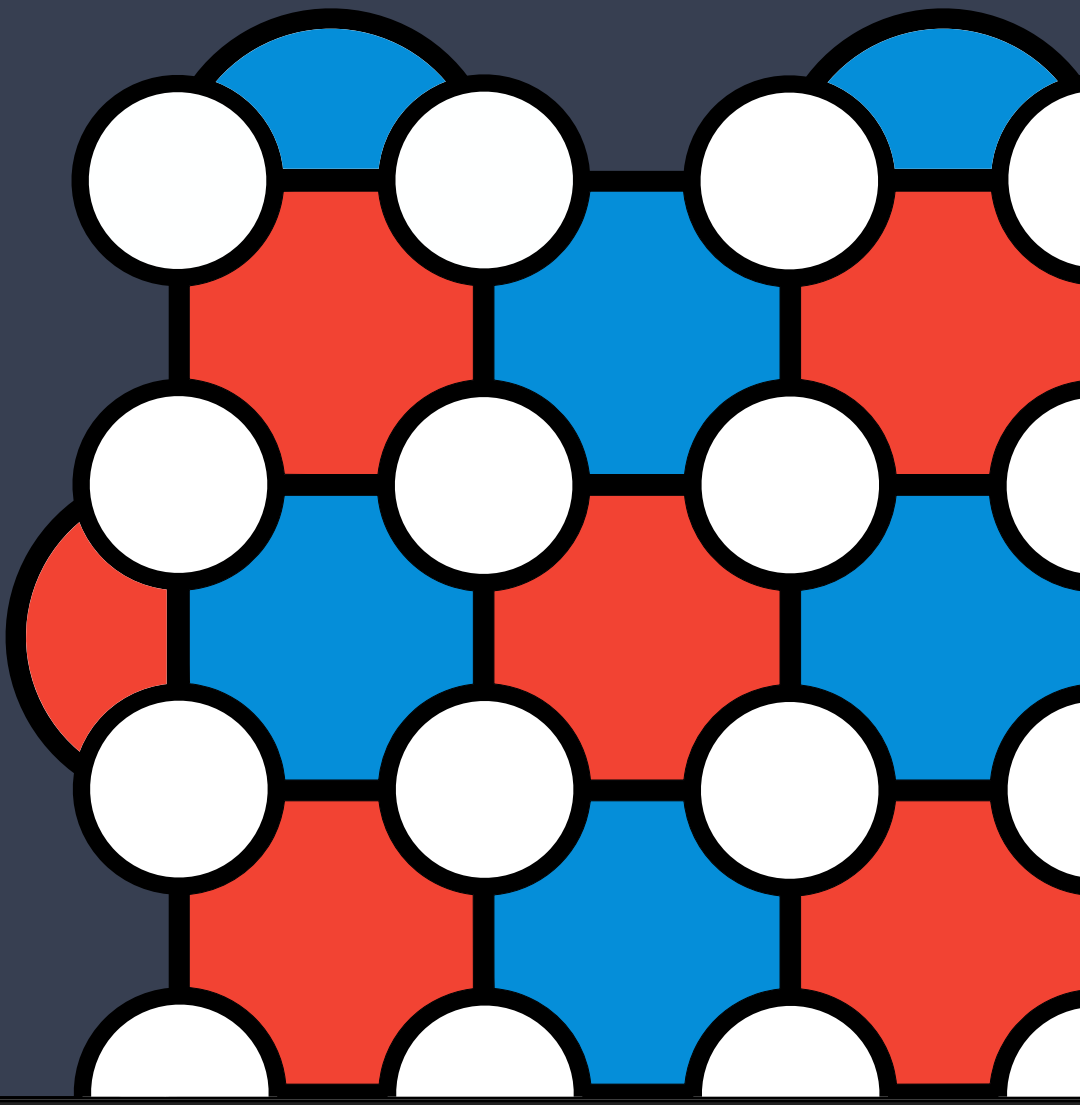
Machine Learning assisted

Quantum Error Correction

and

Quantum Feedback Control



Kai Meinerz

Dissertation

Machine Learning assisted Quantum Error Correction and Quantum Feedback Control

Inaugural-Dissertation

zur

Erlangung des Doktorgrades

der Mathematisch-Naturwissenschaftlichen Fakultät

der Universität zu Köln

vorgelegt von

Kai Meinerz

aus Wuppertal

Köln 2023

Berichterstatter:	Prof. Dr. Simon Trebst
	Prof. Dr. David Gross
Tag der mündlichen Prüfung:	15.03.2024

Abstract

Quantum feedback control is a field of research that deals with the manipulation of quantum systems towards a goal, based on continuous observation of the systems. These control strategies have been identified as an essential part of the development of future quantum technologies. An example of this is the formulation of quantum error correction strategies for fault-tolerant quantum computing. The design of control strategies is a challenging task, however, as incomplete descriptions of the quantum system at hand and external noise factors often introduce uncertainties into the system, leading to performance degradation. In this thesis, we focus on the use of machine learning based approaches to find quantum control feedback strategies. These approaches have demonstrated the ability to find near-optimal and robust strategies.

In a first study, we develop a control strategy for quantum error correction on topological surface codes in the form of a hierarchical decoder. Using a combination of machine learning and combinational decoding, we were able to demonstrate scalable decoding. We achieved nearly linear-time scaling, while still maintaining a high precision decoding comparable to state-of-the-art conventional decoding strategies. The robustness of the decoding strategies is demonstrated through performing tests on different error models including depolarizing noise and faulty syndrome measurements, as well as by changing the underlying error correction code to the rotated surface code. Analyzing the correction applied by the hierarchical decoder provides us with insight into the learned strategies and identifies possible strengths and weaknesses of the decoder, allowing for possible further developments of machine learning and algorithmic decoders based on these findings.

In a second study, we investigate the use of reinforcement learning for quantum feedback control. Since reinforcement learning encompasses model-free approaches, it is a prime candidate for finding robust strategies that are not hindered by unknown uncertainties in the experimental system. A well-known problem of reinforcement learning is the sometimes unstable training, caused by the encountered "exploration versus exploitation" dilemma during the process. Therefore, we have implemented a toy model, the quantum cartpole, designed to serve as a benchmark environment for the development of reinforcement learning based quantum feedback strategies. Based on weak measurements, the implemented control strategies have to deal with partial observability and measurement induced feedback. We provide benchmarks for three different variations of the system, including linear and nonlinear systems, using the classical control theory algorithm, linear quadratic Gaussian control, and a reinforcement learning based control strategy. By examining these results, we show the importance of state estimation techniques as part of the control process, and demonstrate the ability of reinforcement to find novel strategies that outperform conventional control strategies in highly nonlinear systems.

Contents

Outline

This thesis presents the numerical results of quantum feedback control strategies applied in the field of quantum computing using machine learning as principal underlying method. Quantum feedback control describes the manipulation of quantum states towards a certain goal. The feedback is determined based on continuously performed measurements on the system.

In the introductory **Chapter 1**, we start with a general discussion of quantum control and its importance for the development of quantum technologies. Here we set a focus on its influence in the field of quantum computing, by highlighting concrete applications. In addition, we shortly discuss the advantages of using machine learning to develop control strategies. **Chapter 2** discusses machine learning algorithms, giving a short overview of their historic development and introducing all techniques which are used in this thesis to develop control strategies. This includes basic constructs like artificial neural networks, found in all deep learning algorithms, and two different kinds of machine learning algorithms. The first being supervised learning and the second reinforcement learning.

In **Chapter 3**, we take a look at error correction for quantum computing. We start the discussion, by introducing the task of error correction in classical computing and how redundancy is used to increase the fidelity of a bit. Going to quantum computing, we discuss the unique challenges posed by the no-cloning theorem and the fact that only indirect measurements can be used. As framework to construct quantum error correcting code we discuss the stabilizer formalism. This formalism is used to construct the toric code and rotated surface code. **Chapter 4** follows up by discussing decoders for the surface codes as well as the numerical methods used to quantify the behavior of the decoders. The decoders include two well-known algorithmic decoders used as comparison and a machine learning based hierarchical decoder. The decoders are then tested for speed and precision using the toric code and rotated surface code for different underlying noise models. In addition, we perform an investigation to analyze the decoding strategies learned by the hierarchical decoder.

In **Chapter 5**, we discuss quantum feedback control, starting with the classical version of the problem. We then discuss how feedback control changes in the quantum case, discussing notion of measurements and how they interact with a quantum system. Then we discuss the application of a classical control algorithm, which we transfer to the quantum case. **Chapter 6** takes the previous discussed definition of the quantum feedback problem and converts it into a benchmark environment for reinforcement learning. We provide benchmark values for the classical and reinforcement learning strategies, comparing them on different variations of the system, including linear and non-linear variations. Finally, we investigate the learned strategies of the reinforcement control strategies.

In **Chapter 6**, we summarize our results and discuss future lines of research.

Chapter 1

Introduction

Quantum control is a research area focused on manipulating and controlling quantum systems towards a set goal. The unique properties of a quantum system, such as entanglement and coherence, pose challenges to the field, which do not occur in classical mechanical systems. As a result, classical control theory is inadequate, which necessitates the development of a specific quantum control theory.

The earliest application of quantum control can be traced back to the laser development in the 1960s [1], where initial trials to create and break particular molecular bonds were conducted. Even though these first experiments failed at the time, a successful realization was managed in the 1980s, where, using femtosecond laser pulses, a chemical reaction could be controlled, determining the final product [2]. Ever since the field of quantum control theory has increased rapidly [3], making noticeable contributions to physical chemistry and quantum optics as well as contributing to our understanding of fundamental aspects of quantum mechanics [4].

This trend of growing interest is unlikely to stop, as in recent years the development of the general principles of quantum control theory has been recognized as an essential requirement for the future application of quantum technologies [5]. The development of these technologies are driven by two main factors. The first is the constant miniaturization of hardware until length scales are reached where quantum effects become visible. The second is the creation of quantum states with taylor-made designed properties, enabling new developments in the field of computing, communications or sensing. Being applicable to a wide range of different fields, it is not surprising that it is at the center of public attention, with government-funded programmes around the world investing billions of euros, to advance the development of quantum technology, such as the European Quantum Flagship initiative [6]. The same is true in the private sector, where global companies such as Google, IBM and Microsoft are investing in quantum technologies.

In this thesis we want to focus on the application of quantum control to the field of quantum computing. Therefore, in this introductory chapter, we want to give the reader a brief introduction to quantum control theory, and contextualize it in the field of quantum computing by highlighting areas of application and the goal towards which it is working. This chapter has been written on the basis of Refs. [3, 7, 8], which are recommended for further reading.

1.1 Quantum control models

Quantum control is a broad field which encompasses the engineering and manipulating of quantum system to desired outcome. The algorithms that determine the applied control are called *control strategies* and generally, but not exclusively, are described by applying modification to the Hamiltonian

$$H(t) = H_0 + H_c(t), \tag{1.1}$$

where H_0 describes the dynamics of the bare system and the H_c describes the added controls. Taking an arbitrary closed quantum system $|\psi(t)\rangle$, the system evolves according to the Schrödinger equation [9]

$$i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle = H_0|\psi(t)\rangle, \qquad |\psi(t=0)\rangle = |\psi_0\rangle, \tag{1.2}$$

where H_0 is the Hamiltonian of the system and the initial state of the system obeys $|\psi_0|^2 = \langle\psi_0|\psi_0\rangle = 1$.

In the case of a closed quantum system, the control of the system is often described by the *bilinear model*. There H_c can be realized using a set of control functions $u_k(t) \in \mathbb{R}$, which are coupled to the system via time-independent Hermitian interaction Hamiltonians H_k $(k = 1, 2, ...)$. This yields the total Hamiltonian

$$H(t) = H_0 + \sum_k u_k(t)H_k \tag{1.3}$$

and the time evolution is determined by

$$i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle = \left[H_0 + \sum_k u_k(t)H_k\right]|\psi(t)\rangle. \tag{1.4}$$

Here the origin of the name becomes apparent, as the total Hamiltonian is independently linear in both H_0 and H_c, and therefore bilinear. This Hamiltonian generally allows one to tackle the typical control problem of finding a set of control $u_k(t)$ and a final time $T > 0$, which takes the initial state $|\psi_0\rangle$ to a desired target state $|\psi_f\rangle$. The total Hamiltonian determines the unitary time evolution operator $U(t)$, which describes the transition from the initial state $|\psi_0\rangle$ to $|\psi(t)\rangle$ as shown in

$$|\psi(t)\rangle = U(t)|\psi_0\rangle. \tag{1.5}$$

Substituting Eq. (1.5) into Eq. (1.4) shows that $U(t)$ has to satisfy the differential equation

$$i\hbar\dot{U} = \left[H_0 + \sum_k u_k(t)H_k\right]U(t), \qquad U(0) = I. \tag{1.6}$$

This allows the description of closed systems such as spin systems in nuclear magnetic resonance, allowing the formulation of control strategies to improve the sensitivity of the system in the presence of relaxation [10–12].

In the case that one wants to control a quantum system over an indefinite amount of time, it is necessary to chose controls depending on the current state of the system. This requires knowledge about the state and therefore requires observation of the system. These systems are often described using their density matrix

$$\rho = \sum_j p_j|\psi_j\rangle\langle\psi_j|, \tag{1.7}$$

where $\langle\psi_j| = (|\psi_j\rangle)^\dagger$ and $\sum_j p_j = 1$. The evolution of a system, which is continuously measured in the observable X, can be described with stochastic master equations (**SME**) [13]

$$d\rho = -\frac{i}{\hbar}[H, \rho]\,dt + \mathcal{H}[X]\,\rho dW + \mathcal{D}[X]\,\rho dt \tag{1.8}$$

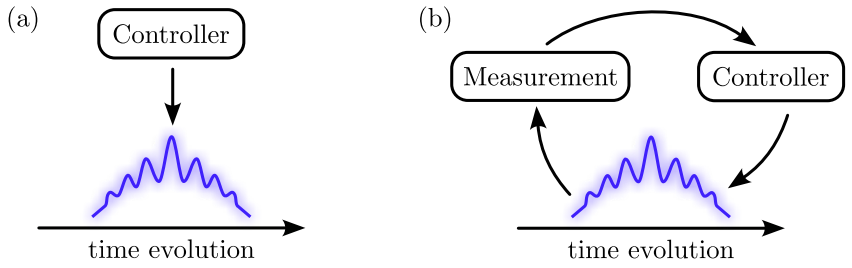


Figure 1.1 – **Controller design** in quantum control theory. (a) The open-loop controller directly acts on the system, based on a predetermined scheme. (b) The closed-loop controller decides its action depending on measurement results from the system. Figures inspired by [16].

where the Hamiltonian is given as $H = H_0 + H_c$, with H_0 describing the dynamic and H_c the applied control. The second term in the equation is the measurement superoperator,

$$\mathcal{H}[X]\rho = X\rho + \rho X^\dagger - Tr\left[\left(X + X^\dagger\right)\rho\right]\rho \tag{1.9}$$

describing the resulting change in the state of the syste, when performing a measurement. Here the term *superoperator* refers to the fact that $\mathcal{H}[X]$ operates on ρ from both sides. The variable dW associated with the measurement refers to a *Wiener increment*, which returns a random value based on zero-mean Gaussian and width equal to dt. Performing the measurement then gives the measurement result

$$dy = Tr\left[\left(X + X^\dagger\right)\rho\right]dt + \kappa dW, \tag{1.10}$$

where κ is some factor related to the measurement strength. Based on the measurement results one chooses H_c accordingly. The third term of the SME is a *Lindblad superoperator* [14]

$$\mathcal{D}[X] = X\rho X^\dagger - \frac{1}{2}\left(X^\dagger X\rho + \rho X^\dagger X\right) \tag{1.11}$$

describing the effect of noise due to the environment.

It has to be pointed out that the SME in 1.8 is only a typical representation of SME's and can change depending on different measurement processes [15].

1.2 Control strategies

Even if one has a quantum system that can be described by one of the mentioned control models, it is still necessary to find a suitable control strategy to fulfil a given goal e.g. finding u_k in the bilinear model. These strategies can be separated into two different classes: open- and closed-loop control.

Open-loop control In the case of open-loop control, the control strategy has been designed before the experiment and will be executed without modification as demonstrated in Fig. 1.1 (a). These strategies are based on the knowledge previously attained, and are often described using the bilinear model [3]. As the controls are set in the beginning, it is necessary to either know H_0 exactly or approximately, in order to formulate the control strategies. Therefore open-loop control strategies have shown success in case of simple closed quantum system, but can run into trouble for more complex systems, as unknown parts of the Hamiltonian or noise sources causes degradation of the performance of the control strategy [17, 18].

Closed-loop control In contrast to the open-loop system, the closed-loop systems utilize measurement results from the system in order to determine the best control strategy as shown in Fig. 1.1. The best example for these strategies is quantum feedback control, where systems are continuously measured and subjected to control actions, whose form depends on these measurement results.

To help us discover these control strategies, different methods have been developed that focus on different aspects of controlling a system. One of these methods, that is often used in open-loop systems, is optimal control. The idea of this is to choose a set of controllers u_k that can achieve a goal while minimizing a cost function. This cost function determines the realized control strategy and can be chosen according to the problem at hand. Typical cost functions are the minimization of the time to reach the target [19], the control energy [20] or the error between the final state and the target state. Even the combination of these costs is possible. Since the cost function can be chosen freely, it follows that there is generally no globally optimal strategy. Even if the underlying problem is the same, the optimal strategy for minimizing the required time or for minimizing the control energy are two distinct strategies.

As the name suggests, it may seem that any designed control strategy should be an optimal strategy. But in an actual implementation or in a complex system, trying to achieve an optimal strategy can be either nearly impossible or even counterproductive. As for quantum systems, it is inevitable to encounter uncertainties and perturbations of the system [21], which increase the difficulty of designing a control strategy. The same is true when only incomplete information about the system is available, causing a difference between the assumed and the actual model. In particular, open-loop strategies suffer from these additional difficulties, since the strategies are set in advance and cannot react to unexpected influences. Therefore, the robustness of control strategies, defined as how sensitive the reached state is to deviations in the parameter space of the Hamiltonian, has been identified as an important factor for practical application in quantum systems [22]. Naturally closed-loop strategies have been recognized as candidates for implementing robust control, as they can use information about the state of the system to negate unwanted influences caused by external noise.

Here we want to take a closer look at closed-loop strategy quantum feedback control, as it is designed to constantly observe a system and apply feedback accordingly. This strategy has already been successfully used preventing entanglement decay in optical quantum computing systems as well as making it robust against signal lost [23] or in state purification of superconducting quantum systems [24]. The difficulty in quantum feedback control is the acquisition of information, as every information gain also introduces a disturbance to the system, following the Busch's theorem [25]. Since continuous measurements of a quantum system are inherently noisy [26], the formulation of a control strategy often involves a filtering step, which estimates the actual observable from the measurement results.

Thanks to the rapidly increasing accessibility of computational resources, machine learning techniques have proven to be a promising candidate in realizing quantum feedback control, allowing the automatic learn of control strategies based on available data. This leads to robust control strategies, as demonstrated for the example of learning error mitigation strategies [27].

In particular, *reinforcement learning* (RL) techniques are prime candidates for quantum feedback control, as their innate algorithms are already based on receiving feedback depending on performed action. As RL is a model-free learning technique, it requires no prior knowledge of the system, and is able to learn the underlying dynamics and noise sources, thereby reducing the error introduced by approximating the system. RL based control has been shown to be able to achieve performance superior to conventional methods [28, 29]. In the context of quantum computing, precise control of qubits [28, 30] and quantum gates [31] has been successfully

demonstrated. It has also been used to construct error-correcting codes [32] and to find decoding strategies [33]

ML not only provides robust control strategies at the end of development, but has also shown advantages in the search for control strategies. It has demonstrated to be a flexible approach, as learned control strategies can be easily transferred between similar task without the need to modify the underlying approach [34]. Often it is enough top perform a short training on the new task to receive a top performing control strategy. Additionally ML has also shown its ability to approximate already existing control strategy, achieving similar performance, while using less computational resources [35].

As this thesis sets a focus on implementing ML based control strategies, a detailed introduction to machine learning is given in Ch. 2.

1.3 Promises of fault-tolerant computing

The development of quantum technologies offers exciting opportunities both in scientific research and in various applied fields. Some of these potential applications could be transformative for the respective field. Here we want to take a closer look at the example of quantum computing.

Under the premise of having achieved fault-tolerant quantum computing, it promises to implement algorithms which provide polynomial and superpolynomial speed up compared to the most efficient known classical algorithms, allowing the solution of problems that even the best supercomputers cannot solve. The first example of this computational advantage was shown in the form of the Deutsch algorithm [36]. Assuming a function acting on an n-bit string returning either 0 or 1 for each value in the string, promising [37] that the resulting output is either constant (all values 0 or 1) or balanced (one half has the value 0, the other has value 1). The task of the Deutsch algorithm is to determine whether a function f is constant or balanced. For a classical computer it takes at worst $\mathcal{O}(n)$ steps to solve this problem, whereas quantum computers would be able to solve it within a single step. This was the first example to demonstrate a *deterministic* quantum advantage.

Probably the most impactful example of a speed-up achieved by a quantum algorithm was presented by Shor in 1994 [38, 39]. He introduced a quantum algorithm, for the prime factorization of large numbers in polynomial time, demonstrating an almost exponential speedup[1] over the best known classical algorithm.This discovery had significant implications as the factorization of large numbers is an intentionally hard problem at the core of public key cryptography and is implemented in the RSA algorithm [42]. Decrypting an RSA2048 key, which would take a classical computer billions of years, could be solved by a quantum computer in a matter of hours. This discovery was an early example of the usefulness of a quantum computer and a strong motivator for research in this field.

Besides being a security risk, quantum computing promises exciting possibilities for scientific research as well as industrial application. In the perspective of research the simulation of quantum mechanics has long been acknowledged as a challenging problem, as the needed computational resources scale exponentially due to the exponential growth of the Hilbert space. While this *exponential explosion* can be counter acted by approximation methods like Monte Carlo, these are not always available, depending on the exact nature of the underlying problem,

[1] Shor algorithm is able to factor an integer N in $O\left((\log N)^2 (\log\log N)\right)$ [40] in its fastest implementation, while the fastest classical algorithm takes $O\left(\exp\left[1.9(\log N)^{1/3}(\log\log N)^{2/3}\right]\right)$ [41].

and also face their own limitations. The solution for this problem was proclaimed by Feynman in 1981 in his seminal lecture with his famous words "Nature isn't classical dammit, and if you want to make a simulation of nature you better make it quantum mechanical, and by golly it's a wonderful problem because it doesn't look so easy" [43]. As the Hilbert space of the quantum computer grows exponentially with the number of physical resources, it is a natural candidate to simulate these systems.

For the industry, quantum computing might revolutionize the manufacturing of fertilizers by studying the nitrogen-fixing FeMo cofactor [44], which is an enzyme converting nitrogen N_2 into ammonia (NH_3). The enzyme is able to accomplish the transformation of the molecule at ambient conditions, whereas the current production method needs high temperatures and pressures [45]. In 2019, this process has consumed 1% of the global energy production [46], so unraveling the chemical process behind the FeMo cofactor could lead to a large scale application and significantly reduce the required energy. Or it could help the field of drug development, which is a notoriously known to be an expensive line of research as the development of new drug takes between 12-15 years and approximately 80% fail in development [47]. By simulating relevant molecules, desired and undesired effects can be predicted, thereby speeding up the development process [48].

While all these promises of quantum computing sound fantastic, unfortunately it requires fault-tolerant quantum computing, which has yet to be realized and will take *some* years of development. The difficulty comes from the fact that noise sources introduce errors into the system. Because quantum systems are inherently fragile, it is necessary to isolate them from the environment as much as possible, but at the same time it is necessary for the quantum system to interact strongly with each other in order to process information. This is why quantum computers are built inside dilution refrigerators [49], which cool the computing chips down to near zero Kelvin, minimizing external noise. But even the remaining noise can be more than enough to threaten the integrity of the qubits, so quantum control techniques are used to combat the noise and make quantum computing possible.

1.4 Quantum control application in quantum computing

A variety of control strategies have been developed to combat noise and errors occurring in quantum computing. The control strategies can be divided into three different groups called error suppression, mitigation and correction. Here we want give a short description of the different branches, and put them into a contextual frame of quantum control theory by providing examples of state-of-the-art applied control strategies.

Error suppression Error suppression is the most basic level of handling error, which aims to reduce the probability of errors occurring in the first place. These techniques are commonly open-loop designs, where the knowledge of undesired effect allows the implementation of modification to avoid or negate their negative effects. Mostly error suppression is considered at a hardware level and consists of adding or altering control signals.

A class of of this application are *dynamic decoupling* (**DD**) techniques [50], such as the spin echo, which utilizes pulses to qubits in order reset their value to their initial state. The basic theory of DD is as follows. Considering a time independent Hamiltonian with is respective time evolution over a duration τ

$$H = H_S + H_B + H_{SB} \qquad f_\tau = U(\tau) = \exp(-i\tau H), \tag{1.12}$$

where H_S and H_B describe the system and bath respectively and H_{SB} describes the interaction between the two. The system Hamiltonian can be further divided into $H_S = H_0 + H_1$ where H_1 describes an undesired behavior of the system. Together with the system bath interaction the error Hamiltonian is written $H_{\text{err}} = H_1 + H_{SB}$ representing the term DD wants to remove.

In order to do that, a time-dependent control Hamiltonian is added to the system to compensate for H_{err}:

$$H(t) = H_{S_0} + H_B + H_{\text{err}} + H_c(t). \tag{1.13}$$

Assuming an ideal pulse sequence the control Hamiltonian is given by a sum of instantaneous error free Hamiltonians $\{\Omega_0 H_{P_k}\}_{k=1}^n$ generating pulses at corresponding time intervals $\{\tau_k\}_{k=1}^n$ with Ω_0 being the peak value of the pulse. According to the bilinear model from Eq. (1.3), the control Hamiltionan and corresponding unitary evolution are given as

$$H_c(t) = \Omega_0 \sum_{k=1}^{n} \delta(t - t_k) H_{P_k}, \qquad P_k = \exp\left(-i\frac{\pi}{2} H_{P_k}\right), \tag{1.14}$$

where $t_k = \sum_{j=1}^{k} \tau_j$. The total time evolution of the system is given as

$$U(T) = f_{\tau_n} P_n \cdots f_{\tau_2} P_2 f_{\tau_1} P_1 \tag{1.15}$$

with $T = \sum_n \tau_n$ is the total time sequence and the pulses are assumed top be instantaneous. The goal of control theory to determine the optimal sequence of pulses to decouple the system from the bath. This is an example of open-loop control as the pulse sequence is determined beforehand and does not take in any feedback. In the optimal control case the pulse sequence will completely decouple the system from the bath environment. In realistic settings only approximate decoupling is reached, so reducing the error further is a highly active field of research, while taking increasingly more realistic physical scenarios into consideration.

Error mitigation In contrast to the error suppression, error mitigation is a post-processing closed-loop control method, trying to eliminate the noise in estimating expectation values, based on measurement results.

Considering a circuit, the goal is to determine the expectation value of some observable X. In a noiseless circuit, this can be done by running the circuit a number of times, producing the perfect output state ρ_0, but since a noiseless circuit is still a distant dream, a noisy state is reached ρ_{noise} allowing only an approximation of the expectation value X_{est} based on a noisy measurement of the expectation value X_{noise}. The ultimate goal of error mitigation is therefore to reduce the difference between the estimated and actual expectation value, typically given by the *mean square error* (MSE)

$$\text{MSE}[X_{\text{est}}] = E\left[\left(X_{\text{est}} - Tr[X\rho_0]\right)^2\right], \tag{1.16}$$

as $Tr[X\rho_0]$ is generally unknown, the MSE is usually estimated via numerical methods or physical experiments by applying them to some specific use case. To minimize the MSE different techniques have been developed, which generally make use of additional runs of the circuit at the same or higher noise level. Therefore error mitigation is a closed-loop control method, as measurements results are needed to formulate a control strategy. An example of this is *zero noise exploration*, which uses the measurement results at different noise levels $\{(\lambda_m, \text{Tr}[O\rho_{\lambda_m}])\}$ to estimate the expectation value at zero noise using a function $f(\lambda, \theta)$, that depends on the

noise level λ and is determined by its parameters θ. The error-mitigated estimation value is therefore given by

$$E\left[O_{est}\right] = f\left(0, \theta^*\right), \tag{1.17}$$

where θ^* indicates the optimal parameters of the function. Research in this field is working on finding suitable control strategies, not only depending on the quality of estimation, but also the numbers of data point needed.

Error correction *Quantum error correction* (**QEC**) is the long term goal to realize fault-tolerant quantum computing. It aims at building a *quantum error correcting code* (**QECC**) by introducing redundancies in the system, enabling the correction of errors that occur in order to preserve the logical information. This is based on the quantum threshold theorem [51], which states that when operating below a certain physical error rate, which determines the probability that errors occur, scaling up the quantum code suppresses the logical error rate. The current quantum computers are still to noisy to implement active error correction, as single qubit error rates are still around 10^{-3} [52, 53], while the threshold theorem requires an error rate around 10^{-4} [54].

Quantum error correcting is an example of quantum feedback control as continuous measurements are used to apply correction onto the system according to a chosen control law. Therefore it naturally belongs to closed-loop control just like error mitigation. Here we give a short explanation of error correction, a more detailed description can be found in the dedicated chapter Ch. 3.

Continuous quantum error correction can naturally be expressed using stochastic master equation 1.8. A straightforward example of this can be demonstrated using a three qubit system $\alpha|000\rangle + \beta|111\rangle$, follwing [55], where each qubit may be affected by an error. In this case, a bit flip error would be represented by an X operator. The qubits evolve according to the noise term

$$d\rho_{\text{noise}} = \gamma\left(\mathcal{D}\left[XII\right] + \mathcal{D}\left[IXI\right] + \mathcal{D}\left[IIX\right]\right)\rho\ dt, \tag{1.18}$$

where γ is the probability of a bit flip error occurring in the time interval $[t, t+dt]$. The goal is to correct an error as soon as it occurs, meaning that after identifying an bit flip error, it would be corrected applying by a X operator on the respective qubit. Therefore the feedback Hamiltonian is given by

$$H_c = \lambda_1 XII + \lambda_2 IXI + \lambda_3 IIX, \tag{1.19}$$

where λ_k are the control parameters determining which correction are applied. In order to choose the appropriate λ_k it is necessary to locate the error on the bit flip code. For this neighboring qubits are projectively measured, using the observables

$$M_0 = ZZI \tag{1.20}$$
$$M_1 = IZZ. \tag{1.21}$$

This tells us whether an error occurred on one of the measured qubit, but not on which one. Expressing the measurement using the SME formalism results in

$$d\rho_{\text{meas}} = \sqrt{\kappa}\left(\mathcal{H}[ZZI]dW_1 + \mathcal{H}[IZZ]dW_2\right)\rho \tag{1.22}$$
$$dy_1 = 2\kappa\langle ZZI\rangle dt + \sqrt{\kappa}dW_1 \tag{1.23}$$
$$dy_2 = 2\kappa\langle IZZ\rangle dt + \sqrt{\kappa}dW_2, \tag{1.24}$$

where dy_k are the measurement results, κ is the measurement strength and dW_k are Wiener increments. The task of the control strategy is finding suitable λ_k based on the received dy_k.

