2020.

[lDa] Scikit learn Developers. Scikit-learn iris dataset. `https://scikit-learn.org/stable/datasets/toy_dataset.html#iris-dataset`. Last accessed April 03, 2024.

[lDb] Scikit learn Developers. Scikit-learn kddcup 99 dataset. `https://scikit-learn.org/stable/datasets/real_world.html#kddcup-99-dataset`. Last accessed April 05, 2024.

[LHF+00] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, 2000. Recent Advances in Intrusion Detection Systems.

[LMR14] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, July 2014.

[LYC14] Guang Hao Low, Theodore J. Yoder, and Isaac L. Chuang. Quantum inference on bayesian networks. *Phys. Rev. A*, 89:062315, Jun 2014.

[MBS+18] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1):4812, November 2018.

[MDTD23] Alberto Manzano, David Dechant, Jordi Tura, and Vedran Dunjko. Parametrized Quantum Circuits and their approximation capacities in the context of quantum machine learning, November 2023. arXiv:2307.14792 [quant-ph].

[MP22] Javier Mancilla and Christophe Pere. A preprocessing perspective for quantum machine learning classification advantage using NISQ algorithms. *Entropy*, 24(11):1656, November 2022. arXiv:2208.13251 [quant-ph, stat].

[MPD+23] Charles Moussa, Yash J. Patel, Vedran Dunjko, Thomas Bäck, and Jan N. Van Rijn. Hyperparameter importance and optimization of quantum neural networks across small datasets. *Machine Learning*, August 2023.

[NFT20] Ken M. Nakanishi, Keisuke Fujii, and Synge Todo. Sequential minimal optimization for quantum-classical hybrid algorithms. *Phys. Rev. Res.*, 2:043158, Oct 2020.

[OM20] Wesley O'Quinn and Shiwen Mao. Quantum machine learning: Recent advances and outlook. *IEEE Wireless Communications*, 27(3):126–131, 2020.

[PMS+14] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), July 2014.

[Pre12] John Preskill. Quantum computing and the entanglement frontier, November 2012. arXiv:1203.5813 [cond-mat, physics:quant-ph].

[Pre18] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018. arXiv:1801.00862 [cond-mat, physics:quant-ph].

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[QBW23] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. MQT Bench: Benchmarking software and design automation tools for quantum computing. *Quantum*, 2023. MQT Bench is available at `https://www.cda.cit.tum.de/mqtbench/`.

[Qis23a] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023.

[Qis23b] Qiskit contributors. Qiskit machine learning tutorial. `https://github.com/qiskit-community/qiskit-machine-learning/blob/stable/0.7/docs/tutorials/02a_training_a_quantum_model_on_a_real_dataset.ipynb`, 2023. Last accessed April 03, 2024.

[RML14] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Phys. Rev. Lett.*, 113:130503, Sep 2014.

[SAGB22] N. Schetakis, D. Aghamalyan, P. Griffin, and M. Boguslavsky. Review of some existing QML frameworks and novel hybrid classical–quantum neural networks realising binary classification for the noisy datasets. *Scientific Reports*, 12(1):11927, July 2022.

[SFLP] Salvatore J Stolfo, Wei Fan, Wenke Lee, and Andreas Prodromidis. Cost-based Modeling and Evaluation for Data Mining With Application to Fraud and Intrusion Detection: Results from the JAM Project. `https://kdd.ics.uci.edu/databases/kddcup99/task.html`. Last accessed April 05, 2024.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997.

[SK19] Maria Schuld and Nathan Killoran. Quantum machine learning in feature Hilbert spaces. *Physical Review Letters*, 122(4):040504, February 2019. arXiv:1803.07128 [quant-ph].

[SKK+23] Asel Sagingalieva, Mo Kordzanganeh, Andrii Kurkin, Artem Melnikov, Daniil Kuhmistrov, Michael Perelshtein, Alexey Melnikov, Andrea Skolik, and David Von Dollen. Hybrid quantum resnet for car classification and its hyperparameter optimization. *Quantum Machine Intelligence*, 5(2), September 2023.

[SMM23] Harshdeep Singh, Sabyashachi Mishra, and Sonjoy Majumder. Benchmarking of Different Optimizers in the Variational Quantum Algorithms for Applications in Quantum Chemistry. *The Journal of Chemical Physics*, 159(4):044117, July 2023. arXiv:2208.10285 [quant-ph].

[Spa92] J.C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.

[SS18] Pramila P. Shinde and Seema Shah. A review of machine learning and deep learning applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6, 2018.

[UK21] Antony Unwin and Kim Kleinman. The Iris Data Set: In Search of the Source of *Virginica*. *Significance*, 18(6):26–29, December 2021.

[WBL12] Nathan Wiebe, Daniel Braun, and Seth Lloyd. Quantum algorithm for data fitting. *Phys. Rev. Lett.*, 109:050505, Aug 2012.

[WBLS22] Manuela Weigold, Johanna Barzen, Frank Leymann, and Marie Salm. Data encoding patterns for quantum computing. In *Proceedings of the 27th Conference on Pattern Languages of Programs*, PLoP '20, USA, 2022. The Hillside Group.

[WG15] Nathan Wiebe and Christopher Granade. Can small quantum systems learn?, 2015.

[WKS14] Nathan Wiebe, Ashish Kapoor, and Krysta M Svore. Quantum deep learning. *arXiv preprint arXiv:1412.3489*, 2014.

[WSW+19] Max Wilson, Sam Stromswold, Filip Wudarski, Stuart Hadfield, Norm M. Tubman, and Eleanor Rieffel. Optimizing quantum heuristics with meta-learning, 2019.

[YS20] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, November 2020.

[YZ20] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications, 2020.