After each correction, new errors may occur, requiring a continuous process of measurement correction. In the optimal case, this will allow us to secure quantum information over arbitrary time. The control strategy that determines the correction must satisfy several conditions. The most obvious is the precision of the proposed correction, as it directly affects the fidelity of the logical qubit. The second is the operation time needed to determine the correction, since new errors can occur during this time. Finally, the robustness of the strategy is also important, as noise sources may not be uniformly distributed over the qubits and may vary over time [56].

1.5 Noisy intermediate scale

As the development of quantum computing is pushed further to realize the desired fault-tolerant computing some day, the current state of quantum computing is described as the *noisy intermediate scale quantum* (**NISQ**) era [8]. The intermediate scale here refers to the number of experimentally realized qubits which is loosly set as up to a few hundred [8] and the noisy part refers to the quality of said qubits, which are still prone to noise and errors. While this number is not large enough to realize the promising quantum algorithms like Shors algorithm, it has been enough to demonstrate examples of quantum advantage.

During the NISQ era, numerous achievements in quantum computing have been achieved. The most prominent example is likely Google's experiment, where they claimed to have demonstrated the first quantum advantage, using their 53 qubit Sycamore processor. Quantum advantage refers to the solving of a problem, which cannot be solved by state-of-the-art supercomputers in any feasible amount of time, irrespective of the usefulness of the problem. This was demonstrated as the Sycamore computer completed a calculation in about 200s which would have taken 10,000 years on the then best supercomputers [57]. This result was heavily discussed in the quantum computing community [58], as it was later demonstrated, that using newly developed classical algorithms the Sycamore task could also be solved in 15 hours using an array of 512 GPUs [59], showing that the concept of quantum advantage heavily depends on the availability of classical methods. Nevertheless this was a crucial proof of concept, demonstrating that the coherent control of the states of a 2^{53} computational space over an extended period of time is possible.

Recently IBM has claimed to have reached the next step in quantum advantage, dubbed as *quantum utility* [60], by demonstrating the ability to solve problems at a scale beyond brute force classical simulation for a *practical* application [61]. Using a 127-qubits quantum processor with 60 layers of 2 qubits gates. As benchmark they have used the Trotterized time evolution of a 2d transverse Ising field. One of the key factors in reaching this milestone was the integration of error mitigation strategies in form of zero noise exploration [62] as well as probabilistic error cancellation [63]. Most recently they followed these successes by unveiling their newest quantum processor Condor, housing 1,121 superconducting qubits, pushing the limit of the intermediate scale definition.

At the moment, unfortunately, QEC has not been realized for computational tasks[2], not only because current error rate are still to high, but also the overhead introduced by QEC itself is too big. Even in the currently leading implementation, the surface code [65], the required number

[2] An experiment, implementing QEC on a $d = 7$ surface code, claims that their logical encoding has surpassed the fidelity of the physical qubits [64]. At the time of writing this dissertation, the results are still being discussed and have yet to pass the peer review phase.

of qubits scales with $O\left(d^2\right)$, where d is the code distance and relates to the number of error that can be corrected. In order to reach fault-tolerance, the code distance must be chosen high enough to compensate the noise level of the system. With the current error rates, this would result in an unrealistic number of needed qubits. In IBM's roadmap of quantum computer development, the first demonstration of real-time error correction is scheduled for 2026 [66], while in 2029 error correction should be implemented as a standard. Still, first implementations $d = 3$ and $d = 5$ surface codes have delivered a proof of concept by demonstrating an increased logical fidelity with increasing code distance [52] and successful preservation of logical states over multiple error correcting cycles in a $d = 3$ surface code [67]. Thereby giving hope for fault-tolerant implementation of quantum computing.

Chapter 2

Machine learning

The origin of machine learning in its modern sense started in 1957 as the psychologist Frank Rosenblatt created a machine implementing the *perceptron* [68] for recognizing the letters of the alphabet. This was the prototype of the artificial neural networks, which are a key element of today utilized machine learning techniques. Unfortunately, the development of this field slowed down as it was shown in 1969 that perceptron are limited in the complexity of solvable problems and are unable to represent logical functions like XOR or NXOR [69]. Until the 1980s the machine learning field entered a time period called the first *AI winter* [70]. It was only with the proposal of multilayered neural network structures [71] and successful training via backpropagation [72] that hope for future success in this field rekindled. However, real success remained less then expected causing the interest and investment once again to die down in the early 1990s. This marks the start of the second AI winter [70].

The turning point for machine learning came in early 2010 as the invention of *deep machine learning* algorithms promised exciting new applications, while at the same time the necessary hardware became available as the cost of parallel computing and memory dropped, and the realization was drawn that GPUs could be used for machine learning purposes[70]. This was all fueled by the interest of the growing big data industry. This all led to a rapid development in the machine learning field which still has not stopped until today and has already found its way into our daily life. Be it as autonomously driving cars [73], generative AI like ChatGpt [74] or facial recognition software [75].

With the rising capabilities of ML algorithms, they naturally found their way into the scientific world, driven by the dream of achieving *automated science* [76, 77]. The first example of successful application of ML in the field of condensed matter physic was demonstrated by Carassquilla *et al.* [78], where a ANN was trained to identify the phase transitions of different Hamiltonians. Similarly, in the field of quantum computing R. Melko and G. Torlai have demonstrated that a simple stochastic neural network model can be trained to construct a general error-correcting protocol [79]. Since many different approaches to construct and train these neural decoders have emerged, one of the most common approaches is to utilize the ML capabilities of pattern recognition via convolutional networks [80] and supervised learning [81].

Another promising field of machine learning for quantum control are *reinforcement learning* **RL** based techniques, which are able to explore solution to a given task autonomously. This strength of these techniques has been demonstrated by achieving performance beyond human expertise in complex games [82–84]. Recently, these techniques have found their way into the field of quantum control, where they have demonstrated their capability to provide experimental control by being able to precisely control qubits [28] or even help at a higher level by designing error correcting codes [85]. So RL techniques could help realize fault-tolerant quantum computing.

In this chapter, we give a concise introduction to *machine learning*, explaining the funda-

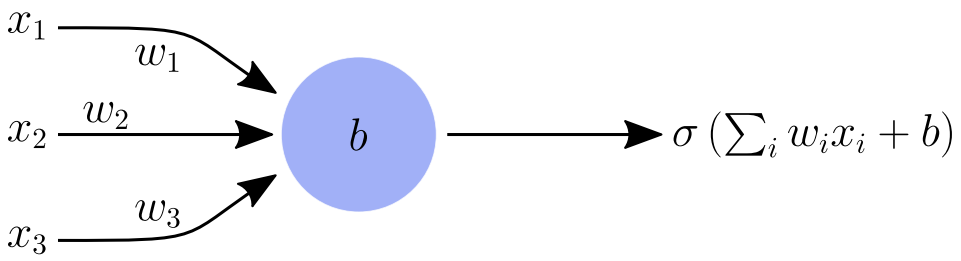


Figure 2.1 – **Visual representation of a neuron.** A singular neuron is defined by the innate bias b and the weights w_i, where each input is assigned its own weight. The output at the end is transformed using an activation function σ. If one chooses the Heaviside step function $\Theta(x)$ as activation, the neuron takes on the special form of the *perceptron*.

mental building blocks as well as the advanced techniques utilized in this thesis. Here we have made the pedagogical decision to write the chapter in a self-contained manner, so that no prior knowledge of the topic is required and all implemented ML algorithms can be understood.

2.1 Artificial neural networks

The key element that allows machine learning take a leap forward are *artificial neural networks* (**ANN**s). The first ANN was introduced in 1943 by *Warren McCulloh* and *Walter Pitts* [86], who published a general theory of information processing based on networks of binary decisions. The elements of the network are called *neurons*, based after their inspiration, the biological neurons. The first actual implementation was done by F. Rosenblatt in 1958 [68], where they coined the name *perceptrons*, as the name of their specific neuron implementation. As shown in Fig. 2.1, each perceptron can take an input vector of arbitrary size $x \in \mathbb{R}^n$ and in return gives a binary output of $f(x) \in [0, 1]$, which is a weighted linear combination of the inputs

$$f(x) = \Theta\left(\sum_i w_i x_i + b\right), \tag{2.1}$$

where w_i are the specific weights of the inputs, b is the bias of the perceptron and Θ being the *Heaviside* function. The weights and biases are then tuned so that the perceptron returns the desired output. The perceptrons were then assembled into a network, called *multi layer perceptron* **MLP**, by using the output of one layer of perceptrons as the input for the next layer. The first MLP consisted of three layers. To describe the MLP, the nomenclature was introduced to refer to the first layer as the input layer, the last layer as the output layer and all layers in between as hidden layers, as shown in Fig. 2.2.

2.1.1 Backpropagation

The first deep learning algorithm for MLP, meaning that all weights and biases can be modified, was published in 1966 [87]. This made it possible to train a MLP with any number of hidden layers by training the network layer by layer. This laid the foundation for the learning method commonly used today, called *backpropagation* [72]. The process of backpropagation can be divided into three steps. The first step is the *forward propagation* in the MLP, where an input is fed into the network and passed through all the layers until an output is received at the end.

The second step is the calculation of a loss function $L(y, \tilde{y})$, which measures the discrepancy between the output y of the MLP and the target output $\tilde{y}$, and the calculation of partial

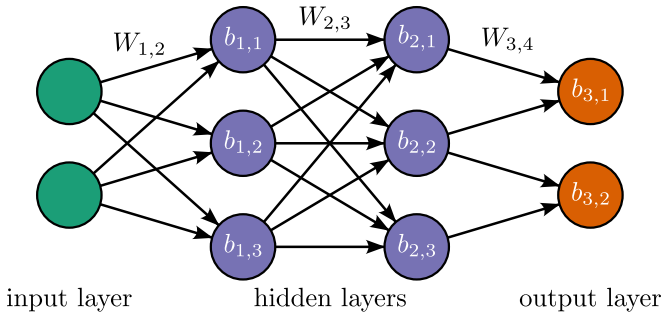


Figure 2.2 – **Artificial neural network.** The network can be divided into three separate parts. The first layer (green) is the input layer, which receives the initial data, whereas the last layer is the output layer (orange) that returns the output, after the data propagated through the whole network. All layers in between of these two are called hidden layers.

derivatives of the loss function $\frac{\partial L}{\partial w^i_{j,k}}$ and $\frac{\partial L}{\partial b^i_j}$. While the exact form of the loss function can be chosen freely, depending on the underlying task, it must fulfill certain conditions: It must be able to be written as an average $L = \frac{1}{n}\sum_x L_x$ over the training inputs x and must be a function of the output of the MLP $L(y)$ [88]. The computation of the derivatives of the loss function follows the chain rule, whereby we start here with the simpler case of the biases of the MLP:

$$\begin{aligned}\frac{\partial L}{\partial b^l_j} &= \frac{\partial L}{\partial \sigma(z^l_j)}\frac{\partial \sigma(z^l_j)}{\partial b^l_j} \\ &= \frac{\partial L}{\partial \sigma(z^l_j)}\frac{\partial \sigma(z^l_j)}{\partial z^l_j}\frac{\partial z^l_j}{\partial b^l_j} \\ &= \delta^l_{i,j},\end{aligned} \tag{2.2}$$

where we introduce the intermediate quantity $z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$ and can therefore use $\frac{\partial z^l_j}{\partial b^l_j} = 1$. We also have introduced

$$\begin{aligned}\delta^l_j &= \frac{\partial L}{\partial \sigma(z^l_j)} \\ &= \frac{\partial L}{\partial \sigma(z^l_j)}\frac{\partial \sigma(z^l_j)}{\partial z^l_j}\end{aligned} \tag{2.3}$$

as a *neural error* associated with the j-th neuron l-th layer. When the error approaches zero, it indicates optimality of the neuron [88].

We repeat this process with the weights of the network, resulting in the derivatives

$$\begin{aligned}\frac{\partial L}{\partial w^l_{i,j}} &= \frac{\partial L}{\partial \sigma(z^l_i)}\frac{\partial \sigma(z^l_i)}{\partial w^l_{i,j}} \\ &= \frac{\partial L}{\partial \sigma(z^l_i)}\frac{\partial \sigma(z^l_i)}{\partial z^l_i}\frac{\partial z^l_i}{\partial w^l_{i,j}} \\ &= \delta^l_j \sigma(z^{l-1}_k),\end{aligned} \tag{2.4}$$

where we can see a dependence on the preceding layer, which was missing in Eq. (2.2).

Now that we have expressed both partial derivatives by the neural error δ_j^l, we examine the connection with the error of the next layer δ_j^{l+1}. Using $z_k^{l+1} = \sum_j w_{kj}^{l+1}\sigma(z_j^l) + b_k^{l+1}$ we obtain the relation

$$\begin{aligned}\delta_j^l &= \frac{\partial L}{\partial z_i^l} \\ &= \sum_k \frac{\partial L}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_l^k} \\ &= \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_l^k} \\ &= \sum_k w_{kj}^{l+1} \frac{\sigma(z_j^l)}{\partial z_j^l} \delta_k^{l+1},\end{aligned} \tag{2.5}$$

which is the central equation of backpropagation. Here we can see that the calculation of the neural errors starts at the last layer and then propogates backwards. Since the loss function depends directly on $\sigma(z_k^L)$, the neural error of the last layer δ_j^L can be calculated directly using Eq. (2.3).

But there is also a problem in this backpropagation method. Since each neural error δ_j^l depends on the neural error in the next layer δ_j^{l+1} and, assuming small errors here, the learning progress of the layer l is slower then of the $l+1$ layer. With each backward pass, the training of neurons in the layers is impeded, ultimately causing a decline in performance when a large number of hidden layers are utilized. This issue is commonly referred to as the *vanishing gradient* problem [89].

Conversely, the *exploding gradient* presents the opposite problem where the gradient grows increasingly large with each layer, rendering the network incapable of learning.

2.1.2 Activation functions

The activation function of a neuron defines the output that a neuron can give, where it is necessary for the backaction that the function and its derivatives can be evaluated efficiently. In addition, non-linear functions should be chosen, as they allow networks to solve non-trivial problems with a small number of hidden nodes. It has been shown that a two-layer neural network with a nonlinear activation function is a universal function approximator [90].

For the network of the neural decoder we will utilize two different activation functions, these being the *sigmoid* and *rectified linear unit* **ReLU** [91] activation functions.

The sigmoid function [92] is defined by

$$f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \tag{2.6}$$

and, because of its range of $[0,1]$ as depicted in Fig. 2.3 a), is often used to predict probabilities as output. Often the generalized version of the sigmoid function is used, called the *softmax* function [93]

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}, \tag{2.7}$$

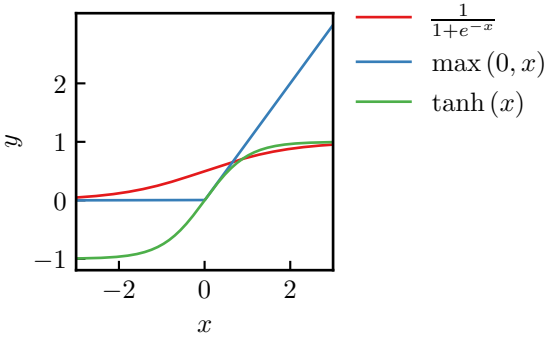


Figure 2.3 –
Illustration of activation functions. The three functions shown are commonly used activation functions in neural networks, all of which fulfill the requirement of nonlinearity. The sigmoid (red) and tanh (green) functions are similar to each other by having finite limits $y_{\text{sigmoid}} \in [0, 1]$ and $y_{\tanh} \in [-1, 1]$, respectively, and being continuous functions, whereas the ReLU function (blue) is not differentiable at $x = 0$ and the output is a left closed interval $y_{\text{ReLU}} \in [0, \infty)$.

where $\sigma(z)_i$ is the output of the i-th neuron. This forces the output of the neural network to be $\sum_i \sigma(z)_i = 1$ and is therefore a natural choice in multi-class classification problems, to learn the probability distribution.

The ReLU activation function is defined as the positive part of its argument

$$f(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else.} \end{cases} \qquad f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0. \end{cases} \tag{2.8}$$

as shown in Fig. 2.3 b). Compared to the sigmoid activation function it has advantages of efficient computation since only one comparison operation is needed, and it suffers less from the *vanishing gradient problem* [94]. These advantages allows the ReLU function to train deep neural networks more efficiently [94], and therefore has been the most popular activation function for deep neural networks [95].

2.1.3 Convolutional neural networks

A specific architecture of neural networks are the *convolutional neural networks* **CNN**'s. These networks are designed to extract locally confined features by sliding filters across the input data and condensing relevant information into a feature map. Therefore, CNNs are widely used in the application of image recognition and processing [96].

CNN's are usually constructed using different types of layers in the network, including pooling layers and fully connected layers, the first being the signature *convolutional layers*, which is the origin of the networks name. And here we want to focus on this particular layer. In this layer an input tensor is taken and divided into sub-regions. These regions are then fed into a single neuron, giving the output

$$z^l_{j,k} = \sigma\left(\sum_{l=-K}^{K}\sum_{m=-K}^{K} w^l_{l,m} z^{l-1}_{j+l,k+m} + b^l\right). \tag{2.9}$$

Here $z^{l-1}_{j+l,k+m}$ is the input of the convolutional layer, and K is the size of the kernel. The same kernel with the same weights w and biases b is applied to each of these sub-regions. The output is then collected into a *feature map* as shown in Fig. 2.4, which can then be further processed by applying pooling, fully connected or additional convolutional layers next. As the kernel is moved around the input data, special consideration must be given to the boundaries of the input and the possibility that part of the kernel may lie outside of the input data. The solution to this is either to keep the kernel always inside the boundaries at all times, or to use padding

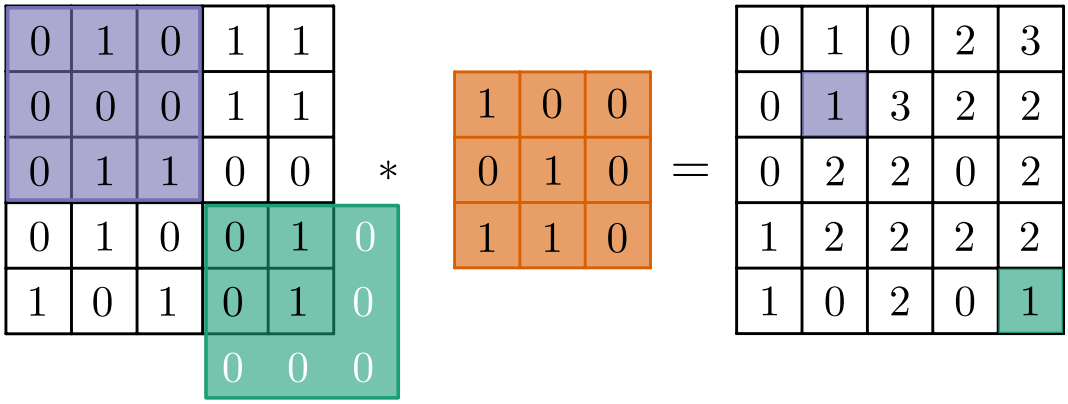


Figure 2.4 – **Illustration of the convolution operation** in a convolutional layer. Here the input is represented by a 5×5 matrix and the feature map (orange) by a 3×3 matrix. The feature map is shifted around the input, returning the sum of the element-wise multiplication of a subregion (blue) as a value. If the feature map is shifted to a position, where parts of the subregion (green) are not part of the input matrix, the outlying values are set to 0.

techniques. This means that the parts of the kernel that lie outside the input are filled with, for example, 0's in the case of zero-padding [97], as shown in Fig. 2.4.

This structure of the CNN leads to advantages over regular fully connected networks. First, using the kernel as input allows for a reduction in the number of free parameters in the networks. For example, assuming an input shape of 100×100, a fully connected layer would result in 10000 weights *per neuron*. However, a convolutional layer with a 5×5 kernel, would only require 25 weights *per feature map* [88]. This makes it easier to avoid problems in the training process caused by exploding or vanishing gradients, and makes it more robust against overfitting. Second, sharing the weights of the filters results in a translation-equivariant feature map [98].

2.2 Supervised learning

To train the neural decoder, we have chosen to use *supervised learning* as the training method. This means that we choose a loss function $L(y, \tilde{y})$ that depends on the output of the neurons y and the desired output $\tilde{y}$. The desired output $\tilde{y}$ is created together with the input of a training batch and is therefore commonly referred to as the *labels* of the training data.

2.2.1 Optimizer

The network is tuned using the *stochastic gradient descent method* **SGD** [99], an iterative optimization algorithm for finding local minima of differential functions. The idea is that in iteration takes a step in the opposite direction to the gradient of the current point, as shown in Fig. 2.5. At the end of this, a local minimum of the function will be found. The adjustment of the weights and biases is given by

$$\theta^{i+1} = \theta^i - \eta \nabla_\theta L(y, \tilde{y}), \tag{2.10}$$

where θ^i is the target parameter in the i-th iteration and η is the learning rate that determines the step size of the SGD. The gradient of the loss function can be calculated using Eq. (2.4) and 2.2 from the backpropagation algorithm.

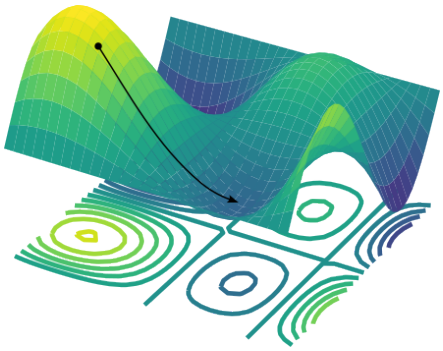


Figure 2.5 –
Visualization of the training process of a neural network in a loss landscape. At the beginning of the training the ANN starts at a random point in the landscape and with each iteration it moves against the local gradient according to the SGD algorithm. This 'rolling downhill' like movement continues until the network is being trapped in a minimum. This figure was inspired by [100].

The learning rate η generally determines the speed of the learning process, where the larger the learning rate, the faster the learning converges, while smaller learning rates reduce fluctuation during the training process and are therefore generally able to achieve higher accuracies in the fitting process. However, both learning rate extremes have also their own set of problems, small rates can more easily get stuck in local minima, while larger rates are more easily affected by the exploding gradient problem [88]. Therefore, modern ML techniques use adaptive learning rates that change over time.

One example for this is the ADAM optimizer [101]. This algorithm combines two different extensions of the SGD. The first is the introduction of momentum [102] into the update scheme. This means that the last update $\Delta\theta$ is kept and used to update the next step, which is then a linear combination of the $\Delta\theta$ and $\nabla L(y^L, \tilde{y})$.

$$\begin{aligned} m_\theta^{(t+1)} &= \beta_1 m_\theta^{(t)} + (1-\beta_1)\nabla L^{(t)} \\ \hat{m}_\theta &= \frac{m_\theta^{(t+1)}}{1-\beta_1}, \end{aligned} \tag{2.11}$$

with β_1 being the forgetting factor of the gradient. This allows for faster convergence towards a local minimum [103].

The second being an adaptive learning rate, which is set specifically for each parameter by dividing it by a running average of the magnitude of the recent gradient of the parameter [104]

$$\begin{aligned} v_\theta^{(t+1)} &= \beta_2 v_w^{(t)} + (1-\beta_2)\left(\nabla_\theta L^{(t)}\right)^2 \\ \hat{v}_\theta &= \frac{v_\theta^{(t+1)}}{1-\beta_2^t}, \end{aligned} \tag{2.12}$$

where β_2 is the forgetting factor of the second moments of the gradient. Putting these two methods together gives the update scheme

$$\theta^{(t+1)} = \theta^{(t)} - \eta \frac{\hat{m}_\theta}{\sqrt{\hat{v}_\theta + \epsilon}} \tag{2.13}$$

2.2.2 Loss function

As described in the previous section, the exact form of the loss function can vary and is therefore chosen to suit the problem. The two loss functions used in this thesis are the *mean square error* **MSE** [105] and the *categorical cross entropy* **CCE** [106].

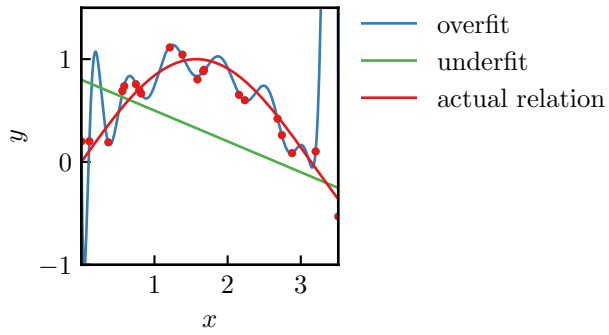


Figure 2.6 –
Faulty training of a neural network for a training data set (red dots). The overfitting case (blue curve) fits the function well to the training data, but fails to represent the actual underlying relation (red curve). Conversely, in the underfitting case (green curve), the fitted curve displays significant errors in comparison to the training data.

The MSE loss is given by the square sum of the difference between the labels and the predictions of the ANN

$$L_{\text{MSE}} = \frac{1}{n}\sum_{n}(y_n - \tilde{y}_n)^2 \,. \tag{2.14}$$

Due to its mathematical convenience, the MSE is one of the most commonly used loss functions in machine learning and the de facto standard setting. While the MSE is able to produce acceptable results for most problems, it has the disadvantage of heavily weighting outliers due to the quadratic term.

The CCE is a loss function specifically for multi-class classification problems, meaning that the output of the network returns the probabilities over C classes. It is assumed that the labels are represented in a one-hot fashion, so that only a single element in the label vector is non-zero and instead one. The CCE is given by

$$L_{\text{CCE}} = -\log\left(\frac{e^{\tilde{y}_p}}{\sum_j^C e^{y_j}}\right), \tag{2.15}$$

where the index p is determined by the non-zero element of the label vector. The advantage of this loss function is that it has a smooth and convex shape, which makes it easier for gradient-based optimizers to find the global minimum. The main disadvantage is the same as for the MSE, in that it is sensitive to outliers.

There are several challenges in training an ANN, one of which is the initialization of the weights and biases. The initial starting point of the training can determine whether the algorithm will converge at all [93], and if it does converge, the starting point can determine how fast the convergence is achieved and whether it is in a local or global minimum. An important requirement for the initialization, is the symmetry breaking of parameters between different nodes. If two hidden nodes with the same activation function are connected to the same inputs and have the same parameters, then a deterministic learning algorithm will constantly update both nodes in the same way.

The standard way of initialization is to draw values from a Gaussian distribution, where the actual choice of the Gaussian distribution doesn't seem to matter much [93].

An additional issue arises with overfitting or underfitting of data, as demonstrated in Figure 2.6. Overfitting occurs when the ANN produces near-perfect predictions on the trained data, but fails to do so with new data. This indicates that the model has not learned the fundamental underlying relationship, but rather has simply "memorised" the correct data. This may occur due to insufficient sample size for training, presence of noise in data, or a model with excessive complexity. Underfitting is the similar but opposite scenario, where the model is

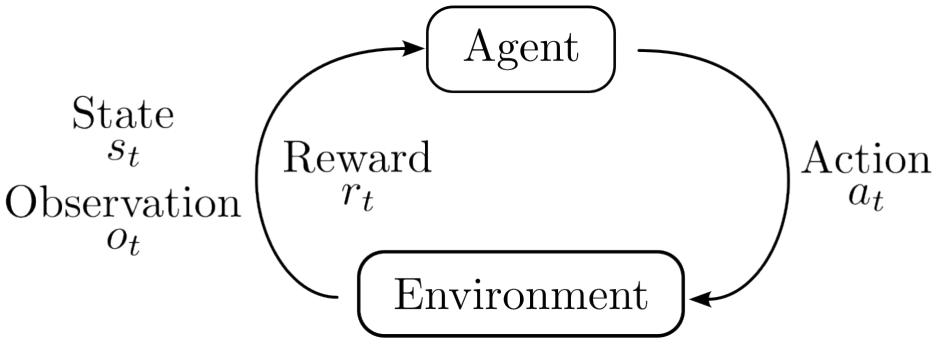


Figure 2.7 – **Visualization of the agent-environment interaction loop.** For an arbitrary timestep the agent receives an observation o_t based on the state s_t of the environment, along with a reward r_t that assesses the quality of the state. The agent then takes an action a_t that affects the environment and changes the state. The loop then starts again by returning the new state s_{t+1} and the corresponding reward r_{t+1} to the agent.

also unable to learn the underlying relationship, resulting in high prediction error rates. This often results from incomplete training or a model lacking sufficient complexity [93].

2.3 Reinforcement learning

The alternative machine learning method that will be employed is *reinforcement learning* **RL**. RL consists of two primary components, the agent and the environment. The environment refers to the agent's surrounding world, wherein it encounters and interacts with the problem, while setting the problem's rules. The agent is integrated into the environment and receives a form of observation o_t of the environment's state s_t. Based on this observation, the agent selects an action a_t that modifies the environment, causing a change. Upon a new state being achieved in the environment, the agent receives rewards, which is a measure for the quality of the environment's state.

Due to the straightforward design of the environment-agent interaction loop, it is possible to construct such a loop for almost any problem. This could be a basic physical toy model, such as the classical cartpole issue [107], which will be discussed in Section 6.1, or a complicated game like Go [83].

2.3.1 General terminology

Lets start with a clarification of the general terminology used in RL. The state $s_t \in \mathcal{S}$ of the environment is a complete description of the world at time step t, where no information is missing. It is typically represented as a real-valued vector, matrix or higher-order tensor. The observation $o_t \in \mathcal{O}$ is a partial description of the state S_t that is passed to the agent. In the case where the environment is fully observed we have $o_t = s_t$ [1]. The agent chooses an action a_t from an initially defined action space $a_t \in \mathcal{A}$. Depending on the environment, one can choose to define a discrete action space with a limited number of actions, e.g. possible moves in a game of Go, or a continuous action space.

[1] In much RL literature the state symbol s is used interchangeably with the observation o, even when we have only partially observed environments. In this thesis, we follow the notation of [108] and therefore choose to use the state s_t.

How the agent chooses the action is guided by a set of rules called **policy**. The policy is the mathematical construct behind turning an observation to an action and can be either deterministic or stochastic,

$$a_t = \mu(s_t), \qquad a_t \sim \pi\left(a_t|s_t\right). \tag{2.16}$$

While it is possible to describe the policy using any form of mapping from observation space to action space, such as lookup tables, deep reinforcement learning requires a parameterised policy π_θ that utilizes a computational function depending on a set of parameters θ. These parameters can be adjusted with an optimization algorithm. A suitable function choice is the previously discussed ANN 2.1.

The sequence of states s and actions a are called **trajectories**[2] τ

$$\tau = (s_0, a_0, s_1, a_1, \dots). \tag{2.17}$$

The initial state, denoted as s_0, is selected randomly from a start state distribution provided by the environment. The transition between two subsequent states is determined by the laws of the environment and solely relies on the prior state and action taken by the agent. Analogous to the policy, this transition may be either deterministic or probabilistic.

$$s_{t+1} = f\left(s_t, a_t\right) \qquad s_{t+1} \sim P\left(\cdot|s_t, a_t\right). \tag{2.18}$$

Based on the trajectories we can define the **reward** and **return** in the context of reinforcement learning. The reward is given by a reward function

$$r_t = R\left(s_t, a_t, s_{t+1}\right), \tag{2.19}$$

depending on the chosen action and the state before and after the action was applied. The reward function lacks an explicit form but is initially chosen according to the environment and learning objectives. In many cases, this means simplifying the function so it only relies on the current state, denoted as $t_t = R\left(s_t\right)$. The return, $R\left(\tau\right)$, is the sum of all rewards collected over the course of a trajectory

$$R\left(\tau\right) = \sum_{t=0}^{\inf} \gamma^t r_t, \tag{2.20}$$

where $\gamma \in (0, 1)$ is the discount factor. The discount factor here is a mathematical trick to make an infinite sum finite, by allowing the sum to converge. This corresponds to the intuitive idea that rewards received now are preferable to rewards in the future.

2.3.2 Value functions

One important concept in RL is assigning a state or state-action pair a value, which reflects the quality of the state considering the future of the environment. This gives rise to the *On-Policy Value Function*

$$V^\pi\left(s\right) = \underset{\tau\sim\pi}{E}\left[R\left(\tau\right)|s_0 = s\right], \tag{2.21}$$

which gives the expected return assuming one starts in the state s_0 and always follows the policy π.

[2] Alternatively used terms for trajectories are episodes or rollouts. We have opted for the trajectory term to keep the notation in line with [108].

Similar to this function one can define the *Action-Value Function*

$$Q^{\pi}(s,a) = \underset{\tau\sim\pi}{E}\left[R(\tau)\,|s_0 = s, a_0 = a\right], \tag{2.22}$$

which additionally takes into account that one also starts with the action $a_0 = a$. This starting action is arbitrary chosen and does not need to align with the policy π, unlike all following actions.

Both of these two functions can be combined together into the *Advantage Function*

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s). \tag{2.23}$$

The intuition behind this function is that it is not necessary to describe exactly how good an action is, only what its relative advantage is over other actions. In many cases, the Value-Function is estimated using an ANN [109].

Kinds of RL Different types of RL algorithms are defined according to generally accepted core concepts. The significant distinction is between model-free and model-based RL, which determines whether the agent has access to (or learns) a model of the environment. Since the model of the environment is generally not available to the agent, it must learn it completely from experience, which presents a challenging problem. Therefore, model-free techniques are generally more popular and more extensively researched.

The training of model-free RL is based on either the optimization of the policy π or learning of an optimal action-value function $Q(s,a)$. Comparing both methods, policy optimization is generally more stable and reliable, while Q-learning is more sample efficient [108]. In the following we will focus on policy optimization for model-free RL.

2.3.3 Policy optimization

The central idea behind the training of an RL agent is to find the optimal policy π^*, which maximizes the *expected return* $J(\pi)$. In order to calculate this return value one needs to know the probabilities over the trajectories. In case of a stochastic environment and a stochastic policy the probability of a single trajectory is given by the product of the policy and state transition probability

$$P(\tau|\pi) = \rho_0(s_0)\prod_{t=0}^{T-1} P(s_{t+1}|s_t,a_t)\,\pi(a_t|s_t), \tag{2.24}$$

where ρ_0 is the probability of starting in the state s_0. With this the expected return follows immediately as

$$J(\pi) = \int_{\tau} P(\tau|\pi)\,R(\tau) = \underset{\tau\sim\pi}{E}[\tau]. \tag{2.25}$$

The optimal policy can therefore be defined as

$$\pi^* = \arg\max_{\pi} J(\pi). \tag{2.26}$$

To reach this optimal policy, the parameters of the policy will be adjusted incrementally using *stochastic gradient ascent* **SGA**

$$\theta_{k+1} = \theta_k + \alpha\nabla_{\theta}J(\pi_{\theta})\,|_{\theta_k}. \tag{2.27}$$

This technique is analogous to the SGD in Sec. 2.2, with the slight difference of an sign switch, instead of minimizing a cost function the goal is here to maximize the expected return. The similarity can be extended even further, since most modern RL techniques also incorporate the improved optimizer algorithms like the ADAM optimizer.

In order to calculate the gradient of the expected return, we have to rewrite the function in a form, where it can be numerically approximated. This can be accomplished by following this approach

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau\sim\pi_\theta}{E}[R(\tau)] \\
&= \nabla_\theta \int_\tau P(\tau|\theta) R(\tau) \\
&= \int_\tau \nabla_\theta P(\tau|\theta) R(\tau) \\
&= \int_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) R(\tau) \\
&= \underset{\tau\sim\pi_\theta}{E}[\nabla_\theta \log P(\tau|\theta) R(\tau)] \\
&= \underset{\tau\sim\pi_\theta}{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(t)\right],
\end{aligned} \tag{2.28}$$

where in the end, we obtain an expectation as an expression that we can estimate numerically by calculating the average over a set of N trajectories $D = \{\tau_i\}_{i=1,\dots,N}$ consisting of N trajectories

$$\hat{g} = \frac{1}{|D|} \sum_\tau \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(t). \tag{2.29}$$

The different trajectories are obtained by letting the agent act on the environment according to the policy π_θ.

A special property of the resulting equation for the expected return is that we can modify the expectation value using any function $b(s)$, which only depends on the state of the environment, resulting in the expression

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{E}[\nabla_\theta \log \pi_\theta(a_t|s_t)(R(\tau) - b(s_t))]. \tag{2.30}$$

This modification is made possible through the expected Grad-Log-Prob Lemma showing

$$\underset{a_t\sim\pi_\theta}{E}[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)] = 0, \tag{2.31}$$

the derivation of this lemma is shown in App. A.1.

A common choice for a the function b is the advantage function $A^\pi(s_t, a_t)$. Taking a step in the policy gradient direction should enhance the likelihood of selecting an above-average action and subsequently reduce the selection of below-average actions. Since the advantage function, which is defined as $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, aims to determine whether the action was preferable to the default policy behavior, it is a natural choice to use it for the gradient in Eq. (2.30), so that it only points in the direction of increased π if the advantage is positive $A > 0$. In theory, using the advantage function yields the lowest possible variance [110], while in practice the advantage function is not known and has to be approximated.

This is done by approximating the value function V with an ANN and defining the temporal difference residual of V with discount

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t), \tag{2.32}$$

which can be considered as an estimate of the advantage function of the action a_t [108]. From this the *generalized advantage estimator* **GAE** [110] is derived as an exponentially-weighted average of the residuals

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+1}^V. \tag{2.33}$$

The GAE makes an compromise between bias introduced by $V^{\pi,\gamma}$ and variance of the GAE for $0 < \lambda < 1$, controlled by the parameter λ.

While there are different methods to estimate the value function, the chosen method in this work is the sometimes called Monte Carlo approach [108] of estimating. The value function is represent by an ANN and optimized via SGD minimizing the mean-squared error

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(V_\phi\left(s_t\right) - R_t\right)^2, \tag{2.34}$$

where $\hat{R}_t$ is the discounted sum of rewards as given is Eq. (2.20) and D_k is a collected set of trajectories τ.

2.3.4 Proximal policy optimization

Among the diverse reinforcement learning techniques at our disposal, we will focus on the implementation of *proximal policy optimisation* **PPO** [111]. PPO belongs to the category of model-free RL techniques and acts as an on-policy algorithm that can be applied to both discrete and continuous action spaces.

The aim of this particular RL technique is to make the greatest possible improvement while avoiding a decline in performance. To achieve this, the policy update in the PPO is defined as follows

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{E} \left[L\left(s, a, \theta_k, \theta\right)\right], \tag{2.35}$$

where multiple steps of SGD over mini batches are performed to maximize the objective. Here L is given by

$$L\left(s, a, \theta_k, \theta\right) = \min\left(\frac{\pi_\theta\left(a|s\right)}{\pi_{\theta_k}\left(a|s\right)} A^{\pi_{\theta_k}}\left(s, a\right), g\left(\epsilon, A^{\pi_{\theta_k}}\left(s, a\right)\right)\right), \tag{2.36}$$

and where

$$g\left(\epsilon, A\right) = \begin{cases} (1+e)A & A \geq 0 \\ (1-e)A & A < 0 \end{cases}. \tag{2.37}$$

To gain a better understanding of the role of function $g\left(\epsilon, A\right)$, it is useful to consider the two potential scenarios.

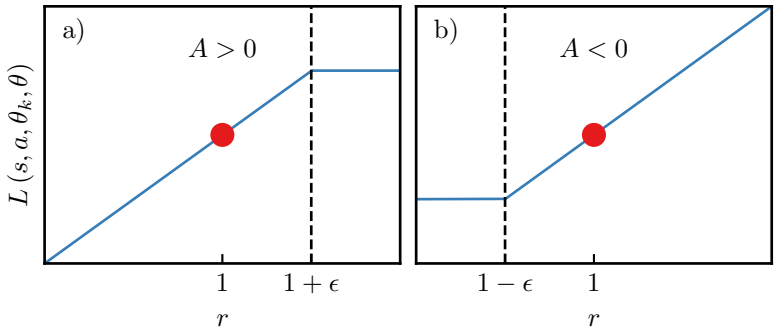


Figure 2.8 – **Clipping of the surrogate function**. The plots show the values of the surrogate function L for a single time step depending on the probability ratio, where (a) shows the case for a positive advantage function $A > 0$ and (b) for a negative advantage function $A < 0$. The red dot on the curve indicates the beginning of the optimization process.

Positive advantage If we assume that the advantage function returns a positive value, the function (2.36) reduces to

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon,)\right) A^{\pi_{\theta_k}}(s, a), \tag{2.38}$$

which has the consequence that the objective L increases, when the action a becomes more likely (when $\pi_\theta(a|s)$ increases). This growth is then stopped by the min-function when $\pi_\theta(a|s) > (1+\epsilon)\,\pi_{\theta_k}(a|s)$, only allowing a maximum value $(1+\epsilon)\,A^{\pi_{\theta_k}}(s, a)$) as illustrated in Fig. 2.8 a).

Negative advantage The same process can be followed, when the advantage function returns a negative value. The function (2.36) then reduces to

$$L(s, a, \theta_k, \theta) = \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1-\epsilon,)\right) A^{\pi_{\theta_k}}(s, a), \tag{2.39}$$

which has the consequence that the objective L now decreases, when the action a becomes less likely (when $\pi_\theta(a|s)$ decreases). This fall is then stopped by the max-function when $\pi_\theta(a|s) < (1-\epsilon)\,\pi_{\theta_k}(a|s)$, only allowing a minimum value of $(1+\epsilon)\,A^{\pi_{\theta_k}}(s, a)$) as illustrated in Fig. 2.8 b).

So as it can be seen, this clipping function $g(\epsilon, A)$ serves as a regulator by removing the incentives for new policies to strive far away from the old policies, independent from the direction. The hyperparameter ϵ controls the allowed distance between the policies.

2.3.5 Challenges in reinforcement learning

While reinforcement learning offers flexibility in learning various problems, it comes with increased difficulties in the training process. Because ANNs are used to approximate the desired function, RL carries the same problems as supervised learning and can suffer from overfitting, initialization problems, and the exploding and vanishing gradient.

Additionally it also has its own set of problems caused by the agent learning through a process of trial and error and generating its own training data set to learn from. This creates the problem, that the agent is dependent on a monotonic improvement during the training process, which can lead to a quick performance degradation [112].

As long as the agent is improving, it supplies itself with useful and relevant data. However, if the performance drops, it will also affect the generated data, which may then be insufficient for the agent to recover quickly. This, in turn, causes the performance of the agent to drop and starts a cycle which that can lead to complete failure of the training process.

In order to avoid this, it is necessary to avoid initial performance degradation that can be caused by errors in the function approximation [113–115], poor exploration [116–118] or even supoptimal hyperparameters [119, 120]. To reduce these potential errors, an extensive optimization of hyperparameters and training algorithms is often necessary, but even then performance can degrade due to the stochastic nature of the training process. Therefore, it is common practice to train agents in parallel on the environments [121] or use transfer learning [34, 122] to stabilize the training process.

Chapter 3

Quantum error correcting codes

This chapter introduces quantum error correcting codes as a method to protect a logical qubit over time in order to achieve fault-tolerant computing. The chapter commences with a review of classical error correction and provides a grounding in terminology, core concepts for encoding information, and a brief discourse on error correcting code quality assessment. This continues in the field of quantum error correction, assessing transferable concepts from the classical field and discussing new challenges and how to deal with them. Subsequently, we detail a general way to construct a quantum error-correcting code using the stabilizer formalism introduced by [123]. In the final section of this chapter, we examine topological codes, which are a type of stabilizer codes that are particularly advantageous for experimental implementation. In addition to defining their general properties, we discuss two concrete examples of implementation, namely the toric code introduced by Kitaev [124] and the rotated surface code [125].

3.1 Classical versus quantum error correction

For classical technologies data is represented using binary encoding. Information is stored by stringing bits of values $\{0, 1\}$ together, forming the *logical information* one wants to store. Due to external influences, these bits can be perturbed and information can be lost. Therefore, to keep the information stable over an extended period of the time, error correction is needed. The basic principle of error correction is to introduce *redundancy* to the system, by encoding information into multiple bits, mapping the logical information onto multiple qubits $\{0, 1\} \rightarrow \{c_1, c_2 \in (\mathbb{C}^2)^n\}$, where c_1, c_2 are called *codewords* and being strings of n bits representing the logical states of the encoded bit. The encoding follows a set of rules described as the *error correction code* **ECC**. It is important to note that a bit spans the entire space $\mathbb{C}^2$, while the codewords only form a subspace of $(\mathbb{C}^2)^n$ called *codespace* $\mathcal{C}$. Therefore, if some noise occurs that corrupts the information and takes a bit string from a codeword to some other string $c \notin \mathcal{C}$, it is possible to decode the string back to initial codeword and thus preserve the information.

Classical coding theory mostly concentrates on *linear codes* [123]. They are a subclass of ECC's defined by the relation between the initial logical information and the encoded redundancy. Assuming that we have k bits, which can be represented as a binary vector v, that we want to encode using n bits, then the encoded data given by linear code is an n vector $s = Gv$, where G is the $n \times k$ generator matrix G (with entries from $\mathbb{Z}_2$) representing the code. The columns of the generator matrix form a basis for the k-dimensional coding subspace of the n-dimensional binary vector space. The resulting initial information is now encoded as a codeword in the form of the vector s.

After the encoding of some codeword, we can now consider that t bit-flip errors occur. These errors can be described as an error vector e with the same dimension as the codeword. The new state is given by $s' = s + e$. Since we typically do not know what kind of error has occured,

we perform a measurement using a so called parity matrix P, returning an error syndrome Ps'. The parity matrix is the dual matrix of the generator matrix and therefore has the property $Ps = 0$, which simplifies the error syndrome

$$Ps' = P(s + e) = Pe \tag{3.1}$$

to be only dependent on the error and not the initial codeword. From the resulting syndrome, a decoding strategy performs a recovery operation r to return the vector to a codeword $P(e+r) = 0$. If one gets back to the initial codeword, the decoding was successful.

An example of a classical error correcting code is the three-bit repetition code, where information is encoded by copying the value of each bit onto two extra bits $\{0, 1\} \rightarrow \{000, 111\}$, with $\{000, 111\}$ being the codewords. Assuming that we want to encode two bits at the same time, the generator matrix of this code is given as

$$G = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}. \tag{3.2}$$

Decoding is the performed following the majority rule, which when applied will set the value of all bits to the most common value. This way that the original information is preserved, even if there is an error on a single bit e.g.

$$v = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \xrightarrow[encode]{s=Gv} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \xrightarrow[noise]{s'=s+e} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \xrightarrow[decode]{s'+r} v = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{3.3}$$

Following the ECC it is easy to see that errors on two or more bits will lead to the destruction of the original information

$$v = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \xrightarrow[encode]{s=Gv} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \xrightarrow[noise]{s'=s+e} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \xrightarrow[decode]{s'+r} \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{3.4}$$

Here one can intuitively see, that increasing the numbers of bits used for encoding would decrease the probability of incorrect decoding, as more errors are needed to overturn the decoding majority rule. This is described by the *Hamming distance* [126], which describes the distance between two vectors as the minimum number of bit-flips needed to transform one vector into another. So assuming t single bit flip errors occur on a codeword, it takes the vector a distance t away from its original value. In order to be able to correct an error, the distance between two codewords has to be at least $2t + 1$. The distance between two codewords is also called *code distance* d and represents a rough measure for the error correcting capabilities of the code.

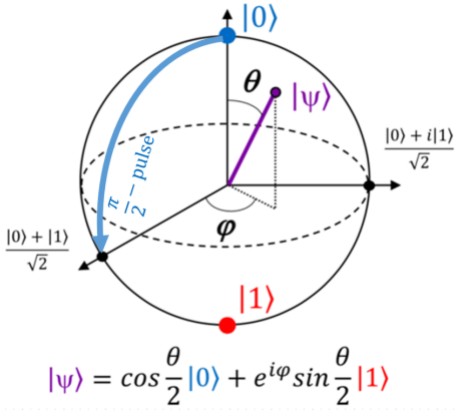


Figure 3.1 – **Bloch sphere representation** of a qubit. Any pure state of a two level quantum system $|\Psi\rangle$ (violet) can be represented a point on a sphere where the north pole represents the state $|0\rangle$ (blue) and the south pole the state $|1\rangle$ (red). The position on the sphere is defined via the angles ψ and θ, which follow from the linear coefficients of a pure state. Taken from [128] under CC0 1.0 licence.

A general description of error correcting codes is given by the $[n, k, d]$ notation, where n is the number of bits used to represent a codeword, k is the number of encoded logical bits and d is the *code distance*. Following this notation the three-bit repetition code would be described as a [6,2,6] code.

Now that we have discussed how ECC are constructed and function, we need some measures to compare the quality of different codes. For this, the threshold theorem [51] comes into mind. Given an ECC with n physical bits and some physical error rate of p, to be considered a viable code, there must be a probability threshold $p_{\text{th}} \leq \frac{1}{2}$, such that the logical error rate p_{log} satisfies the condition

$$p_{\text{log}}(n, p) \leq ce^{-\alpha d} \qquad \forall p \leq p_{\text{th}}, \tag{3.5}$$

where it is assumed that d scales with n. This implies that as long as the physical error rate is below threshold, it is possible to scale up the ECC towards infinity and thereby decreasing the logical error to zero $\lim_{d\to\infty} p_{\text{log}} \to 0$. The actual error threshold depends on the ECC, as well as the utilized decoder, where generally a threshold as high as possible is desired.

Closely related to this is the pseudothreshold [127], which gives a practical limit for whether it is advantageous to deploy the ECC or not. By setting the pseudothreshold as the point where the logical error rate p_{log} equals the physical error rate p, it marks the limit for which the ECC still manages to reduce the error rate. Going above the pseudothreshold actually increases the logical error rate compared to the physical rate. The pseudothreshhold value depends on the code distance and, in the limit of large code distance, recovers the error threshold

$$\lim_{d\to\infty} p_{\text{pseudo}}(d) = p_{\text{th}}. \tag{3.6}$$

Now, when we move on to quantum computing and construct *quantum error correcting codes* (**QECC**), the basic block of information changes from a bit into a *qubit*, which is a normalized vector of a two-dimensional Hilbert space $\mathcal{H}_2$. While bits are in either the '0' or '1' state, qubits can exist in coherent superposition of the two states, which are denoted using the Dirac bra-ket notation as $|0\rangle$ and $|1\rangle$.

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{3.7}$$

where α and β are complex numbers and satisfy the condition $|\alpha|^2 + |\beta|^2 = 1$. Measuring the qubit will project it onto the state $|0\rangle$ or $|1\rangle$ state with probability $|\alpha|^2$ or $|\beta|^2$, respectively.

3.1.1 Quantum errors

In classical systems the only errors that can appear are bit flip errors, which takes a '0' state to a '1' state and vice verse. This changes in the quantum case, since the qubit can take on continuous values for α and β as defined in Eq. (3.7) and therefore has a continuum of states. It immediately follows that we can have an infinite number of different errors, which takes a qubit from one possible state to another. This behavior can be illustrated using a geometrical representation called the *bloch sphere* [128], where the qubit state is described as

$$|\Psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\psi}\sin\frac{\phi}{2}|1\rangle. \tag{3.8}$$

Using this representation the state of a qubit can now be represented as a point on the surface of a sphere, specified by the angle θ and ϕ, as shown in Fig. 3.1. Now looking at qubit errors, the simplest form is coherent errors, which take a qubit from one point of the bloch sphere to another. These errors can be described using an unitary operator $U(\delta\theta, \delta\phi)$, which evolves the qubit state

$$U(\delta\theta, \delta\phi)|\Psi\rangle = \cos\frac{\theta+\delta\theta}{2}|0\rangle + e^{i\psi}\sin\frac{\phi+\delta\phi}{2}|1\rangle, \tag{3.9}$$

to a new point on the bloch sphere given by the coordinates $\theta + \delta\theta$ and $\phi + \delta\phi$, furthermore highlighting the existence of continuous errors.

This creates a challenge for developing error correcting codes, since the theory of classical ECC are based on discrete values and thereby wouldn't be applicable in this situation. To circumvent this problem it is necessary to digitize the errors, reducing them to a finite set. For this we want to rewrite the coherent noise 3.8, by expanding it in the Pauli basis. Here we use the notation

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{3.10}$$

With this the coherent noise can always be rewritten as a linear combination of the Pauli matrices [129] and the 2×2 identity matrix I

$$U(\delta\theta, \delta\phi)|\Psi\rangle = \alpha_I I|\Psi\rangle + \alpha_X X|\Psi\rangle + \alpha_Y Y|\Psi\rangle + \alpha_Z Z|\Psi\rangle, \tag{3.11}$$

where $\alpha_{I,X,Y,Z}$ are the linear coefficients. Now, performing a non disruptive measurement over what type of error has occurred, will cause the superposition to collapse into $|\Psi\rangle$ with probability $|\alpha_I|^2$, or $X|\Psi\rangle$, $Y|\Psi\rangle$, $Z|\Psi\rangle$ with probability $|\alpha_{\mathrm{X,Y,Z}}|$ [123]. This measurement preserves the logical state and will described more closely in Sec. 3.3. In any case the previously continuous error has been reduced to a single error.

X errors The Pauli X errors acts by mapping $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$ and therefore the action on a general quibt state is given by

$$X|\Psi\rangle = \alpha X|0\rangle + \beta X|1\rangle = \alpha|1\rangle + \beta|0\rangle, \tag{3.12}$$

because of this, X errors are regarded as the quantum counterpart to the classical bit flip error, and therefore also generally referred to as *bit flip errors*.

Z errors In comparison to the X errors before, the Pauli Z errors do not have a classical comparison. They act by mapping $Z|0\rangle = |0\rangle$ and $Z|1\rangle = -|1\rangle$ and the action on the general qubit state is

$$Z|\Psi\rangle = \alpha Z|0\rangle + \beta Z|1\rangle = \alpha|0\rangle - \beta|1\rangle. \tag{3.13}$$

They are generally refereed to as *phase flip errors*.

Y errors Pauli Y errors can be expressed as a bit flip and phase flip error occurring at the same time $Y = iXZ$, resulting in

$$Y|\Psi\rangle = \alpha iXZ|0\rangle + \beta iXZ|1\rangle = i\left(\alpha|1\rangle - \beta|0\rangle\right), \tag{3.14}$$

which only differs by a global phase shift of i, which cannot be distinguished in quantum mechanics, since the measurement results are the same [51].

3.1.2 Quantum challenges

Thanks to the digitization of the errors we are able to apply theories from classical coding theory to develop quantum error correcting code. But compared to the classical case there are some additional challenges, which prevent us from simply taking a classical error correcting code. the occurrence of the unique error type *phase flip error*, as previously discussed. Furthermore, we also have to deal with wavefunction collapse and the no-cloning theorem for quantum states.

No-cloning theorem The theorem states that there is no unitary operator U_{clone}, which evolves a state in the way that

$$U_{clone}\left(|\Psi\rangle \otimes |0\rangle\right) \rightarrow |\Psi\rangle \otimes |\Psi\rangle, \tag{3.15}$$

where $|\Psi\rangle$ is the state to be reproduced [130]. This makes it impossible to just replicate states to introduce redundancy in the system, and therefore classical error correcting codes like the 3-bit repetition code cannot be used in the quantum case. But it is still possible to introduce redundancy into the system, by entangling multiple physical qubit into a single logical qubit

$$|0\rangle \rightarrow |\overline{0}\rangle = |00\rangle = |0\rangle \otimes |0\rangle \tag{3.16}$$

$$|1\rangle \rightarrow |\overline{1}\rangle = |11\rangle = |1\rangle \otimes |1\rangle, \tag{3.17}$$

where $|\overline{0}\rangle$, $|\overline{1}\rangle$ are denoted as the logical qubits and $|00\rangle$, $|11\rangle$ are the respective codewords. This does not violate the no-cloning theorem, since an arbitrary codeword will be a linear superposition of these two states and does not carry a cross term.

$$|\Psi\rangle_L = \alpha|\overline{0}\rangle + \beta|\overline{1}\rangle \neq \left(\alpha|0\rangle + \beta|1\rangle\right)^{\otimes 2} \tag{3.18}$$

Wave collapse In classical computing, it is a simply task to measure a desired bit to determine its state without compromising it. In the quantum case it is more difficult, since measurement of the state of a qubit will cause a wavefunction collapse and will erase the encoded information. [123]. So we are unable to determine the state of a qubit during the error correcting process and have to carefully choose measurements, that do not cause a collapse of the wavefunction, while providing information useful for error correction.

So in order to create a working quantum correcting code, we need to introduce redundancy via entangling of qubits and perform indirect measurements to detect errors. Assuming we have three entangled qubits and only consider bit flip errors creating the *3-qubit code*, we can

Error	$Z \otimes Z \otimes I$	$Z \otimes I \otimes Z$	$I \otimes Z \otimes Z$
$X \otimes I \otimes I$	-1	-1	+1
$I \otimes X \otimes I$	-1	+1	-1
$I \otimes I \otimes X$	+1	-1	-1

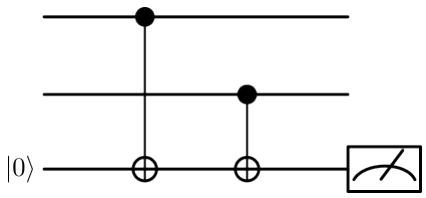


Figure 3.2 – **Parity measurement for a three qubit code**. On the left side the table shows all possible parity measurement results for single bit flip errors on three bit qubit code. On the right two data qubits are entangled with an ancilla qubit, which will be measured. The measurement will detect even parity, returning a +1 eigenvalue if measured and -1 in cases of uneven parity.

perform a projective measurement of the form Z_iZ_j to determine whether the measured state is part of the codespace $|\Psi\rangle \in \mathcal{C} = span\{|000\rangle, |111\rangle\}$ or in an error subspace $|\Psi\rangle \in \mathcal{F}$. This measurement is called a *parity measurement* and returns a ± 1 eigenvalue

$$Z_1Z_2|\Psi\rangle_L = Z_1Z_2\left(\alpha|000\rangle + \beta|111\rangle\right) = (+1)\,|\Psi\rangle_L \tag{3.19}$$

$$Z_1Z_2|\Psi\rangle = Z_1Z_2\left(\alpha|100\rangle + \beta|011\rangle\right) = (-1)\,|\Psi\rangle \tag{3.20}$$

depending whether the two investigated qubits have an even/uneven parity. The returned eigenvalues are called the *syndrome* of the code and tells us if an error has occurred. Since we measure the parity of the qubit pair, it indicates a bit flip of one of the qubits, without telling us which qubit. The table in Fig. 3.2 shows the different possible parity measurements and their respective syndromes for all the possible single bit flip errors for the three qubits code.

In actual experiments the parity measurement is performed by adding an ancilla qubit, which is prepared to be in state $|0\rangle$ and is entangled via a CNOT gate to the two qubits of the parity measurement as depicted in Fig 3.2. After entangling a projective measurement is performed on the ancilla qubit to return the ± 1 eigenvalue.

This example can be seen as a simple (and not necessarily nonsensical) QECC and, similar to classical error correcting codes, the properties can be described using the same [[n,k,d]] formalism. The difference is now that double square brackets are used to indicate a QECC instead of a classical one. So our example would be a [[3,1,3]] code, since $n = 3$ qubits are used to encode $k = 1$ logical state with a code distance of $d = 3$, since it needs at least three bit flip errors to go from one logical state to another. It is important to note that this only applies to bit flip noise, since phase flip errors cannot be detected in the 3-qubit code.

With these techniques it is possible to create a QECC, which allows active error correction for both bit and phase flip errors. The first conceived QECC of this kind was the *Shor code* or *9 qubit code*, which was introduced in a seminal paper in 1996 by Calderbank, Shor and Smolin [131]. As the name suggest, the code utilizes 9 qubits in order to be able to correct a single bit flip, phase flip or one of each, occurring on any of the 9 qubits.

3.2 Physical noise

After defining which errors can occur and how they can be detected, we want to define how they occur, which is defined by an underlying noise model. In this work we will generally only consider *independent* and *identically distributed* (**iid**) noise models. So we assume that the noise affects on every qubit individually. In general, the error models are structured in a way that there is a probability of p_{err}, that an error of any kind appears on a qubit.

We consider three different noise models, which belong to this definition. The first one being

the bit flip noise, which can be express in operator sum formalism [132] as

$$\rho \to (1 - p_{\text{err}})\, p_{\text{err}}\, (X \rho X)\,, \tag{3.21}$$

here p_{err} is the probability that an bit flip error occurs on a qubit and ρ is the density operator. While this error model is not always a good assumption compared to actual error noises in experiments, it is extensively used in benchmarking and designing decoders, due to its simplicity.

The second error model is the depolarizing noise model, which also includes phase flip errors and Y errors, instead of just bit flip errors

$$\rho \to (1 - p_{\text{err}})\, \frac{p_{\text{err}}}{3}\, (X \rho X + Y \rho Y + Z \rho Z)\,. \tag{3.22}$$

As one can see the different error types are equally distributed, each having a chance of $\frac{p_{\text{err}}}{3}$ to occur. Since Y errors can occur, this model is described as a correlated noise model.

The final noise model is the a combination of depolarizing noise and additional syndrome measurement errors. As shown in Fig. 3.2 b), the measurement of syndromes involves an additional ancilla qubit, which can also be subjected to errors. Therefore, it is possible to receive a false-positive or false-negative measurement. For this error model we use a phenomenological noise model, where the ancilla qubits are also subjected to depolarizing noise, but propagation of errors between data and ancilla qubits are neglected.

3.3 Stabilizer codes

Now we want to generalize the procedure we introduced in the last chapter in order to design QECC more efficiently. This generalization was done by David Gottesman [123] by introducing a subgroup of QECC called *stabilizer codes*. This well-studied class of QECC is expected to play a big part in realizing fault-tolerant quantum computing, especially since the surface code belongs to this class of QECC [65]. Stabilizer codes introduce operators called *stabilizers* S_i, and are defined by the following properties.

First one being, that all stabilizers must return an eigenvalue of +1 when applied to a logical state $|\Psi\rangle_L$. This means that each stabilizer has the action $S_i|\Psi\rangle_L = +1|\Psi\rangle_L$ for every possible state of $|\Psi\rangle_L \in \mathcal{C}$. This is generally described as the operators stabilizing the logical state.

Second one being, that stabilizers are elements of the Pauli group $S_i \in G_n$, where G_n is the Pauli group over n-qubits. Therefore, as a property of the Pauli group, two elements $G_\alpha, G_\beta \in \mathcal{P}$ either commute $[G_\alpha, G_\beta] = 0$ or anti-commute $\{G_\alpha, G_\beta\}$. If the elements anti commute they also share an eigenbasis. From this follows that if a stabilizer anti-commutes with an error $\{S, E\} = 0$, it would return an eigenvalue of -1

$$S_i\, (E|\Psi\rangle_L) = -E S_i |\Psi\rangle_L = -E|\Psi\rangle_L, \tag{3.23}$$

indicating the existence of an error, whereas if the stabilizer and error commute $[S, E] = 1$, an eigenvalue of +1 is returned

$$S_i\, (E|\Psi\rangle_L) = E S_i |\Psi\rangle_L = E|\Psi\rangle_L \tag{3.24}$$

and no error is detected.

And finally, all stabilizers have to commute with each other, so that $[S_i, S_j] = 0$ for all i and j. This allows simultaneous measurement of stabilizers. With these properties the stabilizers form an Abelian subgroup $\mathcal{S}_n \subset G_n$.

So from this follows that for a [[n,k,d]] QEC we can measure $m = n - k$ stabilizers, since the number of measurements is determined by the redundancy of the system, ergo the number of ancilla qubits m. This now presents us with the task to construct the stabilizer operators in a way that we get a minimal representation of the stabilizer group

$$\mathcal{S} = \langle S_1, S_2, ..., S_m \rangle \,, \tag{3.25}$$

so that no stabilizer S_i can be expressed as a product of the other elements S_j. Applying these stabilizer measurements, then again returns a syndrome of the size m.

Based on these properties the previously introduced parity measurement would be a stabilizer operator $S_1 = Z_1 \otimes Z_2$ and $S_2 = Z_2 \otimes Z_3$ for the [[3,1,3]] code. It fulfills all conditions, since it consists solely of Pauli operators, stabilizes all logical states and commutes with itself $[S_1, S_2] = 0$.

Additionally, we can define operators which act on the logical states of the code. An [[n,k,d]] code has 2k logical Pauli operators, since for each logical qubit, there exist a logical Pauli $\overline{X}_i$ and $\overline{Z}_i$ operator. Each pair of logical operators commute with all code stabilizers $\mathcal{S}$ and at the same time they anti commute with each other so that,

$$\left[\overline{X}_i, \overline{Z}_i\right]_+ = \overline{X}_i\overline{Z}_i + \overline{Z}_i\overline{X}_i = 0, \tag{3.26}$$

for all logical qubits i.
Since the logical operators commute with the stabilizers, it immediately follows that any product of a logical operator $\overline{L}_i$ and a stabilizer $\mathcal{S}_j$ will also be a logical operator, this is clear from the fact, that a stabilizer maps a logical state onto its $+1$ eigenspace. Therefore

$$\overline{L}_i S_j |\Psi\rangle_L = \overline{L}_i |\Psi\rangle_L, \tag{3.27}$$

meaning that the logical operators are not uniquely defines.

3.4 Topological surface codes

While we now have a general description for the construction of QECC, the challenge is finding a commuting set of stabilizers that enable the detection of errors without disturbing the encoded information. Finding these sets is a non-trivial problem and requires special code construction to find the stabilizers.

To this end we are going to discuss *topological surface codes* [124], which belong to the stabilizer codes and use two dimensional qubit lattices to protect the quantum information. This is achieved by defining topological (non local) logical operators, so that code space solely depends on the topology of the lattice, resulting in a code distance equal to the lattice size $d = L$.

Additionally, these codes only use local stabilizers, meaning that the stabilizer only use nearest-neighbour interactions. This allows for easier implementation in the case of surface codes [65] and has already been implemented, as shown by Google's Sycamore processor [57], which implemented nearest neighbour interaction to ensure forward compatibility with surface codes, and also the IBM processors' implementation is based on a hybrid surface/Bacon-Shor code [133]. In this section we will discuss the two examples of topological surface codes, these being the toric code and the RSC.

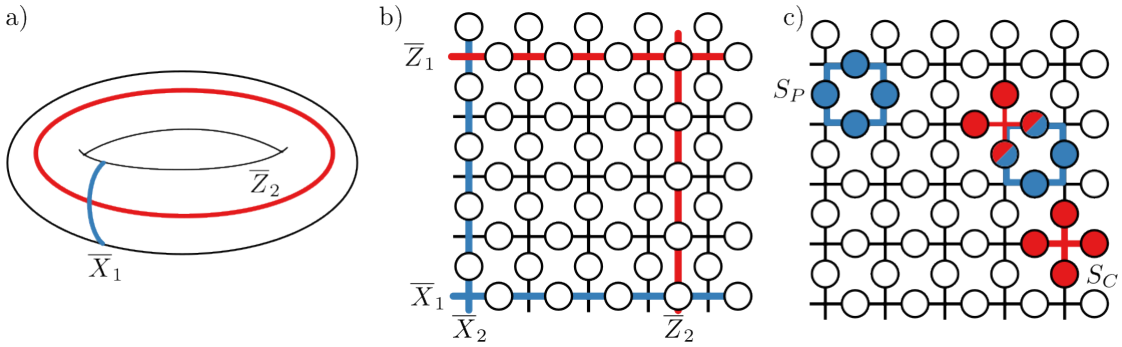


Figure 3.3 – **Topology of the toric code**. The logical qubits are encoded as non-trivial loops running around the torus (a). On the lattice the qubits (white circles) are placed on the links of a square lattice with periodic boundary conditions allowing the definition of 4 different logical operators (b), spanning from one side of the lattice to the other. These being two Pauli-X operators (blue) and two Pauli-Z operators (red). Stabilizers consist of 4 Pauli operators, forming either a plaquette (blue) or a vertex (red) on the lattice (c). Plaquette and cross operators commute with each other, since they can only overlap on two qubits (blue/red).

3.4.1 Toric code

The toric code was the first example of such a topological surface code and was introduced in 1998 by Kitaev [124]. The name stems from the fact, that it is defined on a square lattice forming a torus, which is just a square lattice with periodic boundary conditions. As a topological surface code, the logical qubits correspond to non-local degree of freedom [124] and can be understood as non-trivial loops around the torus as seen in Fig. 3.3 a). Since we can draw two non-trivial loops around the torus (one around and one trough the hole), we can encode two logical qubits in the toric code.

The qubits to encode the logical state are placed on the links of the lattice, so that a lattice of size L would contain $2L^2$ qubits as depicted in Fig.3.3 b).
The stabilizer operators of the toric code are locally defined and consist of a product of 4 operators, these being either 4 X operators to form a *vertex operator* S_V or 4 Z operators for a *plaquette operator* S_P and are defined as

$$S_V = X_u \otimes X_d \otimes X_l \otimes X_r, \tag{3.28}$$

$$S_P = Z_u \otimes Z_d \otimes Z_l \otimes Z_r, \tag{3.29}$$

where u, d, rl, r stand for up, down, left, right, respectively. The names of the operators are given based on the geometric pattern they display on the lattice as shown in Fig. 3.3 c). Based on the lattice we can define L^2 different plaquette and cross operators.

To make sure that we actually get a stabilizer code, we have to confirm that all stabilizer operators commute with each other. Since X operators and Z operators commute trivially with themselves $[X_i, X_j] = [Z_i, Z_j] = 0$ for all i, j, it immediately follows that all plaquette and vertex operators commute with themselves $[P_i, P_j] = [V_i, V_j] = 0$. This leaves the commutation of the plaquette and vertex operators, which also becomes trivial in the case, that the operators do not overlap since $[X_i, Z_j] = 0$ for $i \neq j$. The other case is that the operators coincide on a qubit, causing it to pickup a -1 from the anticommutation of the Pauli operators $[X_i, Z_i]$, but since the plaquette and vertex can only overlap on two qubits at the same time, the sign cancels out. Therefore we have $2L^2$ commutable stabilizer operators, which define our code.

Now that we have defined the stabilizers, one can define the logical operators of the toric

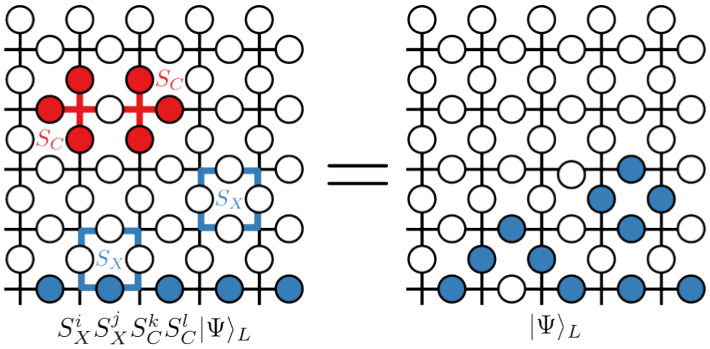


Figure 3.4 – **Trivial and non-trivial** loops on the toric code. Applying stabilizer operators on the lattice can cause the destruction (red) or creation (blue) of trivial loops. Conversely, non-trivial loops can only be deformed by the stabilizer operator. In all cases, applying stabilizers will preserve the same logical state.

code as the product of strings of Pauli operators on the lattice.

$$\overline{X}_1 = \prod_{horizontal} X_i \qquad \overline{Z}_1 = \prod_{horizontal} Z_i \tag{3.30}$$

$$\overline{X}_2 = \prod_{vertical} X_i \qquad \overline{Z}_1 = \prod_{vertical} Z_i \tag{3.31}$$

It is necessary for these logical operators to commute with the stabilizers, which follows directly from the commutation relation already shown before. Applying a stabilizer to a logical operator can be denoted as a deformation of the loop as shown in Fig. 3.4, which returns a loop with the same topological property. So using this behavior it is possible to dissolve trivial loops, while non-trivial loops always remain. This means that the representation of the logical operator is not unique and also that trivial loops do not change our logical code space. The minimal logical operators that can be defined contain L Pauli operators, therefore giving the toric code a code distance of L and defining it as a $[[2L^2, 2, L]]$ code.

Now we can introduce some bit and phase flip errors on the toric code, and as explained in Section 3.3 we apply syndrome measurement based on the stabilizers to detect these errors. Depicting the errors with the associated syndromes on the lattice, one can clearly see that syndromes are formed at the ends of error strings as seen in Fig. 3.5, since stabilizers in the middle of the string always measure two errors at once and therefore do not return a syndrome. This also means, that we cannot measure error strings forming loops. This does not pose a problem for the correction problem, as we have seen before that trivial loop do not change the logical state and therefore do not require correction in the first place. The other possibility would be errors forming non-trivial loops, thereby immediately changing the logical state. Since this would require error strings of length $\geq d$, they are suppressed as the probability that these errors occur decreases with code distance p_{err}^d according to the threshold theorem Eq. (3.5).

As mentioned in the error correction chapter, based on the measured syndrome, one would apply a decoding algorithm that returns a recovery operation to correct the errors and maintains the logical state.

$$RE|\Psi\rangle_L = |\Psi\rangle_L \tag{3.32}$$

Since we do not know how the error string is actually located, in many cases the correction will introduce loops on the torus as seen in Fig. 3.5, where introducing a trivial loop does not change the logical state, whereas if a non-trivial loop is introduced, the logical state changes and the correction has failed. Therefore it is important to find a decoding strategy, which is

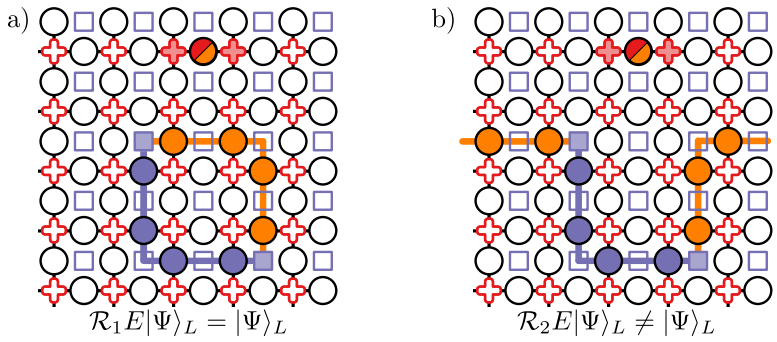


Figure 3.5 – **Recovery operation** on the toric code. Errors on the QEC form error strings (blue) returning eigenvalues of -1 (filled) at their ends during syndrome measurements, while the remaining syndromes return eigenvalues of +1 (empty). To correct the error, a recovery operator (orange) in the form of an Pauli string going from one square to another is applied. If the error and recovery string form a trivial loop (a) the decoding process was successful, and if a non-trivial loop has formed (b) the logical state has changed and the decoding has failed.

able to find the most probable correct recovery operation. Specific examples of decoding will be discussed in the following chapter.

3.4.2 Rotated surface code

While the toric code is a good theoretic model for topological error correcting codes, the periodic boundary conditions are not suitable for practical implementation. So by replacing the periodic boundary condition with open boundaries, the toric code is transformed into the surface code [65, 134]. The stabilizers of the code still remain as in the toric code, carrying over the plaquette and vertex stabilizers. But one has to be careful at the boundary, since some stabilizers will now only consist of 3 physical qubits as shown in Fig. 3.6. Constructing this code with a code distance d, $d^2 + (d-1)^2$ qubits are needed, instead of the $2d^2$ qubits in the toric code. This comes with the limitation that we can only define a single logical qubit on the lattice, making the surface code a $[[d^2 + (d-1)^2, 1, d]]$ code. The loss of one logical qubit is a consequence of the boundary of the code. Focusing on the logical $\overline{X}_1$ operator we can define it as a string running from the top to the bottom border of the lattice, fulfilling the condition in Eq. (3.27). Trying to define a similar logical operator $\overline{X}_2$ running from the left to the right border (like in the toric code) would now introduce errors in the code and would no longer convert a logical state into another $\overline{X}_2|\Psi\rangle \neq |\Psi\rangle$. So the logical operators can only be defined as a string going from one smooth border to another as shown in Fig. 3.6 a).

This version of the surface code was then further developed by considering a rotated version of the code called the *rotated surface code* **RSC**. This version has the advantage of removing qubits at the edges of the surface code, reducing the number of needed qubits, while still having the same code distance d. One even can achieve an higher pseudothreshold under deploarizing noise compared to the surface code [125], because of the data qubits and syndrome measurements. While the RSC still utilizes the same stabilizer operators as the toric code and the surface code, they can be visually represented as a checkerboard on the code as shown in Fig. 3.6 b). Also looking at the border of the code, the smooth border from before remains, while the rough borders have transformed and now consists of 2-qubits stabilizers. This allows us to define the same logical operators as before and to classify the RSC as a $[[d^2, 1, d]]$ quantum code.

Since the RSC still carries the advantages of a topological surface code of only utilizing nearest

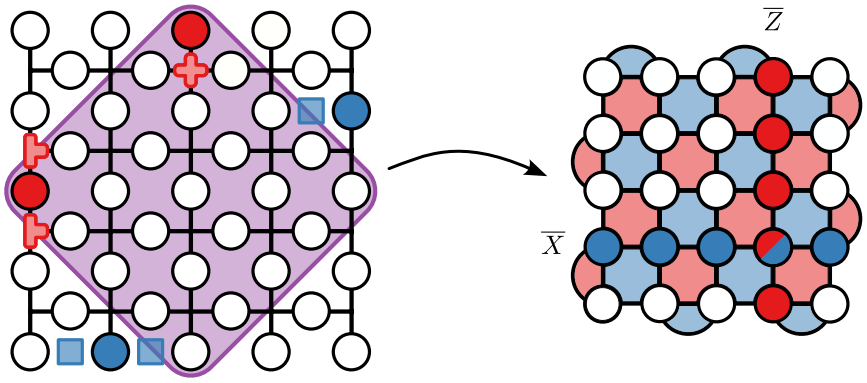


Figure 3.6 – **Surface code** representation on the square lattice. Removing the periodic boundary conditions from the toric code leaves smooth and rough boundaries on the surface code (left). This causes the X stabilizer (blue) to only contain 3 qubits in the parity measurement at the upper and lower border, while still having 4 qubit stabilizers at the left and right border. The same is true for the Z-stabilizer (red), but with the rough border being the left and right. Cutting out a square from this code (violet) and rotating it by 45° returns the RSC (right). The stabilizers are represented as the checkerboard with X stabilizer containing 2 qubits at the upper and lower border and Z stabilizers with 2 qubits at the left and right border.

neighbors, while also having no periodic boundary conditions and only needing a *relatively* small number of physical qubit per logical qubits, this QEC is the most prominent candidate for an actual implementation for fault-tolerant quantum computing and has already been implemented on actual hardware to suppress quantum errors [52].

Chapter 4

Error correction on topological surface codes

With the topological surface code now established, we can encode a logical qubit by utilizing several physical qubits arranged on a lattice. We are also capable of extracting information about potential errors in the form of a syndrome. The subsequent step involves decoding the measured syndrome to perform a recovery operation as indicated in Eq. (3.32). Numerous decoding algorithms have been developed for this purpose, varying in accuracy and speed of the decoding process.

In this chapter, our focus is on comparing the effectiveness of various decoding strategies on surface codes. To achieve this goal, we will begin with a brief discussion of the numerical methods used to extract the decoding accuracy and decoding speed from a chosen decoding algorithm, as well as how to extract the error threshold and pseudothreshold in the following.

Although various decoding strategies exist, we will concentrate on three decoders for surface codes. These being the *minimum-weight perfect matching* (**MWPM**) and *union find* (**UF**) decoders and the *hierarchical decoder*. The MWPM and UF decoders are well-known algorithmic decoders in the field of QEC and, therefore, serve as benchmarks. Conversely, the hierarchical decoder presents a two-step decoding process. The initial phase of error reduction is accomplished by a decoder based on machine learning. This step drastically minimizes the size of the error. Subsequently, an algorithmic decoder completes the error correction process.

In the first step, these decoders are tested under optimized conditions on the toric code under depolarizing noise. The decoding accuracy of various decoders is established by utilizing the error threshold, whereas the decoding performance at small code distances is reported by using the pseudothreshold. Additionally, the decoding speed is explored in terms of time scaling with the code distance. This problem is further extended by including faulty syndrome measurements in the noise model, as well as changing the QECC from the toric code to the RSC for a more realistic scenario.

While the decoding strategies of the algorithmic decoders are well-known from their initial design, the hierarchical decoder carries the 'black box' problem of ANNs [135]. From the results, we can see that the hierarchical decoder has successfully learned a decoding strategy, but we do not know what kind of strategy it has learned. To further investigate this issue, we examine applied corrections and perform case studies focused on correcting specially designed errors. It should be noted that this chapter draws heavily on a previous publication by the author of this thesis [P1].

4.1 Numerical methods

To evaluate the performance of different decoders, it is necessary to be able to compute different observables for a given physical error rate p_{err} and code distance d. The observable can be obtained via Monte Carlo methods [136], enabling us to gather statistics by generating random syndrome-error pairs repeatedly.

Errors can be efficiently generated by using direct sampling [137] based on the investigated noise model. The corresponding syndrome can then be obtained via matrix multiplication of errors with the generator matrix as described in Sec. 3.1. For each error syndrome pair, we apply the decoding algorithm to perform a correction and check whether all logical operators return 0. If this is not the case, the correction has failed, as the logical state has changed compared to before. The original logical state is not relevant for the decoding process, as the syndrome only depends on the errors, as shown in Eq. (3.1). Thus, it only matters if the logical state has changed. After correction, the measured outcomes of the error syndrome pairs, denoted by y_i, are recorded. The estimated value of the observable $\overline{y}$ and the respective standard deviation follows as

$$\overline{y} = \frac{\sum y_i}{N} \qquad \Delta\overline{y} = \frac{1}{N}\sqrt{\sum_{i=1}^{N} (\overline{y} - y_i)^2}. \tag{4.1}$$

Observables of interest include the logical error rate p_{log}, decoding speed, and number and type of applied corrections.

To determine the error threshold, we must first determine the logical error rate for a wide range of physical noise and code distances. For surface code with local error models, the error threshold rate can be mapped to zero-temperature phase transition of a three-dimensional random-plaquette gauge model on classical spins [54]. Therefore, the behavior of the logical error rate corresponds to critical behavior of that spin model [138] and can be described by

$$p_{\text{log}} \propto (p_{\text{err}} - p_{\text{th}})\, d^{1/\nu_0}, \tag{4.2}$$

where ν_0 is the scaling exponent corresponding to the universality class of the model. It is easy to see that if the physical error rate is equal to the threshold error rate, the returning logical error rate is independent from the code distance, indicating a crossing of all measured logical error rates at that point. To account for systematic finite size scaling effects, we can fit our data using some scaling function $\tilde{f}$ and the ansatz

$$p_l = d^{\zeta/\nu_0}\, \tilde{f}\left((p - p_{\text{th}})\, d^{1/\nu_0}\right), \tag{4.3}$$

and thereby extracting the critical exponents. According to the ansatz, plotting $\tilde{f}\left((p - p_{\text{th}})\, d^{1/\nu_0}\right)$ against $p_l d^{-\zeta/\nu_0}$ would result in a data collapse, plotting all data onto a single curve. The fitting and data collapse is done using the *fssa package* [139].

The final step involves identifying the pseudothreshold, which necessitates locating the point where $p_{\text{err}} = p_{\text{log}}$. These points are approximately determined by finding the crossing point between the functions describing the logical error rate and the identity line. The logical error function is approximated as polynomial of first degree running through the data point encompassing the crossing point. The crossing point is then given

$$p_{\text{pseudo}} = \frac{\left(p_{\text{err},1}^2 - p_{\text{err},2}^2\right) - \left(p_{\text{err},1} p_{\text{log},2} - p_{\text{err},2} p_{\text{log},1}\right)}{p_{\text{log},1} - p_{\text{log},2} - p_{\text{err},1} + p_{\text{err},}}, \tag{4.4}$$

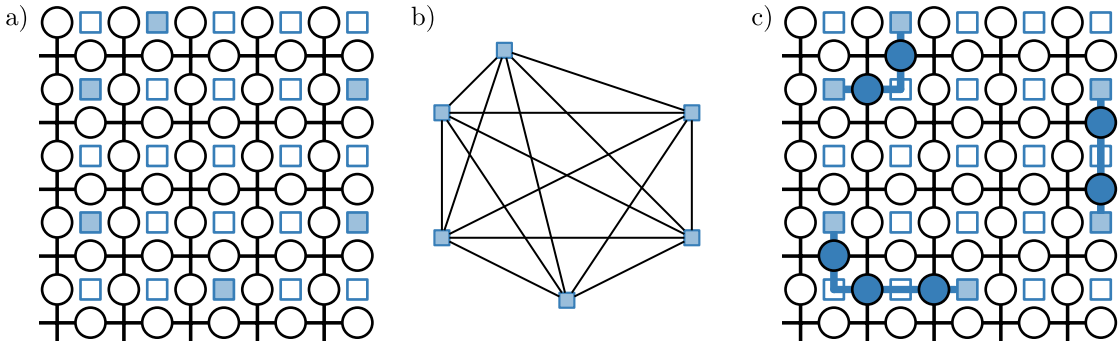


Figure 4.1 – **Decoding process of the MWPM** on the toric code. In (a) a toric code of size $d = 5$ with measured X syndromes (filled squares) is visualized. The MWPM extracts the syndromes to construct a fully connected network between them, as shown in (b), where each edge has a weight corresponding to the Hamming distance between the syndromes. The recovery operation is achieved by selecting the edges with the smallest total sum of weights as shown in (c).

where $p_{\text{log,i}}$ are the logical error rates of the points left and right of the crossing point and $p_{\text{err,i}}$ are their respective error rates.

4.2 Minimum-weight perfect matching

One method of decoding is the *minimum-weights* decoding, which seeks to identify the smallest error associated with the measured syndrome. For general quantum codes, this minimum decoding has no known solution, since it is known to be NP-complete [140–142]. For topological codes, the Edmond's blossom algorithm [143] can efficiently calculate the minimum matching. This algorithm constructs a fully connected graph structure from the measured syndromes and identifies a perfect matching of the syndromes with the minimum-weight, which is illustrated in Fig. 4.1. Therefore, the decoder is referred to as the *minimum-weight perfect matching* (**MWPM**) decoder.

A straight forward implementation of the blossom algorithm can find a perfect matching in $O(n^2m)$ time [144], where n is the number of nodes in the graph and m the number of edges, in case of a complete graph the number of edges would be $m = \frac{n(n-1)}{2}$ allowing us to write the scaling as $O(n^4)$. Since the node of the graph are formed by the measured syndromes and the number of syndromes in a surface code is generally $\propto d^2$, we can write the scaling depending on the code distance as $O(d^8)$.

Thanks to improved implementation of the Blossom algorithm, could be gradually reduced.

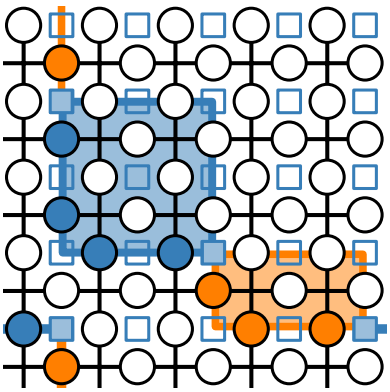


Figure 4.2 –
Maximum probability vs minimum-weight. 2D toric code example with 4 measured syndromes (filled squares). Two possible recovery operations are shown with different homology classes. Both recovery operations are build from two recovery strings with the total length $l = 5$. One of the recovery operations consists of the recovery strings $l = 3$ and $l = 2$ (orange) and the other operation of $l = 4$ and $l = 1$ (blue). Both operations are degenerate and are visualized with an example chain (filled circle). Other possible recovery chains can be freely chosen from inside the highlighted areas (semitransparent rectangles).

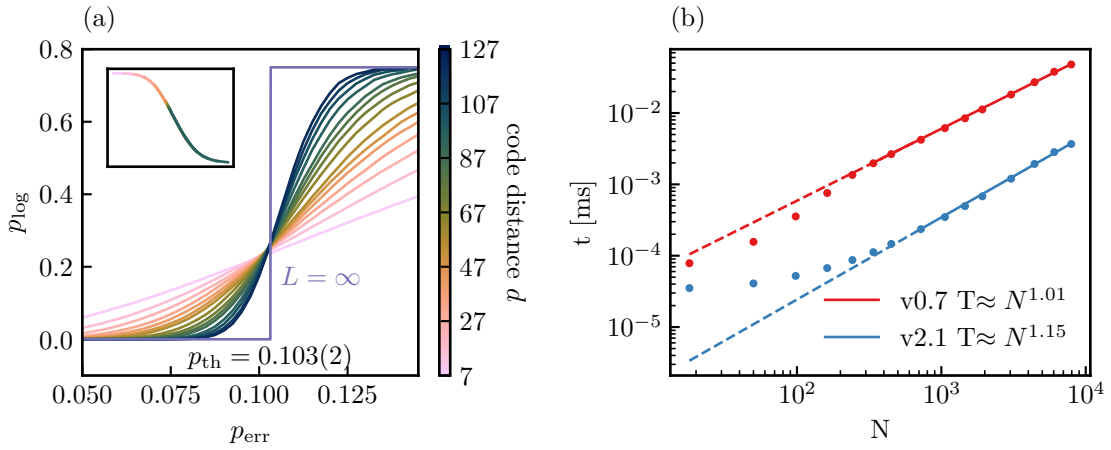


Figure 4.3 – **Performance of the MWPM decoder** under bit flip noise. In (a) the plot shows the logical error rate depending on the physical noise for varying code distances. The blue curve shows the logical error rate in the thermodynamic limit. The insert shows the data collapse of the threshold plot data. In (b) the plot shows the decoding time for $p_{\text{err}} = 0.1$ depending on the number of qubits. The decoding time is shown for PyMatching version 0.7 and 2.1.

In this thesis we will utilize the MWPM implementation called *PyMatching*, examining the two distinct versions, namely v0.7 [145] and v2.1 [146]. The former was employed for data generation during the initial stages of this dissertation, while the latter promised a significant reduction in decoding time while preserving the same error threshold.

We determine the error threshold of this decoder by generating 10^5 error-syndrome pairs under bitflip noise for $p_{\text{err}} \in [0.05, 0.15]$ and $d \in [7, 127]$ as shown in Fig. 4.3 (a). Using the finite size ansatz 4.2 this returns $p_{\text{th}} = 0.1032$ for the PyMatching v.2.1, which is consistent with the literature threshold value for MWPM [147]. Doing the same calculation for PyMatching v0.7 as seen in App. B, Fig. B.1, returns the same error threshold of $p_{\text{th}} = 0.1032$. This value is close to the optimal threshold $p_{\text{opt}} = 0.1094$ [148], which is determined by the critical point of the two-dimensional Random Bond Ising Model on the Nishimori Line. The difference between the value comes from the fact, that the MWPM does not account for error degeneracy. In some rare cases, two different homology classes of errors producing the same syndrome have the same minimum error length, but are not equally likely to occur. Therefore, the MWPM might not chose the most likely error [149]. An example of this can be seen in Fig. 4.2, where two different corrections for a syndrome are presented, one made out of a $l = 2$ and $l = 3$ chain and the other one made out of a $l = 1$ and $l = 4$ chain. For the MWPM both options have the same total weight of 5, so it would choose one of them randomly. Looking closer at the degeneracy of the recovery operations, it shows that the 3-2 length recovery operation has a 3-folded degeneracy, while the 4-1 length recovery operation has a 6 folded degeneracy. Since there are more configurations available for the correction, the 4-1 correction has a higher probability to be the right one.

After specifying the error thresholds, we will compare the decoding speed. Fig. 4.3 (b) depicts a decoding speed comparison between PyMatching v0.7 and v2.1. Both curves demonstrate a polynomial scaling of time, scaling almost linear with the number of qubits. Specifically, the time scaling is $\mathcal{O}^{1.01}$ and $\mathcal{O}^{1.15}$ for version 0.7 and 2.1 of PyMatching, respectively. Additionally, PyMatching v2.1 offers a notable decoding speed increase, surpassing v0.7 by a factor of ≈ 13 at $d = 127$ and ≈ 2 at $d = 7$. These time benchmarks were generated using the Intel Xeon Platinum 8168 CPU with two sets of 24 cores, running at 2.7 GHz.

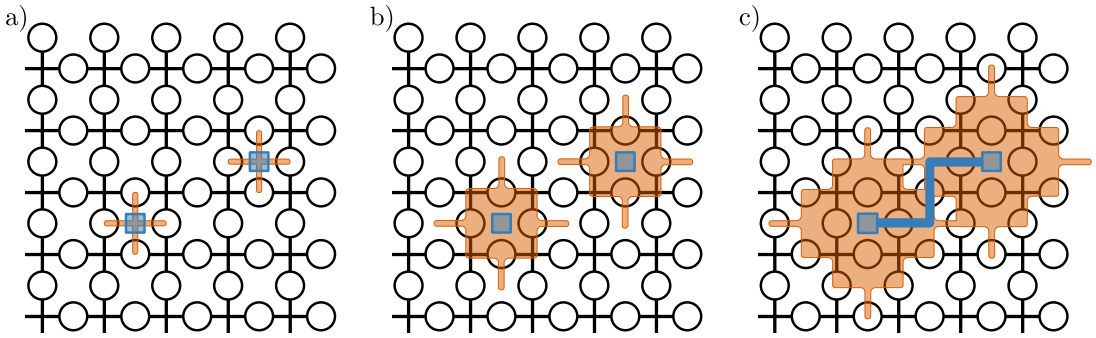


Figure 4.4 – **Union find decoding process** visualized on the toric code. It starts with the initialization stage (a), where every single syndrome (filled blue squares) is initialized as a cluster (orange). This is followed by the growth stage, where every cluster grows by half an edge (b). This is repeated until all clusters are merged into clusters with even numbers of syndromes. Then a peeling decoder is applied to the clusters (c), removing all edges until only a recovery operation (blue edges) remains.

4.3 Union find

Another decoder used in this dissertation is the *union find decoder* (**UF**) [150], which utilizes the key insights from Union-Find data-structure algorithm [151, 152] to design a decoder that has near-linear running time in the worst case given by $O(n\alpha(n))$. Here α is the inverse of the so called *Ackermann function* [152]. If the number of qubits used is smaller then the number of atoms in the universe, then $\alpha(n) \leq 3$ and can be considered as a constant [150].

The general idea of the UF can be described as a cluster growth algorithm. In the initial state every syndrome will be marked as the root of its own cluster. The next step is the growth step, where every cluster will grow over time. If the connection from one syndrome to the next is considered as a single edge, then the connection between syndrome and a qubit will be a half edge. During the growth step every cluster with an odd number of syndrome inside will expand by a half step. If two different clusters become connected, they will merge into a single cluster and therefore stop growing. If no more even clusters remain, all full grown edges are collected and passed to the peeling decoder [153].

The peeling decoder will start by choosing an arbitrary spanning tree of the cluster, which contains no cycles and spans all vertices. This tree is then being systemically reduced, starting at a leaf edge (u, v), according to the following rule. Erase the edge from the cluster and if $u = 1$ then set the edge value to 1 and flip the value of v. In case the value of $u = 0$, then just the edge value is set to 0.

We have implemented a weighted version of the Union Find decoder [S1], which always grows the cluster with the smallest boundary and compare it to the PyMatching Decoder. Performing the same threshold analysis as for the PyMatching decoder reveals an error threshold of $p_{\text{th}} = 0.098$ as shown in Fig. 4.5 (a) matching the literature [150], which is about 5% below the PyMatching threshold. Comparing the decoding speed in Fig. 4.5 (b), the UF decoder returns an almost linear scaling with $\mathcal{O}(d^2)$ which is almost the same scaling as the PyMatching. Comparing the decoding speed directly, UF beats the MWPM by a factor of ≈ 1.4 for $d = 3$, while being slower at $d = 127$ by a factor of ≈ 1.6.

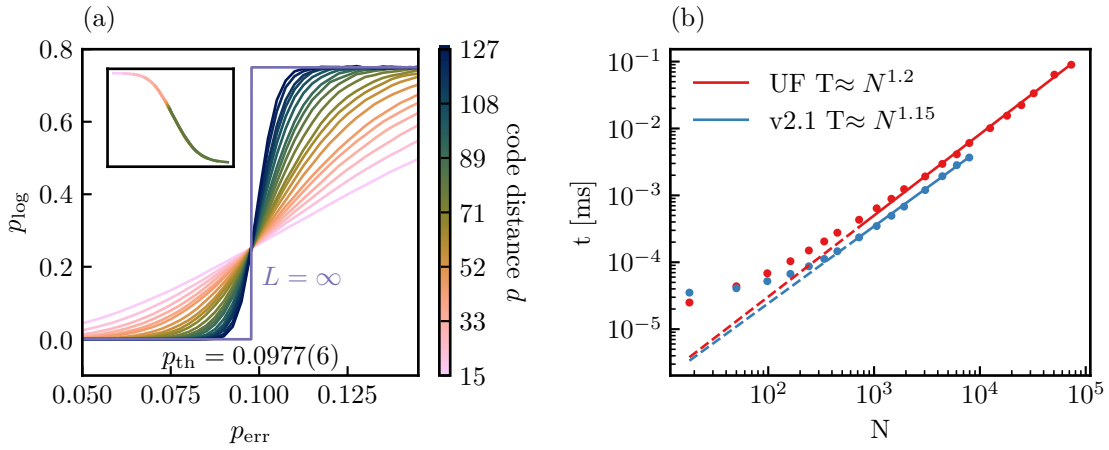


Figure 4.5 – **Performance of the UF decoder** under bit flip noise. In (a) the plot shows the logical error rate depending on the physical noise for varying code distances. The blue curve shows the logical error rate in the thermodynamic limit. The insert shows the data collapse of the threshold data. In (b) the plot shows the decoding time for $p_{\text{err}} = 0.1$ is shown depending on the numbers of qubits. The decoding time is shown for the UF decoder (red) and the PyMatching v2.1 (blue).

4.4 Hierarchical decoder

The basic idea of a neural decoder is using the syndromes as an input and returning an appropriate correcting scheme. In the case of the toric code, the input would therefore be the two-dimensional image of the syndrome measurement, and as output one would expect similar the two-dimensional map of the correction. Because of this structure, this problem can be viewed as an image classification problem, a common field for machine learning application.

The idea of a neural decoder is well established and uses different approaches and network architecture. It has by shown that neural decoder are able to increase the error threshold compared to classical decoders, especially when decoding on correlated noise. However, a common disadvantage is that these neural decoders only work on small code distances [79–81], which is easily understood as a training problem.

Considering a neural decoder on a toric code with bit flip noise that takes the whole syndrome as input and gives a correction for every single qubit, then it has to learn the correct pairing between the syndrome space of 2^d and the correction space of 2^{2d}. Even considering that the network realistically only needs to learn small parts of it, the increasing difficulty leads to the need for longer training times and larger networks, until the training completely fails due to training problems like the vanishing gradient described in Sec. 2.1.1.

This then automatically limits the application range of the decoders, since it is desirable to increase the code distance as much as possible in order to achieve fault-tolerant quantum computing. Because of this we designed a neural decoder with the goal of full scalability in mind. This means a machine learning based decoder, which can be trained once on a small code distance and then be applied on arbitrary large systems.

4.4.1 Implementing scalability

First, the difficulties in designing a fully scalable neural decoder are considered. At the core of the decoder is a neural network, a structure that cannot be scaled on its own without having to retrain the network. As soon as the input or the output changes in size, a fully connected

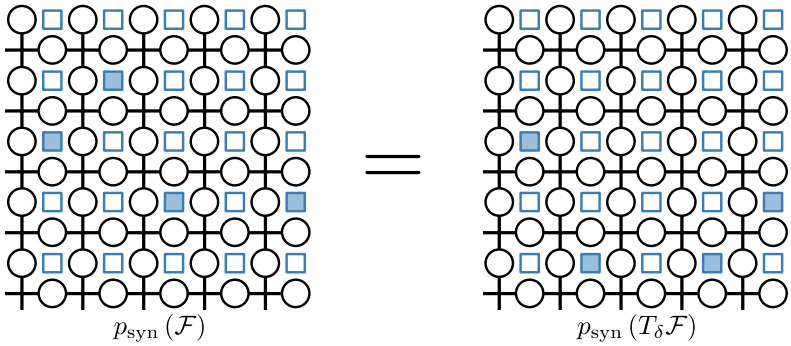


Figure 4.6 – **Translational property of the toric code**, showcasing that two different syndromes have a certain probability of occurrence, if the error generating the syndrome only differs by some translation operation T_δ.

network has to be retrained, which is associated with the problem of exponentially increasing syndrome and correction space.

As seen in Sec. 2.1.3, convolutional layers themselves are translation invariant regarding the input and can therefore be applied to any input size, where the output in form of a feature also scales with the dimension of the input. This makes them a prime candidate to realize a scalable decoder. The problem is that while the convolutional layers themselves are scalable, the CNN's normally are not. As soon as the convolutional layer is connected to a fully connected layer, the network loses the translation invariance.

So in order to avoid that, we first want to exploit the translation invariance of the toric code. This means that error strings $\mathcal{F}$ which only differ by a translation of T have the same occurrence probability if the respective syndromes differ by the same translation T

$$p_S(\mathcal{F}) = p_{T_\delta S'}\left(T_\delta \mathcal{F}'\right). \tag{4.5}$$

So by exploiting the translation invariance it is possible to reduce the syndrome and correction space and therefore be able to train for higher code distances [154]. But this is not enough to achieve a scalable decoder, since the input and output still scale with the code distance.

Because of this we want to set our training goal for the neural decoder to learn the marginal probability of a single qubit, instead the whole correction string

$$p_{\mathcal{F}}(X_j) = \sum_{i \in E(X_j)} p(E_i). \tag{4.6}$$

The probability that a specif error occurs on a selected qubit is given by the sum of all probabilities of the error for all suitable error strings. This means that we reduce the difficulty of learning the entire correction string space of 4^{2L^2} to learning a suitable correction of a single qubit with the 4 possible corrections of I, X, Z, Y. Once we have learned the probability distribution for a single qubit, we can transfer this learned knowledge thanks to the translation invariance of the toric code.

Now that the output size is fixed in space, we have to take care of the input space of the decoder. Since the goal is to operate quantum computing devices far away from the error threshold, we assume that long range correlation between syndromes are suppressed. This allows us to limit the input of the decoder to a 2-dimensional mask around the selected qubit. The mask can then be shifted around to collected the correction probabilities of all qubits in the system, making the network scalable. For the neural network at the core of the decoder, we then simply use a fully connected ANN, as shown in Fig. 4.7

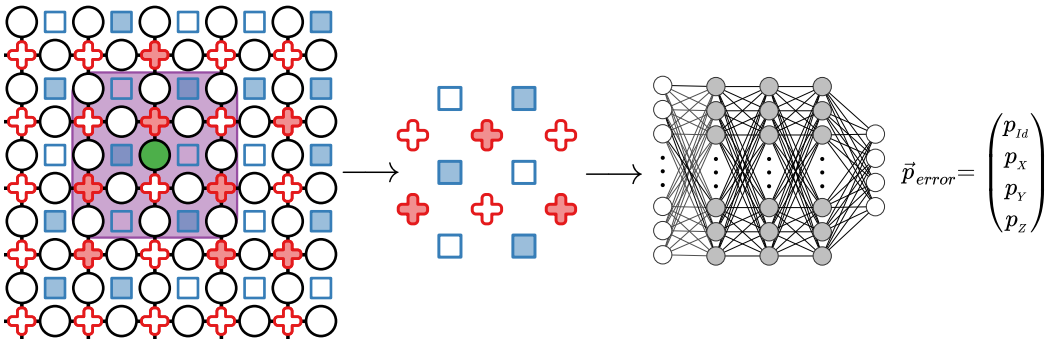


Figure 4.7 – **Construction of the input data and neural network,** demonstrated on a toric code with a random syndrome. Measured X/Z syndromes (filled blue/red) are assigned the values +1/-1 and not measured syndromes (empty squares/crosses) are assigned 0. Selecting a single qubit adjacent to a measured syndrome, all syndromes in the vicinity (violet square) are used as input. Passing the input to an ANN returns the probability values for all 4 possible correction values for the selected qubit. Taken and modified from [P2].

This structure is equivalent to using multiple convolutional hidden layers, where the kernel of the first layer is given by the mask and all further kernels have size of 1x1. The final output corresponds to 4 feature map containing the correction probabilities of every single qubit.

A detail to consider is that the toric code has a unit cell with a basis of two. So while the translation properties of the toric code allows us to easily shift between qubits of the same basis, this does not apply to qubit of different basis indices. This can be easily understood by looking at the syndromes of an Y error on the qubits of different basis indices as shown in Fig. 4.8. When using these syndromes around these qubits, the order of the X and Z errors would be switched. This problem is solved by assigning the X & Z syndromes values of +1&-1 respectively. With this our decoder should be able to distinct the two different cases and learn the corresponding output.

4.4.2 Decoding process

We now have an ANN that we can train on an arbitrary code distance and then deploy it on any other code distance, as long as the code distance is larger then the input area.

When deploying the ANN as a decoder the marginal probability of the correction of every single qubit is evaluated and collected. Afterwards, a correction is applied to the qubits based on the highest correction probability. This is done in a one-shot fashion, meaning that all the

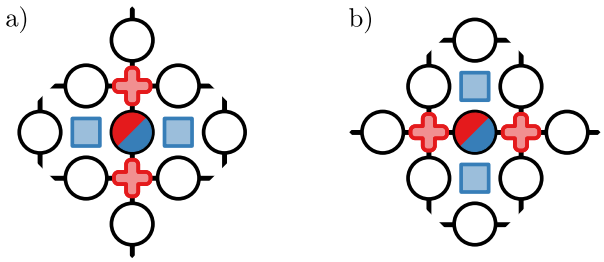


Figure 4.8 – **The basis of the toric code**. Placing an Y error (blue/red) on the first qubit of the basis (a) results in measured X syndromes (blue) left and right from the qubit and measured Z syndromes (red) above and below the qubit. Placing an Y error on the second qubit (b) of the basis results in a switch of the measured syndromes with the X syndromes being above and below and the Z syndromes left and right from the qubit.

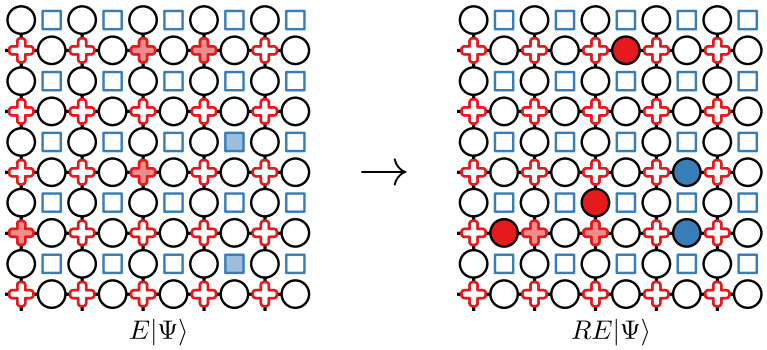


Figure 4.9 – **Applied recovery operation from the neural decoder**. On the left side syndromes on the toric code are shown, containing X (blue) and Z (red) syndromes. Using the neural decoder part of the hierarchical decoder returns a recovery operation which is applied in one-shot to the QEC. The applied X (blue) and Z (red) corrections are presented on the qubits (filled circles). Two error chains of $l = 1$ and $l = 2$ are corrected completely leaving no syndromes behind, while the error chain with $l = 3$ has been reduced to a chain of $l = 1$, resulting in a new syndrome.

correction of every single qubit is applied at the same time.

This decoding process does come with two disadvantages. First of all, since the ANN only learns the marginal probabilities of the corrections, it has problems with learning syndromes with multiple shortest recovery paths. When evaluating a syndrome that has two shortest paths of recovery as shown, the marginal probability would split into $p_I = 0.5$ and $p_X = 0.5$. This means that the output of the ANN depends on random symmetry breaking in the training and would favor and slightly favors one probability over the other. This means that it could learn to always correct the lower qubit or, since the probability are independent from each other, to correct all the qubits and thereby failing to correct the error.

This problem becomes more apparent when looking at syndromes with 3 or more shortest recovery paths. Considering only the two measured syndromes, the degeneracy can be calculated as $\binom{l}{l_v}$, where l is the length of the correction chain and l_v is the number of vertical links inside the correction chain. Then the most probable correction of the qubits not directly adjacent to the syndrome is the identity operation. This means that the ANN should always learn not to perform any correction on these qubits and therefore is not able to correct these errors.

Because of this, it is not necessary to evaluate every single qubit in system, instead we can focus on only trying to correct qubits, which have an inherently high probability of carrying an error, these being qubits directly adjacent to syndromes. This allows us to speed up the decoding process since we only need to evaluate a fraction of the actual qubits, but also limits us to only correcting error strings of $l = 2$.

This is now part of the second problem, namely that after applying the recovery operation given by the neural decoder, it is not necessarily guaranteed that a logical code word is recovered as shown in Eq. (3.32). In order to correct error strings of length three or more, we could run the neural decoder multiple times, trying to shorten the error in every iteration. But even then, the decoder could have failed to learn to correct syndrome with two equally probable recovery strings. These syndromes persist through every application iteration of the decoder. So instead we will apply a classical decoding algorithm, like the MWPM or UF decoder, which guarantees that a logical code word is returned. No special choice is needed for the second decoder, since the updated syndrome S' follows directly from the applied first correction

$$S' = P\mathcal{R} + S, \tag{4.7}$$

with P being the parity matrix and S the initial syndrome. Any decoder which can be used to

training parameters	
batch size	512
epochs	10^6
learning rate	0.001
optimizer	ADAM
loss function	categorical cross-entropy
training code distance d_{train}	7

Table 4.1 – **Hyperparameters** used to train the hierarchical decoder on the toric code. An epoch corresponds to the number of generated training batches for the training.

decode the toric code is suitable as a secondary decoder. Since we apply two different decoder one after the other, we call it *hierarchical decoding* and refer to the first part as a *neural decoder*.

Lazy decoder We will additionally introduce another primary decoder for the hierarchical decoder, based on the Lazy Decoder [155]. This lazy decoder will sweep over the lattice and apply all possible corrections of $l = 1$ and leave all other syndromes alone. Therefore, a secondary decoder is still needed. The lazy decoder will decode the X and Z errors separately, so it will not take Y errors into account. The idea of this decoder is to correct these short range errors as fast as possible, reducing the work of the secondary decoder. The increase of the decoding speed comes with the trade-off an decrease in the logical accuracy.

4.4.3 Training of the decoder

The training of the decoder is done in a supervised learning fashion following Sec. 2.2, utilizing the ADAM optimizer. Because we have reduced the output of the ANN to the probabilities of the different possible errors, which is a classification problem, we use the categorical cross entropy loss function 2.15.

Because the generation of error syndrome pairs is computational cheap, we can train the network on newly generated trainingsdata for every training step instead of creating one fixed training batch. This reduces the chance of overfitting significantly, as it then only depends on the size of the mini-batches. The exact parameters of the training are listed in Tab. 4.1. These parameters were used to train various decoders under the same conditions and have been determined via grid search.

It is important to note, that the training batches have been carefully constructed to contain a significant amount of non-trivial correction labels. This is done because using all qubits to generate the input would result in a training batch with an average of $p_{\text{err}} N_{\text{batchsize}}$ non-trivial correction labels. This can lead to underfitting in the training process, where the network just learns to always apply the trivial correction.

Therefore, we choose to only include qubits in the training data that are directly adjacent to measured syndromes. As each syndrome is a product of four qubits, this guarantees us that at least on qubit carries an error. Combining this with only taking into account the qubits connected to one randomly chosen syndrome per generated error-syndrome pair, we achieve a non-trivial recovery operation rate of $> 25\%$ in the batches.

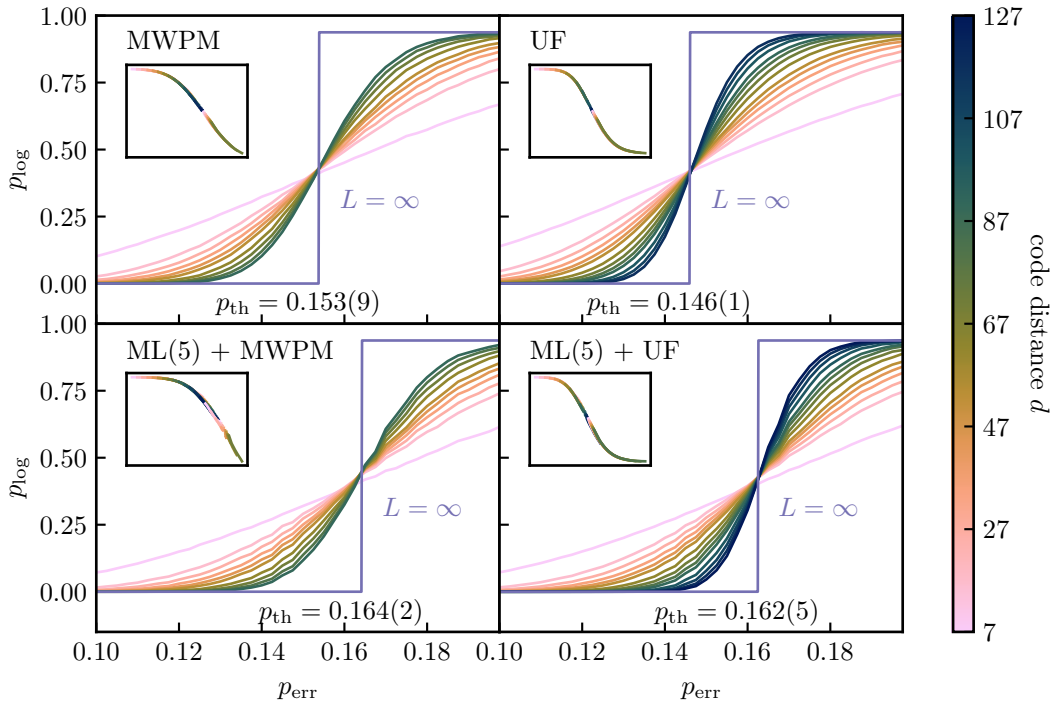


Figure 4.10 – **Benchmark measurement** for different controllers. In each panel the logical accuracy is plotted against the error probability of the depolarizing noise. The curves are colorcoded for the code distance with the blue curve showing the thermodynamic limit. The inserts show the data collapse of the threshold plot data. The 4 different decoders showcased are the MWPM (a), UF (b) and their hierarchical decoder counterparts ML(5) + MWPM (c) and ML(5) + UF (d).

4.5 Depolarizing noise toric code

After defining the decoders of interest, we want to test their performance under depolarizing noise 3.22. For this we first compare the performance defining observables of the decoders, namely the error threshold, pseudothreshold and decoding speed. Then we will tackle the *black box* problem of the neural networks and investigate the decision making of the hierarchical decoder, trying to understand how it chooses the actual corrections.

4.5.1 Benchmarking of the decoding accuracy

We are going to determine the depolarizing error threshold for the MWPM and UF thresholds as well as different combinations of the hierarchical decoder. For this we train two different neural decoders, the first one has am input mask of size $l = 5$ and the second one of $l = 7$. Because of this the neural decoders will be denoted as ML(5) and ML(7), respectively. The ML(5) consists of 3 hidden layers, with 128 hidden nodes each, resulting in 40,068 tuneable parameters. The ML(7) consists of 5 hidden layers with 512 hidden nodes each, resulting in more then $> 10^6$ free parameters. As the decoding capabilities should increase with the size of the ANN, the ML(7) aims to increase the threshold as far as possible, while the ML(5) tries to find a trade off between decoding accuracy and speed. The exact structure of the networks is listed in Tab. 4.2. Additionally, we will also benchmark the Lazy + UF hierarchical decoder.

The performance of the decoder is tested by calculating the logical error rate using Monte Carlo methods described in Eq. (4.1) over 10^5 error syndrome pairs for every data point and

$\ell = 5$ network parameters (optimal wall-clock time)	
hidden layers	3
hidden nodes per layer	128
total number free parameter	40 068
activation functions hidden layer	Relu
activation functions output layer	Softmax
$\ell = 7$ network parameters (optimal error threshold)	
hidden layers	5
hidden nodes per layer	512
total number free parameter	1 103 364
activation functions hidden layer	Relu
activation functions output layer	Softmax

Table 4.2 – **Depolarizing noise networks**. The upper panel lists the detailed network architecture for optimizing wall-clock time based on a 5×5 input mask. The lower panel shows the structure for an optimized error threshold using a 7×7 mask. The table was taken and modified from [P2].

in a range from $p_{err} \in [0.1, 0.2]$.

To give a reference for the thresholds, the theoretical best possible threshold one can achieve for depolarizing noise on the toric code was inferred to be at $p_{opt} = 0.189(3)$ by mapping the decoding problem to a classical disordered eight-vertex model [156].

In Fig. 4.10 the thresholds of the decoders are showcased. Firstly, it is notable that for every single decoder the logical error rate $p_{\log}$ goes towards 0 for $p_{err} \to 0$ and the other way around that $p_{log} = \frac{15}{16}$ for $p_{err} \to 0.5$. The limit of the high noise case is expected as a result of the numbers of logical operators on the toric code. For depolarizing noise 4 different logical operators X_1, X_2, Z_1, Z_2 are used to determine the logical state. In the limit of high noise, the initial logical state is immediately destroyed by the noise and any attempt at correction results in the toric code being shifted into a random code space. This chance of choosing one of the wrong states is given by the number of different combinations of the logical operators, so $p_{log} = 1 - \frac{1}{2^4} = \frac{15}{16}$.

First, comparing the MWPM with the Union Find decoder, one can see that the MWPM decoder with a threshold of $p_t h = 0.1539$ has a threshold about $\approx 5\%$ higher than the UF with a threshold of $p_{th} = 0.1461$. This result is to be expected, since the MWPM takes long range connections between the syndromes into account compared to the UF. Combining MWPM and UF with the ML(5) neural decoder, in each case the threshold increases by up to 11% to $p_{th} = 0.1625$ of the UF and $p_{th} = 0.1642$ for the MWPM. Using the ML(7) as primary decoder further increases the thresholds to $p_{th} = 0.1670$ and $p_{th} = 0.1671$, as seen in Fig. B.2. Looking at the hierarchical decoder variation, the Lazy + UF decoder, in Fig. B.3 a threshold of $p_{th} = 0.1319$ is returned, the lowest threshold yet.

Comparing the depolarizing noise thresholds of UF and MWPM with the previously determined bit flip noise thresholds, one can see a general increase. This is a consequence of the definition of the depolarizing noise and the fact that UF and MWPM are correcting the X and Z syndromes separately. Looking at only one error type, e.g. X errors, the probability of occurrence is $p_X = \frac{2}{3} p_{\text{err}}$ under depolarizing noise. Because of this bit flip error threshold can be derived by multiplying the depolarizing threshold by a factor of $\frac{2}{3}$. This results in

$$\frac{2}{3} p^{depol}_{\text{th,MWPM}} = 0.102(6) \qquad \frac{2}{3} p^{depol}_{\text{th,UF}} = 0.097(4), \tag{4.8}$$

which matches previously determined error thresholds up to an error.

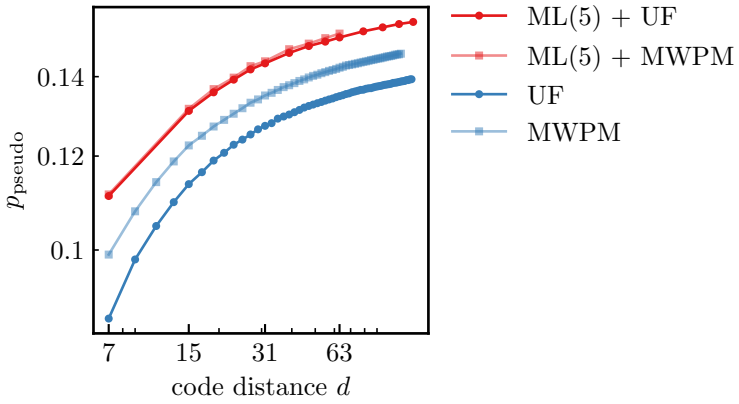


Figure 4.11 – **Comparison of pseudothresholds** on the 2D toric code. The pseudothreshold is plotted against the code distance for the MWPM (light blue) and UF (blue) decoder, as well as the hierarchical decoder counterparts ML(5) + MWPM (light red) and ML(5) + UF (red).

The other measure that we want to compare is the pseudothreshold. Fig. 4.11 shows the pseudothreshold plots for all four decoders. By comparing these curves it is now easier to see that, even in small code distances regime, the hierarchical decoder has an advantage over pure algorithmic decoders. At $d = 15$ the pseudothreshold lies about 8% above the MWPM pseudothreshold and 15% of the UF. This shows that using the hierarchical decoder not only increases the threshold, but also the logical accuracy at small error rates.

4.5.2 Time scaling of decoding

Now we have established that the hierarchical decoder increases the decoding accuracy, but the question about the speed of the decoding process is still open. In Fig. 4.12 we have plotted the average time needed to decode a single syndrome measurement against the code distance of the toric code. The average time was determined using 10^6 samples per data point, benchmarked on:

- Intel Xeon CPU E5-2699A v4 @ 2.40 GHz Cpus
- SXM2 GPU: Nvidia Tesla V100

This was done for the 4 error rates $p_{err} \in [0.01, 0.05, 0.1, 0.1461]$ to get snapshots of the performance for the different error regime. In order to compare the decoding speeds, we plot the average decoding time of the hierarchical decoder against the UF decoder, since it has an almost linear time scaling and a comparatively high decoding threshold.

Looking at the benchmark curves of the UF the decoding time has a seemingly linear scaling as expected even when testing up to a code distance of $d = 255$ measured at the error threshold. In comparison, the hierarchical decoder has seemingly different scaling behavior depending on the code distance and error rate. For p_{th}^{UF} and $d > 63$ it seems that the decoding time of the hierarchical decoder has the same scaling as the UF, as both curves evolve in parallel. It also manages to achieve an improvement in decoding time by $\approx 70\%$. The same behavior can be seen for $p_{err} = 0.01$ and $d > 127$ with an improvement of decoding time of $\approx 40\%$ for $d = 255$.

Now looking at the other side of these mentioned limits, the hierarchical decoder converges against a constant decoding time. Investigating the cause of the behavior has shown, that for small code distances and error rates the decoding time is dominated by the time needed to transfer and initialize the data on the GPU. We have determined the minimum time needed for this process by using empty arrays as input and a minimal neural network, which only consists of the input and output layer. The determined time was then denominated as *kernel time*.

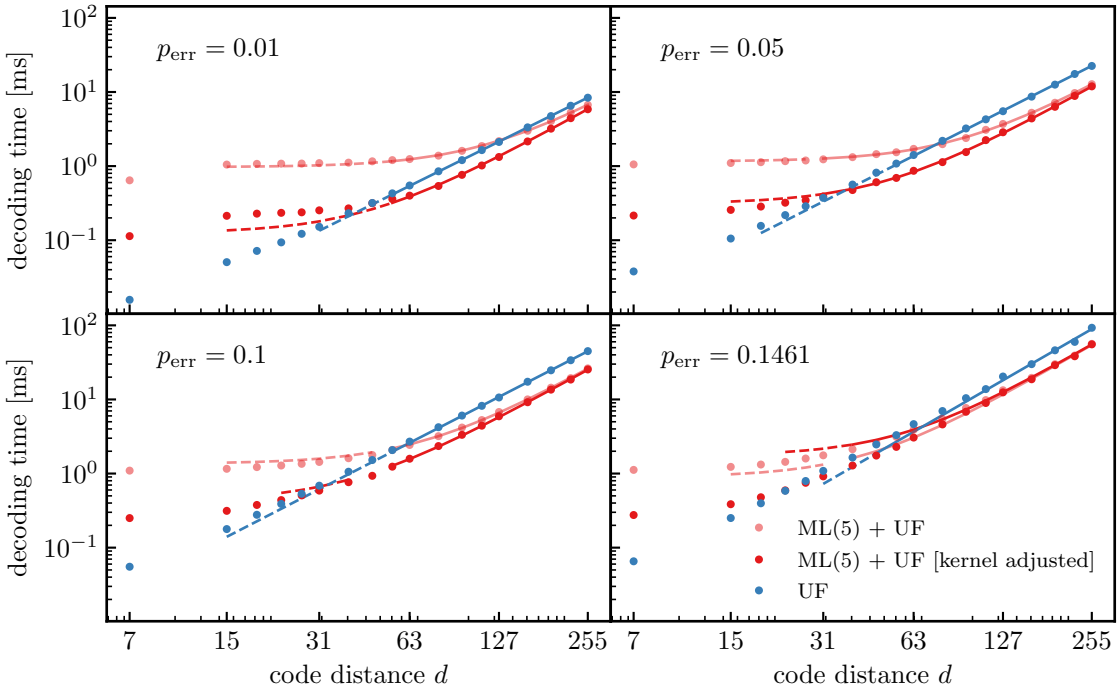


Figure 4.12 – **Comparison of decoding times** of the hierarchical decoder (blue) and UF decoder (orange) for various physical error rates. Shown is the average decoding time measured as wall-clock time. The *kernel adjusted* time represents the ML(5) + UF with a subtracted constant to compensate kernel launch times. The figure was taken and modified from [P2].

To get a clearer picture of the scaling, we subtract the kernel time[1] from the measured decoding time, and assume that it would be considered in an actual hardware implementation. The resulting curves are deemed as *kernel adjusted* and show a more clearer scaling behavior. The most notable change occurred in $p_{err} = 0.01$, where the linear scaling behavior is now observable for $d > 63$ instead of $d > 127$. For the kernel adjusted time, the hierarchical decoder achieves a speed improvement for $d \approx 31$ at the UF threshold and $d \approx 45$ for $p_{err} = 0.01$.

Similar benchmarks were performed for different network structures of the neural decoder as well as different combinations of decoders. The results are summarized in the Table 4.3, where have listed two different neural network structures, which are indicated as ML(5) and ML(7). This notation corresponds to an input size of $l = 5$ and $l = 7$, respectively. The ML(7) was constructed with the purpose of achieving the highest possible threshold, while the ML(5) is designed with the decoding speed vs decoding accuracy trade off in mind. These networks have been combined together with UF and two different version of the PyMatching MWPM implementation. Looking at the determined thresholds, the highest values are returned by the ML(7) hierarchical decoder with either UF or MWPM as secondary decoder. Both thresholds are equal up to the error margin. Now comparing the measured average decoding times, it is notable that using the PyMatching v0.7 as secondary decoder, results in time one order of magnitude higher then using other secondary decoders. Even when adding the neural decoder beforehand, and thereby cutting the decoding time in half for $p_{\text{err}=0.01}$, it cannot compete to just

[1] When subtracting the kernel time, we have an additional factor $(1 - p_{err})^{2d^2}$ that takes into account the probability that no error has occurred. If this is the case, then no data is transferred to the GPU in the first place.

depolarizing noise ($d = 255$)

algorithm	p_{th}	$t_{p=0.01}$	$t_{p=0.05}$	$t_{p=0.1}$	$t_{p=0.1461}$
ML(7) + UF	**0.167(0)**	10.5	25.1	43.4	78.6
ML(5) + UF	0.162(5)	6.7	**12.8**	**26.2**	**56.2**
Lazy + UF	0.131(9)	6.9	20.7	51.1	—
UF	0.146(1)	8.4	22.5	44.9	92.8
ML(5) + PyMatching v2.1	0.163(5)	7.5	20.1	33.9	56.8
PyMatching v2.1	0.154(3)	**5.6**	22.4	49.0	154.2
ML(7) + PyMatching v0.7	0.167(1)	~ 210	~ 530	~ 650	~ 980
ML(5) + PyMatching v0.7	0.163(8)	~ 270	~ 510	~ 650	~ 970
PyMatching v0.7	0.154(2)	~ 560	~ 840	~ 1100	~ 1300

Table 4.3 – **Decoding performance overview**. Comparing the error threshold and average decoding time for various combinations of hierarchical decoder for depolarizing noise. The decoding time is reported in milliseconds and was measured for $d = 255$ and various error rates, where the highest investigated error rate was chosen to be the threshold of the UF decoder $p_{\text{th}}^{\text{UF}} = 0.1461$.

using the UF decoder. Comparing the UF with PyMatching v2.1, it seems that PyMatching allows for slightly faster decoding in low error rates, where as UF has the advantage at higher error rates.

Now looking at the hierarchical decoder, the lowest decoding time was achieved by the combination of ML(5) + UF at all error rates, with the exception of $p_{\text{err}} = 0.01$, where PyMatching v2.1 achieved the lowest time. As seen before ML(5) + UF outperformed the UF, but is also outperformed the combination of a lazy decoder + UF. In this combination lazy decoder quickly sweeps over the lattice and corrects all syndromes corresponding to an error chain with the length of $l = 1$. This combination allows for faster decoding times for the price of lowering the decoding accuracy. Comparing to ML(5) + PyMatching v2.1, both hierarchical decoders achieve similar decoding times, so one might consider choosing the PyMatching v2.1 version for the slight increase in decoding accuracy.

Now we have established that the hierarchical decoder can decrease the decoding time, we want to determine the actual time scaling of the decoder. In Fig. 4.13 (a) the hierarchical decoder demonstrates a seemingly linear scaling in decoding time, but to understand this better, we also take a look at the time scaling of just the neural decoder. In Fig. 4.13 (b) the time measured shows a clear linear scaling. This is to be expected, since by construction the input and the neural network stay the same size independent of the code distance. Therefore the time needed to obtain the correction of a single input remains constant. Therefore only the number of input given to the neural decoder increases with error rate and code distance according to $\mathcal{O}(p_{\text{err}}N)$, resulting in a linear scaling.

Now we also have to consider that some syndromes remain and are taken care of by the secondary decoder. As shown previously the number of remaining syndromes can be described by an effective error rate, therefore reducing the decoding time by some constant factor, while retaining their original scaling. This means the time scaling of the hierarchical decoder is given by $\max\left(\mathcal{O}(N), O\left(N^{X_{\text{secondary}}}\right)\right)$ the larger time scaling of both part. In the case of the UF and MWPM as secondary decoders, we retain the original scaling of $O(\alpha N)$ and $O\left(N^{1.19}\right)$ respectively.

While the scaling of the decoding time is dominated by the secondary decoder, for the small code distances the actual decoding time is dominated by the neural decoder. Since the calculation of the neural decoder is just an ANN the forward pass through an ANN, it can

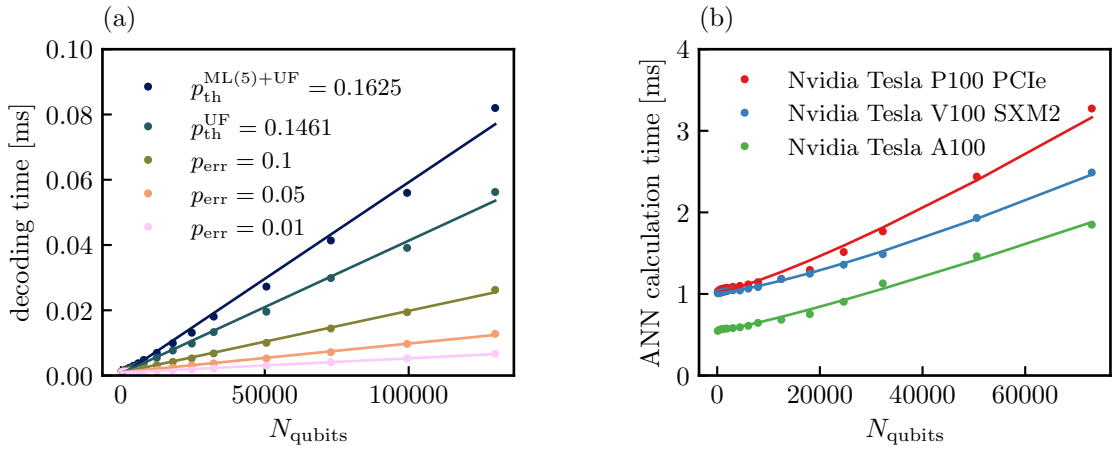


Figure 4.13 – **Scaling behaviour** of the hierarchical decoder. In (a) the decoding time measured as wall-clock time is plotted against the number of qubits for the ML(5) + UF decoder for different error rates. In (b) the measured ANN calculation time is displayed for different GPUs measured for the physical noise rate $p_{\text{err}} = 0.1$.

significantly profit from specialized hardware like GPUs. Using the resources at hand we made an comparison of ANN computation time depending on the number of qubits as shown in Fig. 4.13 (b). Here we test three different Nvidia GPU cards, which can therefore be used as a measure of the improvement of the calculation power. we compare the following graphic cards

- Nvidia Tesla P100 PCIe
- Nvidia Tesla V100 SXM2
- Nvidia A100 Tensor Core-GPU.

Comparing the P100 and V100 card, it is notable that both converge at the same minimum kernel time, which is most likely a result from the having the same connection hardware since both cards come from the same supercomputer environment CHEOPS. Increasing the number of qubits, the V100 card shows a reduction of computation time by $\approx 28\%$ compared to the P100. Now taking also the A100 hard into account, it is notable that the minimum kernel time is roughly halved. This flat reduction in compute time is also carried to to higher number of qubits, as it shows the same linear scaling as the V100, but still reduces computation time further by $\approx 30\%$.

4.5.3 Interpretation of decoding strategies

Seeing that the hierarchical decoder can improve the decoding accuracy at all error rates, it is still open about how much of the decoding is done by the neural and secondary decoder. To judge this relation we will introduce an effective error rate p_{eff} as measure. This measure calculates the matching error rate based on the average numbers of remaining syndrome using

$$p_{\text{eff}} = \frac{\overline{N}_{\text{syn}}}{4\frac{4}{3}d^2}, \tag{4.9}$$

this conversion is based on the assumption of depolarizing noise in the low error rate regime, which is dominated by error chains of length $l = 1$. Using this conversion the effective error

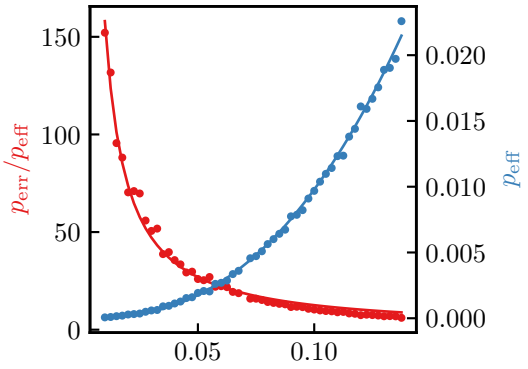


Figure 4.14 –
Neural decoder effectiveness. The graph shows the effective error (blue) and effective error rate (red) of the neural decoder ML(5) depending on the physical error rate. These values are measured for d = 63, but remain invariant under changing the code distance.

rate is shown in Fig.: 4.14, that the neural decoder is able to reduce the effective error rate by two orders of magnitude in the low error regime. For the lowest investigated error rate of $p_{\mathrm{err}} = 0.01$ the error rate is being decreased by a factor of ≈ 150 to $p_{\mathrm{eff}} = 0.00008$, showcasing that the neural decoder is able to correct almost all occurring errors. Looking at the other end of the spectrum at an error rate close to the threshold, the effective error rate increases at faster then the increase of the error rate, resulting in a drop of the improvement factor. Still even close to threshold, we achieve an improvement by a factor of ≈ 6. These results show that most errors are taken resolved by the neural decoder and the secondary decoders only have to deal with a small amount of remaining errors. It should be noted though, that the effective error rate is based on the assumption, that the leftover errors still follow the distribution of the initial error rate, which isn't necessarily true. Since the neural decoder has a bias towards correcting specific errors and by design is unable to correct long error chains, the remaining errors are no longer equally distributed. So the effective error rate should only be taken as an approximation of the number of remaining errors, without information about which errors remain.

We now examine what the neural decoder understands in order to achieve an increase in decoding performance. For this purpose, we apply only the first part of the hierarchical decoder and collect the performed correction over 10^4 correction samples. Fig. 4.15 (a) shows the percentage of the correction type depending on the error rate. The displayed correction types are limited to X,Y,Z corrections, excluding the I correction, in order to make the graph more readable. It is important to note, that the rate of applied corrections are displayed, without considering if the correction was successful or not. In the regime of small error noise, all correction types converge against the same value of $\approx 4.5\%$. Since the small error regime is dominated by errors chains of length $l = 1$, it is to be expected that the neural decoder can learn to correct the errors appropriately. Furthermore, since we examine an independent, uniformly distributed noise model, all three types of errors occur equally often. The limit of 4.5% is a result from the implementation of the neural decoder as described in Sec. 4.4.2. Since we only use qubits adjacent to measured syndromes as input, we always observe more qubits without errors then with errors. In case of a single X or Z error the resulting input for the network would consist of the mask of 7 different qubits, from which one is carrying the error. This results from measuring 2 syndromes, each having 4 adjacent qubits, with one being shared by both. In the case of an Y error, we have 9 input qubits with one carrying the error. Assuming that only errors of length $l = 1$ occur and that the decoder is able to perfectly correct them,

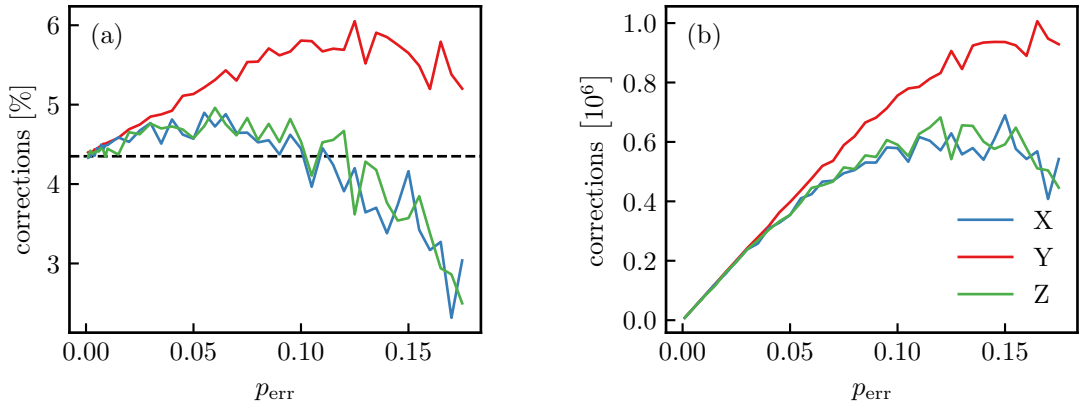


Figure 4.15 – **Correction type analysis**. Graph (a) shows the percentage part of the X/Y/Z corrections (blue/red/green) for the ML(5) neural decoder, plotted against the physical error rate. The black dashed line shows the theoretical limit for perfectly correcting errors in the low error regime. The same corrections are presented in (b) by their collected total number.

the theoretical ratio of a single correction type is given by

$$p_{type} = \frac{1}{3}\frac{1}{\left(7\frac{2}{3}+9\frac{1}{3}\right)} \approx 0.0435, \tag{4.10}$$

matching the observed limited. If the error rate is now increased, this will of course lead to an increase in corrections, as shown in Fig. 4.15 (b). Initially, the number of corrections applied increases steadily until the increase in X and Y stagnates, while the Y corrections are still increasing. At the peak of the x/z correction, roughly 70% more Y corrections are applied then X or Z. There are two factors which can contribute to this distribution. As the error rate increases the average length of the error chain also increases, increasing the difficulty for the neural decoder to correct the errors. In the case of Y errors, there are four syndromes available, allowing for easier identification and making it more robust against long error chains. For example, assuming the extreme case, that an Y error is combined with multiple X errors causing one of the X syndromes to be located outside of the mask, the Y error can still be successfully corrected on the basis of the three remaining syndromes. The same does not apply if there is only a single X or Z syndrome. The other factor is that the increased error density and therefore syndrome density can lead to incorrect Y correction due to the proximity of X and Y syndromes. Furthermore, it can be seen that the neural decoder has difficulty finding the right corrections as it approaches the error threshold. This becomes clear in 4.15 (a), as the percentage of all three correction types starts to decrease near the threshold, meaning that the number of I corrections increases. This is most likely due to having increasingly long range syndrome pairs. Because of syndrome error degeneracy, the neural decoder tends to have difficulties suggesting an appropriate correction, when there are multiple equally probable corrections, as mentioned in 4.4.3. With a higher error rate, differences in the rates of X and Z corrections also occur, varying which correction is applied more often. Since the noise model is independent, there is no reason for either the X or Z error to be decoded more successfully then the other, so the fluctuation are results from the training process of the decoder.

After this high-level analysis of the applied correction, we want to take a closer look at what the neural decoder has learned by feeding it specially constructed syndromes and looking at the resulting outputs. In Fig. 4.16, the measured syndromes are visually represented and the correction with the highest probability is shown on directly connected qubits.

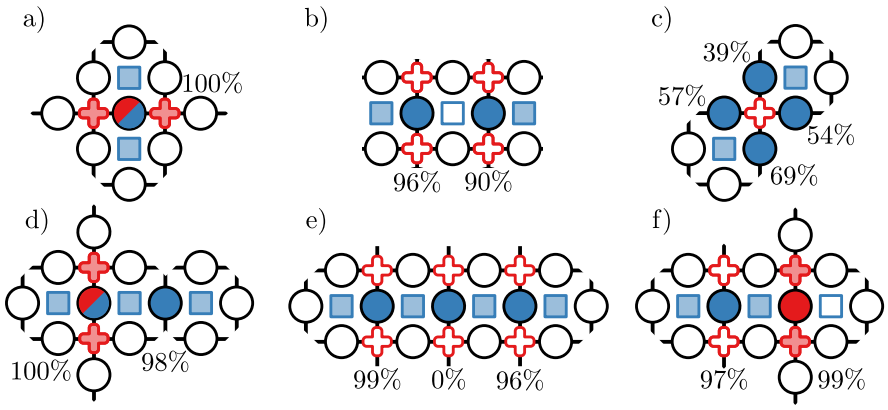


Figure 4.16 – **Learned behavior**. In the graphics the actions of the ML(5) neural decoder are visualized for selected measured syndromes. At the qubits of interest the percentage of performing the correct recovery operation is displayed, where the color of the qubits indicates the correction type with X (blue), Z (red) and Y (blue/red).

We start with two examples which should be easily correctable for the neural decoder. In (a) the syndrome of an Y error results with an appropriate Y correction with a probability of 1. In (b) a bit flip error chain of $l = 2$ is measured, which is the longest error chain that the neural decoder can theoretical completely correct. Applying the decoder returns bit flip correction probability of 96% and 90% and therefore successfully corrects the error. Since the input for both qubits only differs by some rotation, both corrections should have the same probability, assuming the network has been trained perfectly.

Similar to the previous error chain in (c) a bit flip error chain of length $l = 2$ with a two folded degeneracy is measured. The output of the neural network shows a probability of correction between 39% and 69% for the four possible qubits. Since for three qubits three is a probability of $> 50\%$, bit flip corrections was applied to all of them. In this case, the original error chain would not have been corrected successfully, but would only have been reduced to $l = 1$. Since all 4 qubits have a chance of 50% to be part of the minimum-weight correction, the applied correction of the neural decoder depends only on random symmetry breaking in the training progress. After training, corrections might be applied to 0-4 of these qubits.

The next examples tests the neural decoder to distinguish between lazy and non-lazy corrections. In (d) the combination of an Y and X error is shown, which is solved with certainty by the neural decoder. It is important to note, that the decoder did not try to lazy correct a Z error in place of the Y error, since a pair of Z syndromes and only a single X syndrome are directly adjacent. Similar in (e) two bit flip error chains $l = 1$ in close proximity to each other are investigated. Both error chains are corrected appropriately as expected, but also there is no correction applied to the qubit in between both chains, which could have been mistaken for a lazy correction. Finally (f) mimics a syndrome similar to (d), where a qubit is directly adjacent to 2 Z syndromes and one X syndrome, but this time two independent X and Z errors are generated. The neural decoder is able to distinguish between both cases, based on the position of the second X syndrome and only applies X and Z corrections.

This shows that the decoder is able to learn the probability distributions of the error and does not attempt to solely apply lazy corrections. This has also the consequence that it does struggle with error chain, that have a degeneracy of > 1. Tests to mitigate false corrections by applying stochastic corrections instead of deterministic, have shown no advantage in the low error regime and a decrease in performance at higher error rates, as shown in App. B.2.

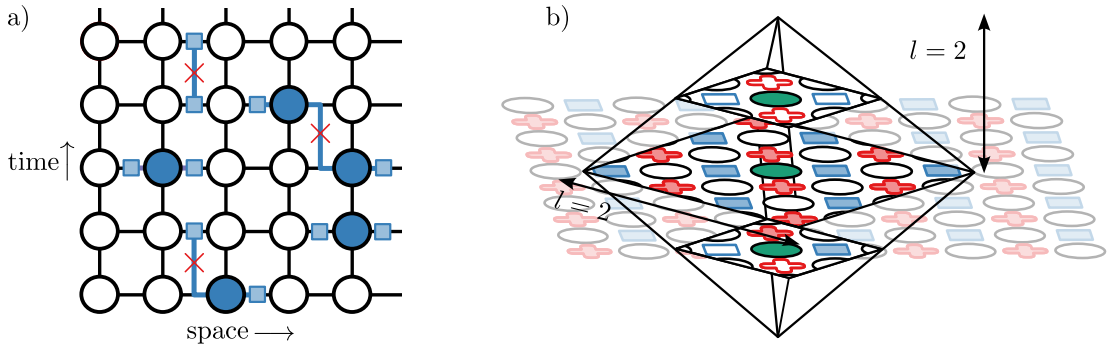


Figure 4.17 – **Faulty syndrome measurements.** Adding faulty syndrome measurements to the toric code means that multiple measurements has to be performed to observe the actual syndrome. This will cause the syndromes to propagate in time. This is shown here for a 1D toric code with only bit flip errors (blue) and faulty syndrome measurements (red crosses). This allows for representing the error strings in $d+1$ dimension. This figure was inspired by [157]. This necessitates to extend the input mask into the third dimension (b). The chosen shape is an octagon around the investigated qubits (green). All syndromes contained inside the octagon are used as input for the mask.

4.6 Faulty syndrome measurement toric code

Now that we have discussed the correction of the toric code under depolarizing noise extensively, we take the next step and look at a phenomenological noise model, where we introduce additionally faulty syndrome measurements into the system. Now, every time a stabilizer operation is performed, there is a chance of p_{syn} that the resulting outcome is flipped. This noise model neglect the possibility of error propagation between data and ancilla qubit, while capturing the essential challenge of fault-tolerant error correction, where the measurement of the errors is inherently faulty.

Because syndromes can no longer be trusted, the syndrome extraction is repeated multiple times adding a time dimension to the toric code, as shown in Fig. 4.17. By looking only at the syndrome changes in time, this corresponds to a 3D toric code [54], therefore by assuming that the faulty syndrome rate p_{syn} and the depolarizing error rate p_{depol} are equally distributed, it simplifies the problem to decoding a 3D toric code under depolarizing noise.

While the size of the time dimension can be arbitrarily chosen, we assume that the formed 3D toric code resembles a cube with $d_x = d_y = d_t$. One difference to the regular 2D code is that we do not have a periodic boundary condition in the time direction. This could lead to the problem that an uneven amount of syndromes are measured. This is a problem for the MWPM and UF decoders, which require an even number of syndromes to perform the correction process. To remedy the situation we assume that the last syndrome measurement is without fault. This guarantees that all error chains are closed at the end and an even number of syndromes remains.

To apply our hierarchical decoder we have to adjust it to the new setting, as it currently does not consider the 3D structure. While the general concept stays the same, we change the input mask from a square to an octahedron, incorporating all syndromes l steps away from the qubit, as shown Fig. 4.17 (b). At the time, borders the mask would stick out at the top or bottom of the QEC. In this case, we apply zero padding as described in Sec. 2.1.3 and set all syndromes lying outside to zero.

At first, we apply the pure MWPM and UF algorithm to the new noise model setting the benchmark values of $p_{\text{MWPM}} = 0.0434$ and $p_{\text{UF}} = 0.038$. As before the MWPM achieves a higher threshold compared to the UF, as shown in 4.18, but compared to the previous results

$\ell = 3$ network parameters (optimal wall-clock time)	
hidden layers	3
hidden nodes per layer	128
total number of free parameter	43,396
activation functions hidden layer	Relu
activation functions output layer	Softmax

Table 4.4 – **Network architecture** for depolarizing noise with faulty syndrome measurements. The table shows the detailed parameters of a wall-clock time optimized network, adapted for the 3D version of the toric code. The table was taken and modified from [P2].

the overall threshold values have lowered to about 1/4 of the previous values. This is an expected development, since with the addition of the faulty syndrome measurement, the overall noise level has increased and thereby the performance of the decoders has decreased.

We use a neural decoder with an input size of $l = 3$, denoted as ML(3) and build from 3 hidden layers with 128 hidden nodes each, resulting in 43,396 free parameters as listed in Tab. 4.4. The calculation of the threshold values comparing the algorithmic decoders with to the hierarchical counterpart, again shows an improvement in the error thresholds. For the MWPM decoder the threshold only increases by $\approx 2\%$ to $p_{\mathrm{ML(3)+MWPM}} = 0.0445$, showing a smaller improvement rate compared to before. In the case of the UF, the threshold improves by $\approx 14\%$ to $p_{\mathrm{ML(3)+UF}} = 0.0434$, achieving same threshold up to the error margin as the pure MWPM decoder, making it once again the better choice.

Looking at the extended benchmark tests, presented in Tab. 4.5, one can see that the highest

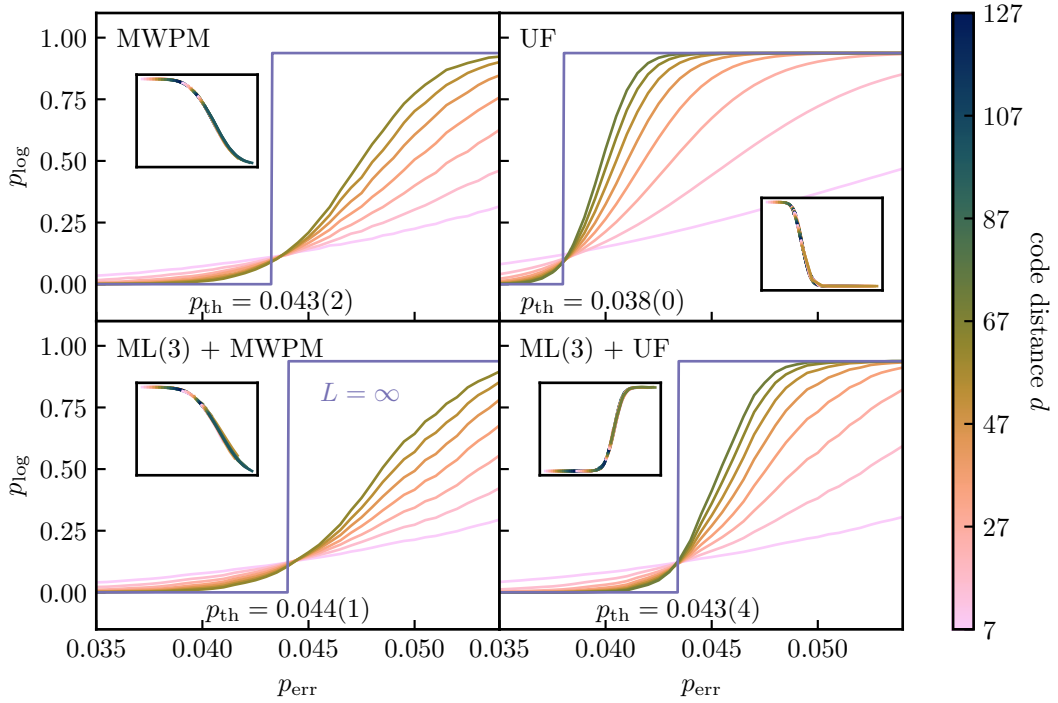


Figure 4.18 – **Depolarizing noise with syndrome measurement noise thresholds.** The threshold plots for the MWPM (top left) and UF (top right) are shown, as well as their hierarchical counter parts ML(3) + MWPM (bottom left) and ML(3) + UF (bottom right). The blue curves show the thermodynamic limit of the curves and the inserts demonstrate the data collapse of the threshold data.

depolarizing noise + syndrome errors ($d = 31$)

algorithm	p_{th}	$t_{p=0.01}$	$t_{p=0.02}$	$t_{p=0.03}$	$t_{p=0.0378}$
ML(3) + UF	0.043(4)	12.1	13.5	15.4	17.8
Lazy + UF	0.031(3)	11.1	12.8	16.6	—
UF	0.037(8)	11.5	13.4	15.7	18.9
ML(3) + PyMatching v2.1	0.044(3)	3.7	**4.9**	**6.4**	**8.8**
PyMatching v2.1	0.043(5)	**3.0**	5.7	9.2	13.2
ML(3) + PyMatching v0.7	**0.044(5)**	14.6	25.8	81.5	229
PyMatching v0.7	0.043(7)	211	239	273	294

Table 4.5 – **Depolarizing noise + faulty syndrome measurement benchmarks.** For various decoders we show the measured benchmarks for the decoding times in milliseconds as well as the error thresholds. Here, the ML(3) represents the neural decoder with $l = 3$ input mask. The decoding times are measured as wall-clock times in milliseconds for code distance d = 31 and error rates $p_{\text{err}} = [0.01, 0.02, 0.03, 0.0378]$, where the last error rate is taken from the UF threshold value. The bold entries highlight the best performing values of the respective columns.

decoding time is still given by the combination of ML(3) + PyMatching v0.7, while the lowest decoding time is now achieved in the low error regime $p_{\text{err}} < 0.03$ by PyMatching v2.1 and in the high error regime $p_{\text{err}} \geq 0.03$ by the combination ML(3) + PyMatching v2.1. Compared to the depolarizing noise case, the ML(3) + UF is no longer able to achieve decoding times rivaling the PyMatching v2.1 decoders. The ML(3) + UF decoder reports an increase in decoding time by almost a factor of 3 in the low error regime compared to the ML(3) + PyMatching v2.1 decoder. Therefore the choice of which decoder to use, falls to the PyMatching hierarchical decoder, as it is able to demonstrate the highest error threshold hold, while also outperforming the algorithmic decoders in most error regimes.

Now, to better understand the performance of the hierarchical decoder, we once again consider the effective error rate. First we have to adjust the calculation of the effective error rate

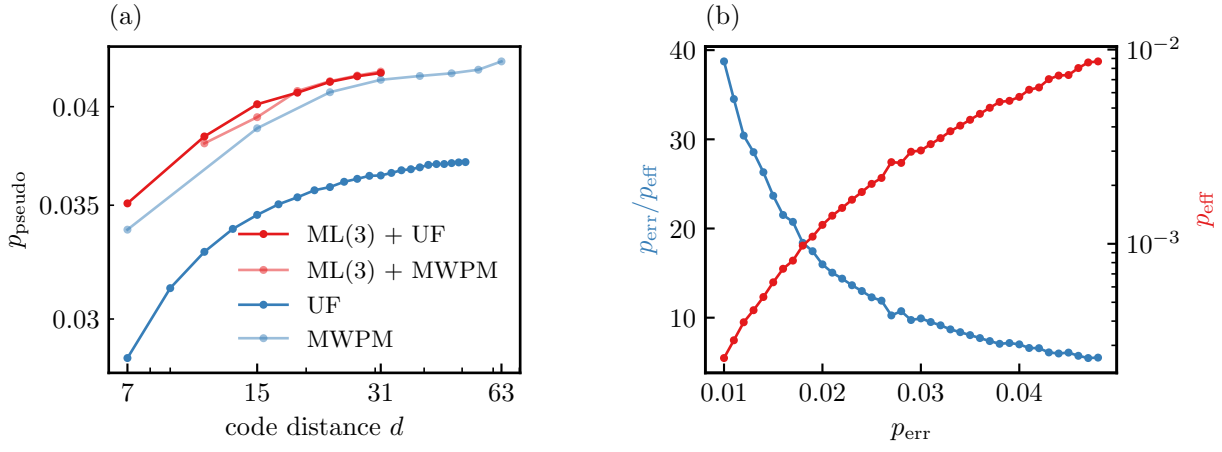


Figure 4.19 – **Faulty syndrome measurements decoding performance.** The decoding performance of the MWPM (light blue) and UF (blue) decoders as well as their ML counterparts ML(3) + MWPM (light red) and ML(3) + UF (red) are visualized using the pseudothreshold depending on the code distance (a). Using only the neural decoder for decoding and measuring the numbers of remaining syndromes allows the representation using an effective error rate (b). This effective error rate (blue) and the improvement factor to the initial error rate (red) are shown depending on the initial error rate.

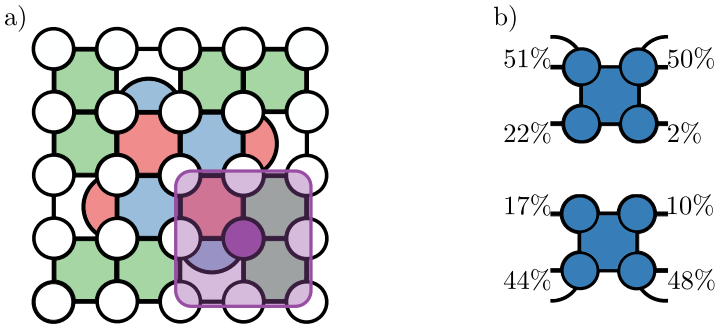


Figure 4.20 – **Neural decoder application on the RSC.** At the boundary of the RSC a padding technique has been applied (a), so that the input mask (violet square) get filled with values of 2 (green) for parts outside of the RSC. After a measurement of a single syndrome at the top or bottom border (b), the probability of the four adjacent qubits is returned.

for the phenomenological noise, which is given as

$$p_{\text{eff}} = \frac{4}{3} 6 d^3. \tag{4.11}$$

Looking at the results showcased in Fig. 4.19 (a), it is once again evident that for small error rates the decoder reduces the errors rate by a factor of 40, decoding almost all occurring errors. Compared to the depolarizing noise, this factor has reduced to a third as before, showcasing that it is more difficult for the neural decoder to handle the phenomenological noise than the depolarizing noise.

4.7 Rotated surface code

Lastly, we also want to take a look at how the decoders perform on the rotated surface code. In general, the error threshold of the RSC is the same as the toric code [54], as for the thermodynamic limit $L \to \infty$ the RSC is equivalent to the toric code with the caveat of only encoding a single logical qubit.

4.7.1 Modifying the hierarchical decoder

In order to apply the MWPM and UF, small adaptions have to made to the algorithm since they need an even number of syndromes to pair up and due to the boundary condition it is possible to measure an odd number of syndromes. There, UF and MWPM have to be able to pair syndromes with the boundaries. In Fig. 4.21 we can see the threshold plot for the 2D RSC under depolarizing noise for UF decoder and the MWPM decoder. As we can see, the error threshold is consistent with the error threshold recorded in Tab. 4.3.

While the hierarchical decoder does not have the limitation of needing even numbers of syndromes in the input mask, it is still necessary to adapt the decoding algorithm. In order to be able to place the input mask centered on a qubit near the borders, we apply a padding technique as described in Sec. 2.2 to fill the entire outside area with constant values of two, as shown in Fig. 4.20 (a) for a mask of size l=3. It is important to choose a padding value beside the values $\{-1, 0, 1\}$ as they are reserved for measured Z syndrome, no syndrome and measured X syndrome. Although it might seem natural to choose a zero padding, as outside the RSC no syndromes are ”measured”, this does not allow the hierarchical decoder to recognize the border

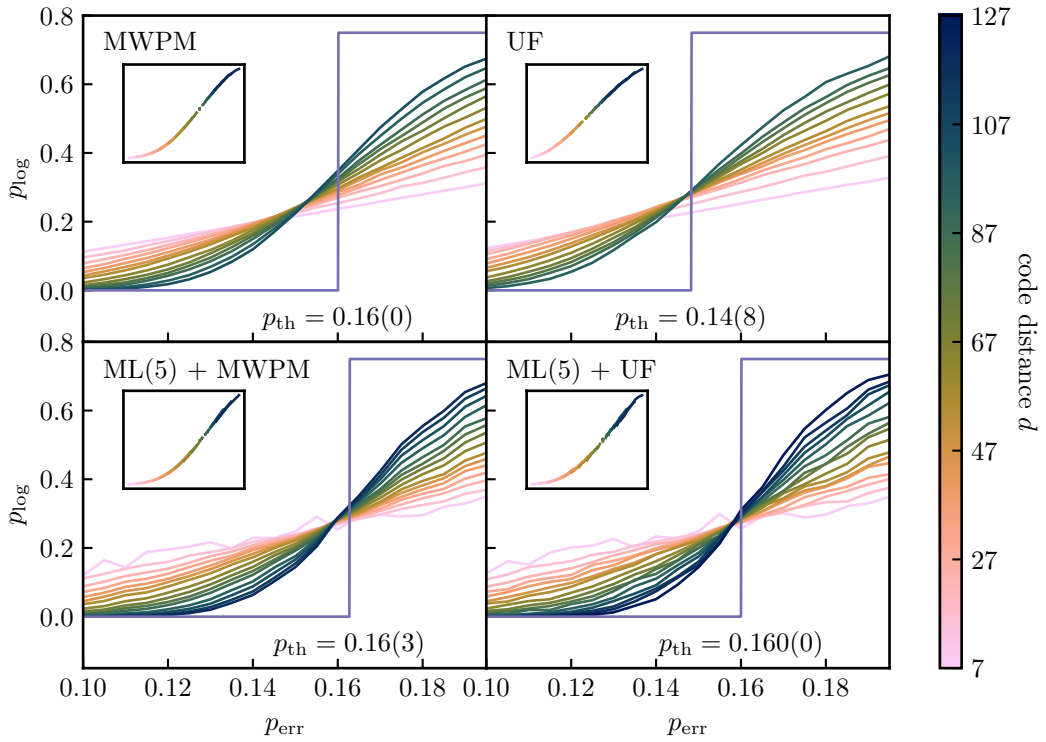


Figure 4.21 – **Error thresholds on the RSC.** Here, the threshold plots for the MWPM, UF, ML(5) + MWPM and ML(5) + UF decoders are shown. The blue curve shows the logical error rate in the thermodynamic limit. The inserts show the data collapse of the threshold data.

and therefore it looses the ability distinguish between a lone syndrome on the top or bottom border, as shown in Fig. 4.20 (b). For both syndromes, the ANN returns the same probability values, as they share the input mask. Therefore all four qubits are equally likely to carry the error, causing the hierarchical decoder to not correct any qubit.

But adding now some other constant padding allows the ANN to differentiate between both sides and to recognize that qubits further away from the border are less likely to carry the error, as shown in Fig. 4.20 (c). This then still leaves with the problem that both qubits at the border are equally probable to carry the error, causing the hierarchical decoder to always correct one of the qubits, or both, or neither, depending on the random symmetry breaking in the training.

4.7.2 Benchmark measurements

We train neural decoder with an input mask of size $l = 3$, and 3 hidden layers with 128 hidden nodes per layer, thereby using the same network structure as the wall-clock time optimized network in Tab. 4.2. The only difference is that we choose a training code distance of $d_{\mathrm{train}} = 17$ in order to minimize finite size effects from the borders. Applying the hierarchical decoder to the RSC using MWPM and UF as secondary decoders returns the error threshold $p_{\mathrm{th}} = 0.163$ and $p_{\mathrm{th}} = 0.160$, showing similar improvements as the toric code.

Looking at the pseudothreshold one can see that the hierarchical decoder not only improves the error threshold, but also improves the pseudothreshold by about 13% for the UF at $d = 9$, while compared to the MWPM the pseudothreshold decreases by roughly 2% as shown in Fig. 4.22. Comparing the pseudothreshold between the RSC and the toric code shows that the

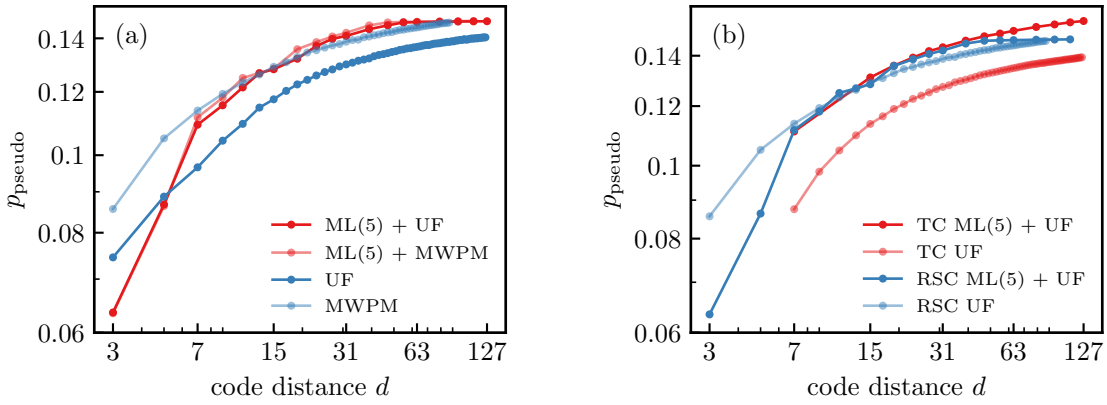


Figure 4.22 – **Pseudothreshold comparison.** Extracted from the threshold data, the pseudothresholds depending on the code distance are shown for the MWPM, UF, ML(5) + MWPM and ML(5) + UF decoders. One comparison is drawn on the 2D RSC between all four decoder variations (a) and one is drawn between the RSC and the toric code (b) for the UF and ML(5) + UF decoders.

RSC UF has a somewhat smaller pseudothreshold than its TSC counterpart, approximately $\approx 6\%$ smaller. The two ML(5) + UF decoders on the other hand show almost the same exact pseudothresholds, with only the RSC having marginally lower pseudothresholds at both ends of the code distances.

Next, we want to take a closer look at the performance of the neural decoder in the hierarchical decoder. Once again we look at the effective error rate to measure how many errors were corrected by the neural decoder. The calculation of the error rate is similar to Eq. (4.9), but the rough boundary has to be considered, where errors only generate singular syndromes. Correcting this terms results in

$$p_{\text{eff}} = \frac{\overline{N}_{\text{syn}}}{\frac{4}{3}(1 + \frac{d(d-2)}{d^2})d^2}. \tag{4.12}$$

In Fig. 4.23 we present the calculated effective error rate depending on the physical noise. We

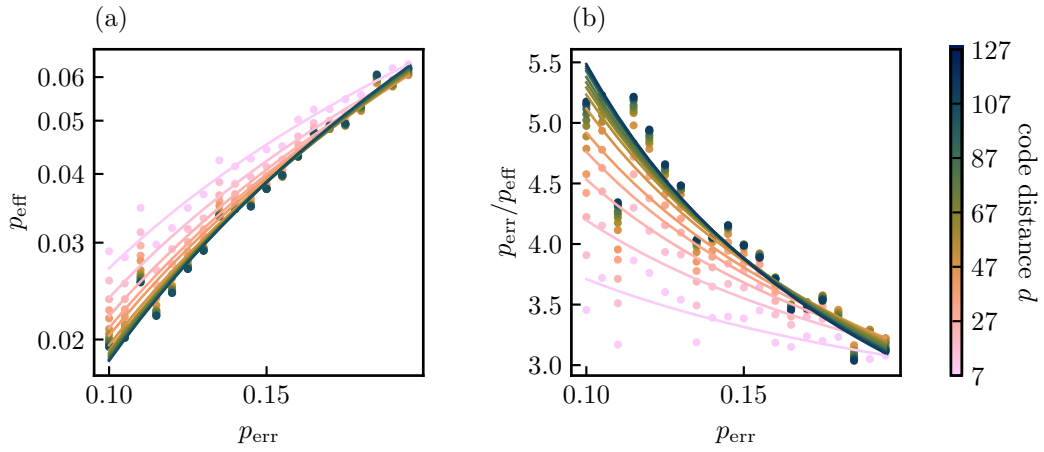


Figure 4.23 – **Performance analysis on the RSC.** From the number of remaining syndromes after applying the neural decoder, the effective error (b) and the error rate improvement (c) are calculated. Both observables are plotted for multiple code distances against the initial error rate.

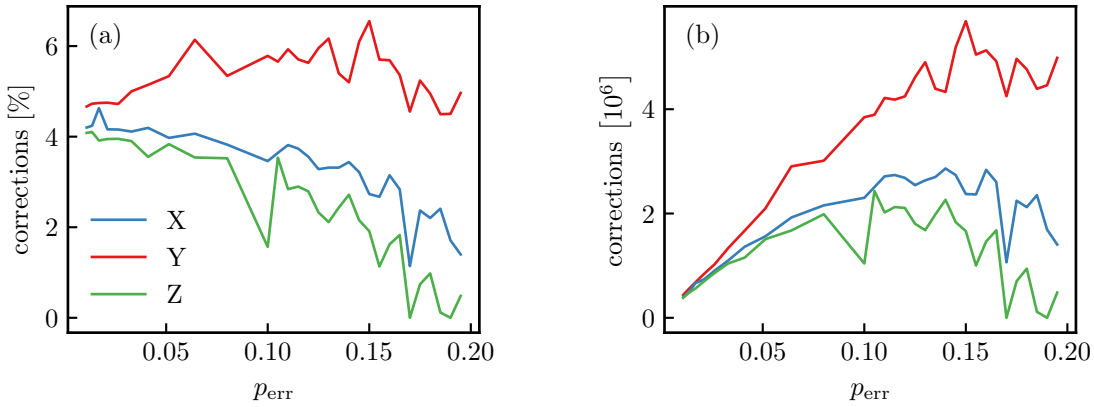


Figure 4.24 – **Applied correction analysis.** Performing decoding on the RSC and applying the neural decoder, the suggested corrections are counted. The number of corrections depending on the type of correction are plotted as percentage (a) and total numbers (b) against the initial error rate.

overlaid the curves for different code distances, showcasing some finite size effects, which weren't observable on the toric code. Looking at low error rates, the effective error rate decreases with code distance, indicating that the neural decoder has trouble with the errors at the border. The same can be seen for the improvement rate, which varies between 3.75 and 5.5 for $p_{\text{err}} = 0.1$. Comparing these results with the 2D toric code, showing an significant decrease in the improvement rate.

Now looking at the different error type corrections performed on the RSC, we can see that for low error rates the three correction percentages of the three different layers are between 4-5%, which are similar values to these for the toric code. It can also be observed that the correction of the three different error types is closer to each other and gives the same value. But different to the toric code, the Y correction still remains above the other two.

Looking at the total number of the different corrections, it naturally increases of number physical errors, as shown in Fig. 4.24. The increase follows the same behavior as for the toric code, as that the Y correction increases more rapidly then the X and Z corrections and as one approaches the error threshold the number of X and Z corrections actually starts to drop again.

4.7.3 Robustness of the decoding strategies

After the initial performance test on the RSC, we also want to test the robustness of the learned control strategies. For this we are going to test the robustness in two different aspects, robustness regarding the training lattice size and training error rate. To judge the robustness of the system, we are going to use the pseudothreshold for the ML(5) + UF version of the hierarchical decoder.

We start with the training lattice size. As we have previously seen in Fig. 4.22 the pseudothreshold of the hierarchical decoder declined towards the small code distances, as the hierarchical decoder did not learn how to correctly factor in the border of the system. In Fig. 4.25 (a) we show the pseudothreshold of the ML(5) + UF decoder, where every iteration of the decoder was trained with the same hyperparamters except the training lattice size, which was varied between $d = 3$ and $d = 9$, as well as the previously determined pseudothreshold at $d = 17$ for comparison. It is appeared that at $d = 3$ the hierarchical decoder has difficulties learning a decoding strategy, as the pseudotrehshold is below the previously determined pseudothreshold at any point. This changes for the $d = 5$, as the performance at small code distances increases

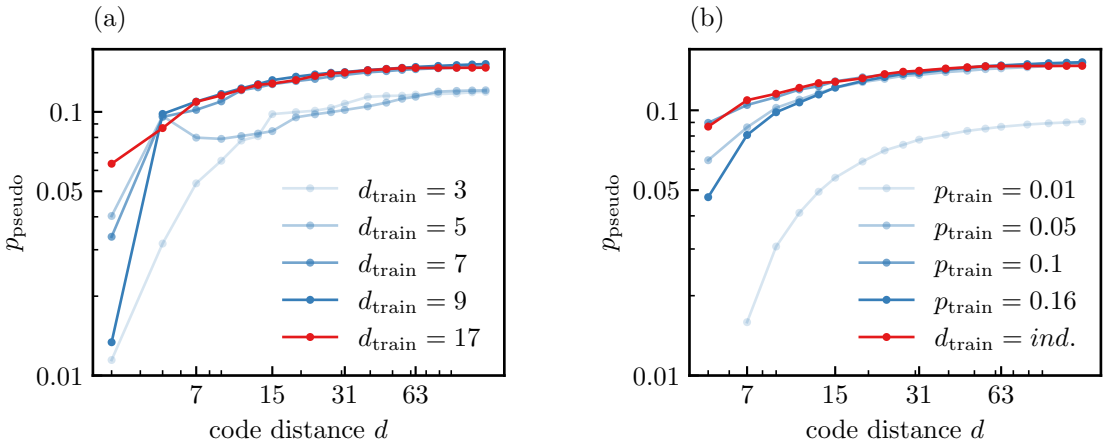


Figure 4.25 – **Robustness of the control strategy** for the ML(5) + UF hierarchical decoder on the RSC. The robustness is visualized using the pseudothreshold for differently trained neural decoders. In (a) the code distance of the trainings RSC was varied between $d_{\text{train}} = 3$ and $d_{\text{train}} = 9$, whereas in (b) only a single model was used trained at p_{train} to determine the pseudothreshold. Both graphs have also included the previously determined results (red), with individually trained model for each error rate and trained at $d_{\text{train}} = 17$.

significantly and spikes at the trained lattice size, being on par with the best performing versions of the hierarchical decoder. Although afterwards the pseudothreshold drops instantly when going to higher code distances, showing a similar performance as the $d = 3$ trained decoder. The remaining two decoders, trained at $d = 7$ and $d = 9$, achieve the same decoding performance as the $d = 17$ decoder. The only exception is the application at $d = 3$, where all three decoders experience a loss of performance. As the performance of the $d = 7$ decoder lies between the $d = 9$ and $d = 17$, it seems that the performance is not depending on the trained lattice size at that point, but on the individual training results instead. In total we can observe that the hierarchical decoder at $d \geq 7$ it demonstrates robustness toward the application on higher and smaller code distances with the exception of $d = 3$.

Now we turn to the robustness regarding the error rate. Until now we have always trained individual decoders for each error rate. In actual experimental application, the error rate may vary with time or is not uniformly distributed on all qubits. Therefore, it is necessary for the hierarchical decoder to be able to successfully decode syndromes generated at different error rates. In Fig. 4.25 (b) the different pseudothresholds are shown for hierarchical decoders trained at different error rates, which have then been applied to every other error rate, as well as the previously determined pseudothreshold. The models are taken from the previously trained models used in Sec. 4.7. We can see that the best performing model is the one trained at $p_{\text{train}} = 0.1$, achieving the same performance as decoder trained individually at every error rate. Increasing or decreasing the training error rate leads to a degradation in performance, where the $p_{\text{train}} = 0.01$ decoder lies significantly below the other decoder.

An explanation for the difference in performance would be that it depends on whether the decoder was able to encounter all necessary syndromes during the training. At $p_{\text{train}} = 0.01$, the decoder will mostly see syndromes of single qubit errors and is therefore not able to learn the correct decoding strategy when encountering syndromes of multi qubit errors. The same is true in reverse for the $p_{train} = 0.16$ error rates, where there are most likely problems with the correct decoding of small errors. However, if an intermediate error rate is chosen for training, the decoder apparently encounters a sufficient number of small and large errors during training to be able to correct both types of errors. This means that the hierarchical decoder can also

show robustness with regard to the applied error rates, which further means that it can cope with time-varying error rates.

Additionally, when comparing the pseuodthreshold curves of both plots in Fig. 4.25, the curves of the single error rate training show a much smoother curve progression. This stems from the fact, that only a single trained model is used per curve in that case, eliminating fluctuation caused by training differences.

4.8 Summary

Throughout this chapter, we have used different decoding algorithms to investigate their performance on topological surface codes. The focus of this investigation is the comparison between the algorithmic decoders MWPM and UF and the newly introduced hierarchical decoder. The hierarchical decoder involves a pre-decoding step followed by the application of an algorithmic decoder. By implementing a neural decoder during the pre-decoding step, we have demonstrated the ability of scalable machine learning based decoding that is able of easily decoding codes with a code distance up to $d = 255$, thus solving the common issue of neural decoders working only on small code distances [79–81, 158–160].

We examine the performance of the hierarchical decoder in terms of error threshold, pseudothreshold, and decoding speed compared to algorithmic decoders. Examining these observables on the toric code when depolarizing noise is applied indicates that the hierarchical decoder considerably increases the error threshold. Additionally, in combination with the UF decoder, it surpasses the pure MWPM decoder in terms of error threshold, albeit with a slightly lower threshold when compared to the ML(5) + MWPM decoder. The performance at smaller error rates and code distances is also compared using the pseudothreshold. The hierarchical decoder outperforms the algorithmic decoders, yet does not exhibit a significant discrepancy in performance, depending on whether UF or MWPM is used as the secondary decoder. Examination of the corrections applied shows that the decoding advantage comes from the neural decoders' ability to correct the correlated Y errors, rather than decoding X and Z errors separately as the UF and MWPM decoders do.

These results hold considerable significance, particularly when factoring in decoding speed. For depolarizing noise, the ML(5) + UF decoder combination demonstrated the lowest decoding time, while maintaining a high logical accuracy, making it a good candidate for implementation. However, it must be noted that the decoding speed benefits were exclusively observed at greater code distances of at least $d = 31$, and only close to the error threshold. Optimally, the goal is to perform quantum computing in error regimes far from the error threshold, where the advantage in decoding time was only seen at even higher code distances. These numbers can be compared to IBM's projected development of quantum computing [161], which aims to build a 10k-100k qubit system as early as 2026, where the hierarchical decoder could display its advantages. Whereas currently, the largest system - Condor - has *only* 1121 qubits [60], where the use of algorithmic decoders seems more sensible.

While it may appear that the hierarchical decoder is too slow for current implementation in the near future, we were limited by the hardware available to us. In the experimental implementation of QEC, the decoding algorithm would not be run by simple CPUs and GPUs, but instead by specialized FPGAs [162, 163]. This allows for the minimization of overhead caused by hardware and software, as demonstrated by the implementation of a variation of the UF decoder on an FPGA [164]. A comparable implementation of the hierarchical decoder could result in a significant improvement in decoding speed, especially since ANNs can be

easily parallelized to achieve faster than linear scaling of parallel time with code distance [165]. Analogous parallel implementations have been examined for minimum-weight perfect matching (MWPM) [166], although these concepts are currently only theoretical and have not been implemented on physical hardware. This is not applicable to the UF algorithm, as at the time of composing this dissertation, no parallelized version of it has been found.

This demonstrated, as proof of concept, that the hierarchical decoder could be a potential choice for a real decoding algorithm, and we further solidified this by extending the noise model to include erroneous syndrome measurements. Despite using a more realistic noise model, the hierarchical decoder achieved a similar increase in decoding accuracy as previously observed.

As the toric code is an optimized quantum error-correcting code which is not experimentally feasible, we also tested the decoder on the RSC. The hierarchical decoder faces a hurdle in that translation invariance is broken due to the disappearance of periodic boundary conditions. The results of the error threshold demonstrate that scalability remains intact and is capable of increasing decoding accuracy. Comparing the effective error rates between the TC and the RSC, the performance is almost identical in the limit of large code distances. This result is anticipated, as the boundaries of the RSC become less significant with greater code distance. In contrast, the performance of the hierarchical decoder drops comparatively at small code distances. This flaw is due to the inherent problem in the neural decoder, which struggles to deal with syndromes having multiple possible recovery options. This is a consequence of utilizing supervised learning. At the border, each singular measured syndrome has two equally probable options to be connected with the boundary, causing confusion for the neural decoder. As a result, the effective error rate decreases, leading to the secondary decoder having more errors to correct and hence an increase in overall decoding time. While there is potential for further improvement in the future, the hierarchical decoder has shown its ability to improve decoding accuracy at small code distances, as shown in the pseudothreshold.

Chapter 5

Quantum feedback control

In this chapter, we will address quantum feedback control, in which individual quantum systems are observed and manipulated over extended periods. Similar to QEC, we will now examine the time evolution of quantum systems and consider the impacts of measurements.

This starts with a look at the standard formalism of classical feedback control, which can be generalized as a dynamical system being observed, which is subject to noise, making the actual state of the system unknown. Contrary to the quantum case the observation does not influence the state of the system.

For the quantum case, we first discuss the process of gaining information from the system using quantum measurements. First, we introduce the general formalism and properties associated with quantum measurements, followed by discussing projective and weak measurements, where the latter provides information of the system without collapsing the state. Using these measurements, a formalism comparable to the classical feedback control can be established for the quantum system, using the *stochastic master equation* to describe the system (**SME**) and derive the equation of motions.

With this in mind, we will examine the *linear quadratic Gaussian control* (**LQGC**) algorithm, a popular classical control algorithm. It possesses the valuable property of being able to optimally control a classical linear system and achieve high performance even in nonlinear cases. This makes the algorithm a suitable performance baseline, against which later control algorithms can be compared.

5.1 Classical feedback control

In this section we want to start with exploring the classical feedback problem [7]. As this is a broad topic we limit ourselves to discussing of discrete systems with Gaussian noise, as the effect of quantum measurement can be modeled as such.

The classical feedback problem works as follows: A dynamical system affected by noise, resulting in fluctuations in the state and measurement results, is driven by some inputs with the goal of influencing the system towards a desired state. These inputs are described by a function that depends on the measurement results. The dynamics can be written as

$$\vec{s}_{t+1} = f(s_t, u_t) + G(s_t, u_t)w_t, \tag{5.1}$$

where $\vec{s}_t$ is the n-dimensional state vector containing all information describing the state of the investigated object e.g. position, momentum, acceleration, angle, etc, and u_t is an m-dimensional control vector describing the external action performed on the system. The time evolution of the system is given by the function f, which depends on the state and control input, and w_t is a set of *Wiener elements* [167] in vector form representing the Gaussian noise. The

matrix G determines how the system is affected by the noise. In the control theory literature, this noise is often referred to as the *process noise* driving the system, which we will adopt as our preferred terminology.

While this describes the dynamics of the system, we also need a description of the observation. The observation process is usually written in the form

$$y_t = h(s_t) + R(t)v_t, \tag{5.2}$$

where the measurement is described by the function h, which returns the n-dimensional measurement vector y_t, and v_t is the *observation noise*[1], described by another set of Wiener elements.

Since the actual state of the system is unknown during the whole process, the control problem is now given by the task of finding a function $c(Y_t)$ based on the returned measurements, which returns a set of inputs u_t for every time step t. This involves defining a *cost function* associated with the system, thereby specifying the desired behavior of the system.

A special simplified case of this scenario is the Gaussian linear system, which reduces the state equations to

$$s_{t+1} = As_t + Bu_t + w_t \tag{5.3}$$
$$y_t = Cs_t + v_t, \tag{5.4}$$

where A,B,C are now linear matrices and both noise vectors are described by zero-mean Gaussian noise, together with their covariances

$$w_t \sim \mathcal{N}(0, \sigma_{\text{dyn}}) \qquad e\left[w_k w_k'\right] = R_k \tag{5.5}$$
$$v_t \sim \mathcal{N}(0, \sigma_{\text{meas}}) \qquad e\left[v_k v_k'\right] = Q_k. \tag{5.6}$$

For this system it is known that it can be optimally controlled using *Linear Quadratic Gaussian Controller* (**LQGC**). This controller itself consists of two distinct algorithms, i.e. the state estimator called *Kalman Filter* [168, 169] and the control algorithm *linear quadratic regulator* (**LQR**) [170].

5.2 From classical to quantum feedback control

The formulation of an equivalent quantum problem brings with it the difficulty of measuring the system. As before in the case of QEC, direct measurement will lead to the collapse of the state. Therefore, an indirect measurement is essential, providing information while maintaining the quantum state. Performing measurements in this manner always incurs an information-gain-disturbance tradeoff [25], whereby the more accurate the information, the stronger the disturbances in the quantum system. An example of such measurements are the weak measurements [171].

5.2.1 Quantum measurements

A fundamental aspect of implementing quantum feedback control is information extraction through measurements. These measurements can be generally described as a collection $\{M_m\}$

[1] To avoid confusion, in control theory the *system* is generally used to describe all parts of the control problem, including the dynamics, measurements, noise sources and control inputs. Therefore, *system noise* is also often used to refer to all the noise sources in the control problem.

of *measurement operators* [172]. These operators act on the state space of a quantum system and return the measurement result m. The probability to measure the result m when acting on the system $|\psi\rangle$ is given by

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle. \tag{5.7}$$

These measurement operators have to fulfill the completeness equation

$$\sum_m M_m^\dagger M_m = I, \tag{5.8}$$

equivalently, this equation can be expressed using the Born rule [9] to obtain the measurement probabilities

$$\sum_m \langle\psi|M_m^\dagger M_m|\psi\rangle = \sum_m p(m) = 1, \tag{5.9}$$

enforcing that the probabilities over all possible measurement results sum to one.

In the classical case, performing a measurement does not influence the system, whereas in the quantum case it results in a backaction on the system and thereby changing the state. Performing a measurement M_m results in the post measurement state

$$|\psi'\rangle = \frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}. \tag{5.10}$$

As one can see from this, the backaction on the system depends on the measurement result m.

While we previously examined the overall formalism of quantum measurement, we now want to look at an important special case of quantum measurement, the projective measurement. The projective measurement is described by an observable M, which is a Hermitian operator acting on the state space. The spectral decomposition of the operator is given by

$$M = \sum_m mP_m, \tag{5.11}$$

where P_m is the *projector* onto the eigenspace of M with the eigenvalues m. The projector itself is a Hermitian operator defined as

$$P_m = |m\rangle\langle m|, \tag{5.12}$$

with $|m\rangle$ being an eigenvector of M with eigenvalue α and forming an orthonormal basis. From this follow that two projectors are orthonormal to each other $P_iP_j = \delta_{ij}P_i$. Using this relation and the fact that the projectors are Hermitian operators, they fulfill the completeness relation Eq. (5.8) for quantum measurement

$$\sum_m P_m^\dagger P_m = \sum_m P_m P_m = \sum_m P_m = I \tag{5.13}$$

Following directly from Eq. (5.9), the probability of the measurement results is

$$p(m) = \langle\psi|P_m|\psi\rangle \tag{5.14}$$

and similarly the post measurement state of the system is given by the eigenstate $|m\rangle$ following Eq. (5.10) as

$$|\psi'\rangle = |m\rangle\langle m|\psi\rangle = m|m\rangle \tag{5.15}$$

with $\langle m|\psi\rangle = m$, which is also known as the projection postulate [173].

Now suppose that we have system a $|\psi\rangle$ and want to perform repeated projective measurements in order to gain information. The first measurement returns the measurement result m and leaves the system in the post measurement state $|\psi_m\rangle = (P_m|\psi\rangle)/\sqrt{\langle\psi|P_m|\psi\rangle})$. Applying another projective measurement directly after the first one does not change the state as $P_m|\psi_m\rangle = |\psi_m\rangle$ and therefore we have $\langle\psi_m|P_m|\psi_m\rangle = 1$. This means that the second measurement will return the same measurement result m, with the same being true for any further measurements.

This shows that the projective measurement collapses the system onto an eigenstate [174] and is therefore not viable for quantum feedback control, as repeated non-destructive measurements are needed. Nevertheless, projective measurements still play an important task in the construction of non-constructive measurements called *weak measurements*.

5.2.2 Weak measurements

In order to get information of the quantum system without disturbing it we choose to perform weak measurements on the system. The idea is to couple the system wavefunction $|\psi\rangle$ with an ancilla wavefunction

$$|\phi\rangle = \frac{1}{(2\pi\sigma^2)} \int dm' \exp\left[-m'^2/\left(4\sigma^2\right)\right] |m'\rangle, \tag{5.16}$$

where σ determines the width of the wavefunction, via the interaction term $H_{\text{int}} = A \otimes p$, where A is an arbitrary Hermitian operator with an continuous spectrum of eigenvalues α. With $|\Psi\rangle = |\psi\rangle \otimes |\phi\rangle$ being the combined initial state, the time evolution is given by

$$\begin{aligned} |\Psi'\rangle &= U(t)\,|\Psi\rangle && (5.17)\\ &= \exp\left(-ixtH_{int}\right)|\Psi\rangle = \exp\left(-ixtA \otimes p\right)|\Psi\rangle && (5.18)\\ &= U(t)\,|\Psi\rangle = \exp\left[-i\lambda\left(-i\alpha\frac{\partial}{\partial m}\right)\right]|\Psi\rangle, && (5.19) \end{aligned}$$

where x is the interaction strength. It is assumed here, that the time evolution is dominated by the interaction during the interaction time t. We express both these parameters via a single parameter $\lambda = xt$ characterizing the interaction strength of both systems. After the interaction time we perform a projective measurement on the ancilla wavefunction, where the measurement action is described by $\prod_m = I \otimes |m\rangle\langle m|$, resulting in the conditional state after the measurement

$$|\Psi_m\rangle = \frac{\prod_m |\Psi'\rangle}{\sqrt{\langle|\Psi'|\prod_m|\Psi'\rangle}} = \frac{M|\psi\rangle}{\sqrt{\langle|\psi|M_m^\dagger M_m|\psi\rangle}} \otimes |m\rangle. \tag{5.20}$$

The operator M_m is the so-called Kraus operator of the measurement and follows from the ancilla wavefunction 5.16 as

$$M_m = \int d\alpha \frac{1}{(2\pi\sigma^2)^{1/4}} \exp\left[-(\lambda\alpha - m)^2/\left(4\sigma^2\right)\right] |\alpha\rangle\langle\alpha|, \tag{5.21}$$

using the spectral representation of the operator $A = \int d\alpha\, \alpha|\alpha\rangle\langle\alpha|$.

After performing the measurement, the ancilla wavefunction is no longer correlated to the system wavefunction and just records the measurement result. By taking the trace over it, we

regain the conditional state of the system wavefunction recovering the post measurement state as in Eq. (5.10)

$$|\psi_m\rangle = \frac{M_m|\psi\rangle}{\sqrt{\langle|\psi|M_m^\dagger M_m|\psi\rangle}}. \tag{5.22}$$

With this we can see that the post measurement state of the system depends on the measurement result of the weak measurement. Therefore, it is necessary to determine, the probability density $p(m)$ of the measurement result m. With $|\psi\rangle = \int d\alpha\, \psi(\alpha)\,|\alpha\rangle$ the probability density follows as $P(m) = Tr\left[M(m)^\dagger M(m)\,|\psi\rangle\langle\psi|\right]$, which we can write as

$$P(m) = \frac{1}{\sqrt{2\pi\sigma}} \int_{-\infty}^{\infty} d\alpha\, |\psi(\alpha)|^2 e^{-(\lambda\alpha - m)^2/(2\sigma^2)}. \tag{5.23}$$

Using the probability density we can calculate the uncertainty of the measurement $\sigma_m = \langle m^2\rangle - \langle m\rangle^2$. First, we will calculate the expectation value $\langle m\rangle$

$$\begin{aligned}
\langle m\rangle &= \int_{-\infty}^{\infty} dm\, mP(m) && (5.24)\\
&= \frac{1}{\sqrt{2\pi\sigma}} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} dm\, d\alpha\, m|\psi(\alpha)|^2 e^{-(\lambda\alpha - m)^2/(2\sigma^2)} && (5.25)\\
&= \frac{1}{\sqrt{2\pi\sigma}} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} dx\, d\alpha\, (\lambda\alpha - x)\,|\psi(\alpha)|^2 e^{-(x)^2/(2\sigma^2)} && (5.26)\\
&= \frac{1}{\sqrt{2\pi\sigma}} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} dx\, d\alpha\, (\lambda\alpha)\,|\psi(\alpha)|^2 e^{-(x)^2/(2\sigma^2)} && (5.27)\\
&= \int_{-\infty}^{\infty} d\alpha\, |\psi(\alpha)|^2 \lambda\alpha && (5.28)\\
&= \lambda\langle A\rangle, && (5.29)
\end{aligned}$$

where we have substituted $x = \lambda\alpha - m$ and used the solutions of the Gaussian integrals $\int_{-\infty}^{\infty} dx\, e^{-ax^2} = \sqrt{\frac{\pi}{a}}$ and $\int_{-\infty}^{\infty} dx\, xe^{-ax^2} = 0$. The expectation value of the measurement is then given by the expectation value of the operator A with the additional factor λ. We can do the same process for $\langle m^2\rangle$

$$\begin{aligned}
\langle m^2\rangle &= \int_{-\infty}^{\infty} dm\, m^2 P(m) && (5.30)\\
&= \frac{1}{\sqrt{2\pi\sigma}} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} dm\, d\alpha\, m^2|\psi(\alpha)|^2 e^{-(\lambda\alpha - m)^2/(2\sigma^2)} && (5.31)\\
&= \frac{1}{\sqrt{2\pi\sigma}} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} dx\, d\alpha\, (\lambda\alpha - x)^2\,|\psi(\alpha)|^2 e^{-(\lambda\alpha - m)^2/(2\sigma^2)} && (5.32)\\
&= \int_{-\infty}^{\infty} d\alpha\, |\psi(\alpha)|^2 \left[2\sigma^2 + \lambda^2\alpha^2\right] && (5.33)\\
&= 2\sigma^2 + \lambda^2\langle A^2\rangle. && (5.34)
\end{aligned}$$

Here we have used the same tricks as before with the additional Gaussian integral $\int_{-\infty}^{\infty} dx\, x^2 e^{-ax^2} = \sqrt{\frac{\pi}{a}}\frac{1}{2a}$. Combining both results yields

$$\begin{aligned}
\sigma_m^2 &= \lambda^2\langle\alpha^2\rangle + 2\sigma^2 - \lambda^2\langle\alpha\rangle^2 && (5.35)\\
&= \lambda^2\sigma_\alpha^2 + 2\sigma^2. && (5.36)
\end{aligned}$$

This result allows us to write the measurement result as a stochastic quantity

$$m = \lambda\langle A\rangle + \Delta W, \tag{5.37}$$

where ΔW is a Wiener increment describing a zero mean Gaussian $\mathcal{N}\left(0, \lambda^2\sigma_\alpha^2 + 2\sigma^2\right)$.

From this we can see that in the case $\lambda = 1$ and $\sigma \to 0$, the uncertainty of the measurement reduces to the uncertainty of the system wavefunction. Therefore, using weak measurements returning the measurements outcomes m, n of two observables, obeying the canonical commutation relations $[\alpha, \beta] = i\hbar$, the known limit of the uncertainty principle is recovered

$$\sigma_m^2\sigma_n^2 = \sigma_\alpha^2\sigma_\beta^2 \geq \frac{1}{4}. \tag{5.38}$$

Considering the case with $\lambda = 0$, we have no interaction between the system and ancilla wavefunction, and the uncertainty of a measurement reduces the variance of the ancilla wavefunction to $\sigma_q^2 = 2\sigma^2$.

5.2.3 Quantum system

Having a quantum system that evolves in time and is observed using continuous weak measurements, suffices to setup up a formalism for quantum feedback control. Using the previously formulated weak measurements of the observable A we can write down the respective SME

$$d\rho = -\frac{i}{\hbar}[A, H] - \frac{k\lambda^2}{4}[A, [A, \rho]]dt + \frac{\sqrt{k}\lambda}{2}(A\rho + \rho A - 2\langle A\rangle\rho)dW \tag{5.39}$$

$$dy = \lambda\langle A\rangle + dW, \tag{5.40}$$

where we recover the general SME form of Eq. (1.8). A detailed derivation of the equation can be found in App. C.1.

Looking at the measurement results in Eq. (5.40), we can already see a comparison to the classical feedback formalism, as it only depends on the actual state and a Wiener increment. But in order to compare both cases with each other it is still necessary to extract the equation of motion from the master equation. We can obtain the equation of motion for the expectation value of the operator B using $d\langle B\rangle = Tr\left[Bd\rho\right]$. From this we get the equation of motion

$$\begin{aligned} d\langle B\rangle = &-\frac{i}{\hbar}\langle B, H]\rangle dt \\ &+ \frac{k\lambda^2}{4}\left\langle ABA - \frac{1}{2}\left(A^2B + BA^2\right)\right\rangle dt \\ &+ \frac{\sqrt{k}\lambda}{2}(AB + BA - 2\langle B\rangle\langle A\rangle)dW. \end{aligned}$$

Contrary to the measurement results, this term does not look similar to the structure of the classical equation of motions. So let us examine a more concrete example by investigating a quantum system with an Hamiltonian with an inverse quadratic potential and a controllable time dependent force F, which is observed using position weak measurements.

$$H = H + H_c \tag{5.41}$$

$$= \frac{p^2}{2m} - \frac{k}{2}x^2 - Fx \tag{5.42}$$

Using Eq. (5.41) the position and momentum moments under time evolution are described via

$$d\langle x\rangle = \frac{\langle p\rangle}{m}dt + \sqrt{\kappa}\lambda^2 C_{xx}dW \tag{5.43}$$

$$d\langle p\rangle = (-k\langle x\rangle) + F)\,dt + \sqrt{\kappa}\lambda^2 C_{xp}dW, \tag{5.44}$$

with $C_{xx} = \langle x^2\rangle - \langle x\rangle^2$ and $C_{xp} = \frac{1}{2}\langle xp + px\rangle - \langle x\rangle\langle p\rangle$. By expressing the equation using the state vector $\vec{s}_t = (\langle x\rangle, \langle p\rangle)^T$ and the control vector $\vec{u}_t = (0, F)^T$ we are able to recover the structure given by the classical state equation in Eq. (5.1). This suggest that it is possible to successfully apply control techniques designed for classical systems to quantum systems [3]. But compared to the classical system, there are two differences,which can be observed. The first one is that the applied noise is now correlated, as the measurement and process noise both depend on the same dW, which form is determined by the applied weak measurement. The second difference is the process noise clear dependence on the covariances C_{xx} and C_{xp}. The covariances are determined by the form of the system, and are therefore subject to change over time, meaning that the process noise has gained a time dependence. So when applying a classical control strategy, these two factor have to be considered.

In the following we want to discus one control technique, which is a standard technique for classical system, but has also seen successful application in quantum systems.

5.3 Linear quadratic Gaussian control

Having formulated the description of the feedback control problem for both the classical and quantum cases, we turn to the active control part. Based on the observation of the system, the control algorithm chooses a suitable control input. As a control algorithm we introduce the well-known technique called *linear quadratic Gaussian control* **LQGC**, which allows for optimal control in classical linear system under Gaussian noise [175]. This algorithm is split into two separate algorithms. The first being a renowned state estimation technique called (*Kalman Filter*) [175]. This filter allows the control even in the presence of statistical noise and has therefore found a wide range of applications, the most notable example being its incorporation into the navigation computers of the Apollo program [176] and spacecraft docking at the International Space Station [177]. The second algorithm is the *linear quadratic regulator* (**LQR**) [178], which returns the control input based on the state of the system.

5.3.1 Kalman filter

We start the description of the LQGC by looking at the fist step, the state estimation done by the Kalman filter. The process of the state estimation can be described as calculating the most probable state based on the measurement and the previous state.

Linear system The first case we consider is a linear Gaussian system as described in Eq. (5.3). It is assumed that the system dynamics A, B, C, as well as the covariance matrices R, Q are known. While these matrices can possibly be time dependent, meaning that the system can be time dependent and the noise non-stationary, we will assume here that they are time independent. This does not change the derivation of the Kalman Filter, and all conclusions for the time independent case are also true for the time dependent case. Additionally, the initial state s_0 is generally unknown, but is sampled from a Gaussian distribution with known mean and variance. It is assumed that the initial state and the two noise sources (system and

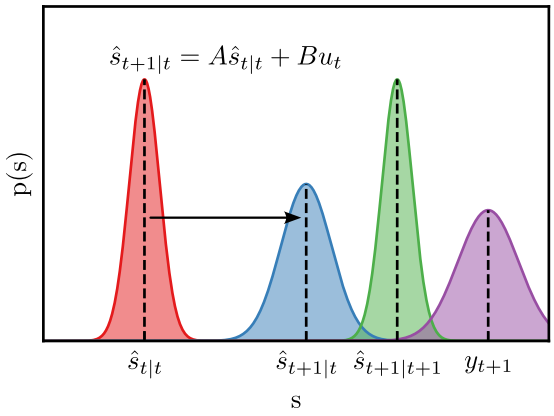


Figure 5.1 – **Kalman Filter step**. Assuming a one-dimensional state described by the variable s in a linear system, a Kalman Filter step can be visualized via the probability density function $p(s)$ of the state and state estimations. Based on the latest known estimate (red) the next state (*blue*) can be predicted based on the dynamics A and input u_t of the system. Together with the measurement of the new state (violet) the new estimate can be calculated (green).

measurement) are *mutually independent* from each other. This assumption corresponds to the *linear Gaussian* (**LG**) assumption.

Below we use the following notation for the conditional mean

$$\hat{s}_{j|k} \coloneqq E\left[s_j|Y^k\right], \tag{5.45}$$

where

$$Y^k \coloneqq \{y_i, i \leq k\} \tag{5.46}$$

denotes the sequence of observations available at the time step k. From this we can differentiate between the three distinct cases of the state:

- estimate of the state if $j = k$
- smoothed value of the state if $j < k$
- predicted value of the state if $j > k$

As the goal of estimator is to calculate the estimates of the state, the goal will be to minimize the estimation error

$$\tilde{s}_{j|k} \coloneqq s_j - \hat{s}_{j|k}. \tag{5.47}$$

Additionally we define the *conditional convariance matrix* of s_k given the data history Y^k

$$P_{j|k} \coloneqq E\left[\left(s_j - \hat{s}_{j|k}\right)\left(s_j - \hat{s}_{j|k}\right)'|Y^k\right] \tag{5.48}$$

$$= E\left[\tilde{s}_{j|k}\tilde{s}'_{j|k}|Y^k\right]. \tag{5.49}$$

The task of the Kalman filter is therefore to map the conditional mean of the current time step $\hat{s}_{t,t}$ and associated covariance matrix $P_{t|t}$ onto the next time step, namely $\hat{s}_{t+1|t+1}$ and $P_{t+1|t+1}$, based on the measurements taken up to this point Y^t.

The predicted state follows from applying the state estimation Eq. 5.3 to the estimation of the current state $\hat{s}_{t|t}$. Since the noise sources are all zero mean Gaussians, the predicted state and the associated state prediction covariance are given by

$$\hat{s}_{t+1|t} = A\hat{s}_{t|t} + Bu_t \tag{5.50}$$

$$P_{t+1|t} = AP_{t|t}A^T + Q. \tag{5.51}$$

Using the same procedure, we calculate the predicted measurement $\hat{y}_{t+1|t}$ and the measurement prediction covariance $S_{t+1|t}$

$$\hat{y}_{t+1|t} = C\hat{s}_{t+1|t} \tag{5.52}$$
$$S_{t+1} = CP_{t+1|t}C^T + R. \tag{5.53}$$

From the calculated covariance follows the Kalman gain K as

$$K_{k+1} \coloneqq P_{t+1|t}C^T S_{k+1}^{-1}. \tag{5.54}$$

The Kalman Filter then allows us to determine the updated state estimate and state covariance

$$\hat{s}_{t+1|t+1} = \hat{x}_{t+1|t} + K_{k+1}\hat{z}_{k+1} \tag{5.55}$$
$$P_{t+1|t+1} = (I - K_{t+1}C)P_{t+1|t}, \tag{5.56}$$

where $z\,(t+1) \coloneqq y_{t+1} - \hat{y}_{t+1|t} = \tilde{y}_{t+1|t}$ is the measurement residual.

From Eq. 5.55 it follows that the Filter is proportional to the state prediction variance, while it is inversely proportional to the measurement residual variance. This leads to the intuitive interpretation of the filter that an inaccurate state prediction (large variance) and an accurate measurement (small variance) lead to a *large* Kalman Filter. This indicates a rapid response in updating the state prediction based on the measurement, allowing large steps between two state predictions. The other way around we have a *small* Kalman Filter for accurate state predictions and inaccurate measurements. Therefore, the state prediction only changes slowly based on the measurement.

To complete the description of the Kalman Filter cycle, we have to look at the first time step. It is assumed that the mean and covariance of the initial state are known

$$E\left[s_0|Y^0\right] = \hat{s}_{0|0} \tag{5.57}$$
$$\operatorname{cov}\left[s_0|Y^0\right] = P_{0|0} \tag{5.58}$$

and therefore can be inserted into Eq. (5.50) & Eq. (5.51).

The computational cost of the Kalman filter is reported to be $(\left(N^3\right)$, where $N = \max(n, m)$ is determined by the dimension of the state and control vector [169].

5.3.2 Kalman filter with same time step correlated noise

Normally it is assumed that the linear system has uncorrelated process and measurement noise. But since we want to apply the LQGC for quantum feedback control, we have to assume correlated system and measurement noise in order to compensate for the weak measurements backaction depending on the measurement results.

Therefore, we now take a look on how to compensate for correlated noise in the Kalman Filter. We consider cross-correlated noise at the same time defined by

$$E\left[v_k v_j'\right] = Q\delta_{jk} \tag{5.59}$$
$$E\left[w_k w_j'\right] = R\delta_{jk} \tag{5.60}$$
$$E\left[v_k w_j'\right] = U\delta_{jk}, \tag{5.61}$$

where the cross correlation between the process and measurement noise is indicated by Eq. 5.61. It has to be noted that these correlations only occur in the same time step, as indicated by the Kronecka Deltas.

To account for this cross correlation it is necessary to rewrite the state equation, in such a form that it has a new process noise, which is uncorrelated from the measurement noise [169]. Using an arbitrary matrix T, we transform the system to

$$\begin{aligned} s_{t+1} &= As_t + Bu_t + w_t + T[y_t - Cs_t - v_t] && (5.62)\\ &= (A - TC)s_t + Bu_t + w_t + Ty_t - Tv_t && (5.63)\\ &= A^* s_t + u_t^* - w_t^*, && (5.64) \end{aligned}$$

with the new transition matrix $A^* = (A - TC)$, new known input $u_t^* = Bu_t + Ty_t$ and the new noise $w_t^* = w_t + Tv_t$. The resulting equation is *completely equivalent* to 5.3, as $y_t - Cs_t - v_t = 0$.

Now the matrix T is chosen, by setting the correlation between the new system noise and the measurement noise to zero

$$E[w_t^* v_t'] = E[[w_t + Tv_t]v_t'] = S - TR = 0, \tag{5.65}$$

which yields

$$T = SR^{-1}. \tag{5.66}$$

From this follows the covariance of the new process noise as:

$$Q^* = Q - SR^{-1}S' \tag{5.67}$$

Using the new state Eq. 5.64 and covariance 5.67, the Kalman gain can be calculated in the usual way, following the steps laid out previously in Sec. 5.3.1

5.3.3 Extended Kalman filter

In the cases discussed previously, the Kalman filter was applied only to linear systems. Now we want to include non-linear dynamics as well as nonlinear measurements, as described in Eq: 5.1 & 5.2, turning the Kalman filter into the *extended Kalman filter* **EKF**.

In this case, the system is no longer described by the matrices A, B, C, but instead by the nonlinear functions $f(s_t, u_t)$ and $g(s_t)$ for the dynamics and measurement, respectively. What still stays the same is, that the system is afflicted with additive Gaussian noise.

The basic principle of the Kalman filter remains unchanged as it calculates the predicted state through recursive computation of the conditional means and covariances. In the nonlinear case, the previous assumption of the conditional mean in Eq. (5.45) is no longer exact, but is instead an approximation $\hat{s}_{t|t} \approx E\left[s_t|Y^t\right]$ and with the associated covariance matrix $P_{t|t}$[2]. For the EKF this is achieved by linearization of the nonlinearities in the dynamics and measurements using first order-series expansions, resulting in

$$A_t = \frac{\partial f}{\partial s}|_{\hat{s}_t, u_t} \tag{5.68}$$

$$C_t = \frac{\partial g}{\partial s}|_{\hat{s}_t} \tag{5.69}$$

[2]Strictly speaking $P_{t|t}$ is no longer the covariance matrix but instead the MSE matrix, since $\hat{s}_{t|t}$ is only an approximation of the conditional mean.

where A, C are now the Jacobians of their respective functions, linearized around $\hat{s}_t, u_t$. The calculation of the Jacobian occurs at the beginning of every cycle, and from then on the EKF follows the same procedure as the standard Kalman filter in Sec. 5.3.1. First, the predicted state and associated covariance is calculated

$$\hat{s}_{t+1|t} = f(s_t, u_t) \tag{5.70}$$
$$P_{t+1|t} = A_t P_{t|t} A_t^T + Q, \tag{5.71}$$

and then the same is done for the measurement

$$\hat{y}_{t+1|t} = g(\hat{s}_{t+1|t}) \tag{5.72}$$
$$S_{t+1} = C_{t+1} P_{t+1|t} C_{t+1}^T + R. \tag{5.73}$$

From the state and measurement covariance we then calculated the Kalman filter

$$K_{k+1} = P_{t+1|t} C_{t+1}^T S_{k+1}^{-1}. \tag{5.74}$$

At the end we can again calculate the estimated state and covariance of the next time step

$$\hat{s}_{t+1|t+1} = \hat{s}_{t+1|t} + K_{k+1} \hat{z}_{k+1} \tag{5.75}$$
$$P_{t+1|t+1} = (I - K_{t+1} C_{t+1}) P_{t+1|t}, \tag{5.76}$$

where $\hat{z}_{t+1} = y_{t+1} - \hat{y}_{t+1|t}$ is again the measurement residual.

The biggest difference in the calculation of the EKF compared to the Kalman filter is the added evaluation of the Jacobian of the state and measurement functions. Because of this, the calculation of the covariances is no longer decoupled from the state estimation calculation, since the state prediction in Eq. (5.70) has to be performed before evaluating C_{k+1}, as this is required in Eq. (5.73). As a consequence, this covariance calculation can generally not be done offline, meaning that they cannot be calculated beforehand.

It is also important to point out that the EKF is not an optimal decoder for the nonlinear Gaussian system, because of the approximation of the dynamic and measurement functions. It has been demonstrated that the advantage in estimating the covariance $P_{t|t}$ using optimal nonlinear estimation compared to the best possible linear estimator can range from 1% for inaccurate measurements to 6% for accurate measurements [169]. The largest difference was achieved somewhere between these two extremes, where the improvement was 15%, demonstrating that the EKF can still achieve high performance in controlling nonlinear systems.

5.3.4 Linear quadratic regulator

The second part of the LQGC consists of the linear quadratic regulator LQR, a control algorithm designed to operate a system at minimum cost [170]. In the case where the dynamics are described by a linear system, subjected only to measurement noise, and the cost is defined by a quadratic function, the LQR is able to optimally control the system [178].

We start the derivation of the LQR for a linear system given by

$$s_{t+1} = As_t + Bu_t, \tag{5.77}$$

where s_t is once again the state vector, u_t the control vector and the matrices A, B define the dynamics of the system. Compared to the previously defined system for the LGC problem in

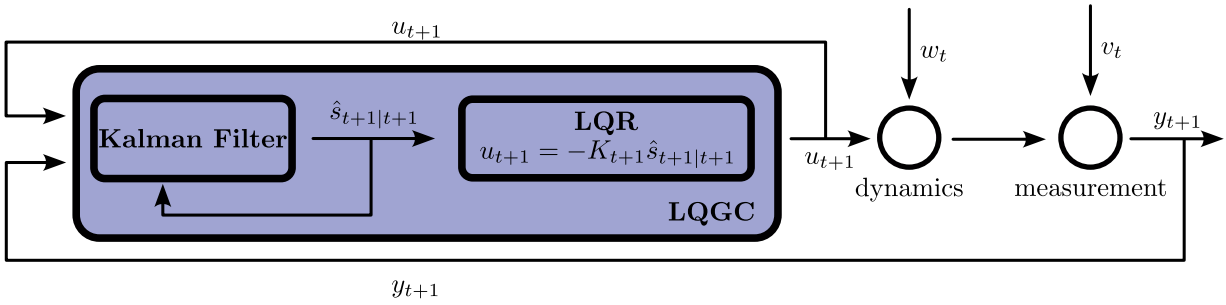


Figure 5.2 – **LQGC flowchart representation.** At each time step the Kalman filter determines the most probable state $\hat{s}_{t+t|t+1}$, which is passed to the LQR. Based on the estimate, the LQR applies a control input u_{t+1} to the system. The combination of the Kalman filter and LQR forms the LQGC (blue). The system evolves in time according to the dynamics, control input and process noise w_t. The following measurement under the influence of the measurement noise v_t is passed to the Kalman filter. Together with the last estimate and control input, the Kalman filter then calculates the next estimate.

Sec. 5.3.1, we are missing the additive Gaussian noise and assume that we are able to observe the state directly, without measurement.

The cost of the system is given by the quadratic function

$$J = \sum_t \left(s_t^T W_1 s_t + u_t^T W_2 u_t + 2 s_t^T W_e u_t \right), \tag{5.78}$$

where the matrices W_1, W_2, W_3 are the weighting matrices, which can be arbitrarily set in order to match the underlying problem. Here we can see that the LQR tries to solve an optimal control problem, as the strategy depends on the predefined cost, as shown in Sec. 6.1.3.

Based on the quadratic cost, the LQR returns the controlling law

$$u_t = -\mathcal{K}_t s_t, \tag{5.79}$$

which minimizes the cost function, where $\mathcal{K}$ is given by

$$\mathcal{K}_t = \left(R + B^T P B \right)^{-1} \left(B^T P A + N^T \right). \tag{5.80}$$

The matrix P is the solution to the *discrete time algebraic Riccati equation* **DARE** [179]

$$P_{k-1} = A^T P A - \left(A^T P B + N \right) \left(R + B^T P B \right)^{-1} \left(B^T P A + N^T \right) + Q \tag{5.81}$$

In the case that we have nonlinear dynamics and linear added controls given by

$$s_{t+1} = f\left(s_t\right) + B u_t, \tag{5.82}$$

we perform similar actions as in the EKF and approximate the nonlinearities using the first order extensions, therefore replacing the dynamic matrix A with the Jacobian

$$A_t = \frac{\partial f}{\partial s}|_{s_t}. \tag{5.83}$$

Now the Kalman filter and LQR can be combined to obtain the LQGC simply by replacing the state vectors used in Eq. (5.79) with the state estimates calculated in either Eq. (5.55) or Eq. (5.75), as illustrated in Fig. 5.2.

Chapter 6

Reinforcement learning quantum feedback control

In the previous chapter we formulated the system equations for a generic quantum feedback model. Now we want to use the similar structure between the feedback control loop and the reinforcement learning structure to reformulate the system as an RL environment. This approach allows the use of an RL-based controller that can learn the underlying dynamics and uncertainties without any prior knowledge. While the potential of reinforcement learning as a technique for quantum system control is significant, it is not a straightforward application. Specialized agents and training techniques are necessary to ensure their efficacy in quantum control. The development of these agents is a time consuming and complex process, even in the case of classical systems. For this reason, it is standard practice to first developed and test new agents on simple benchmark environments, allowing for fast research and better understanding. Our goal here is to design a simple quantum benchmark environment, creating the desired testing ground for quantum systems.

The quantum environment that we are going to implement is the **quantum cartpole**, a term coined by Zhikang T. Wang *et al.* in [180]. In this chapter, we will discuss the construction of this environment, both conceptually and numerically, based on the classical cartpole environment proposed by [107]. We will also examine the significance of weak measurements in this system and their interaction with it. This allows us to translate the quantum model into a classical model mimicking its behavior.

Afterwards we discuss the controllers which will be used for stabilization of the system. Namely the previously introduced LQGC as well as some RL based controllers. While for the RL controllers we have to discuss their explicit design and training regarding the environment, the LQGC has to be adapted to be applicable to a quantum system, as it is originally designed for a classical system.

We begin testing the environment by comparing controller performance on a linear classical system. Next, we compare the classical model with the quantum model for the same controllers and potentials. We then vary the potential, changing the system from linear to nonlinear, and benchmark the controllers against each other.

This is followed by an analysis of the control strategies employed by the controller, with a focus on extracting the strategy of RL-based controllers. Subsequently, a discussion of the results is presented. It should be noted that this chapter draws heavily on a previous publication by the author of this thesis [P2].

6.1 Quantum benchmark environment: The quantum cartpole

Our goal is to create a straightforward quantum environment in which agents can be easily created and tested. We draw inspiration from the most popular classical benchmark environment, the *cartpole environment* [107]. In this environment, a pole is positioned on top of a cart and tips towards the right or left side based on the gravitational potential. The objective is to maneuver the cart to the left and right to counteract the movement of the pole and balance it upright on the cart as shown in Fig. 6.1 a). Failure occurs when the pole surpasses a specific angular threshold.

We will use these simple ideas to create a quantum version of this, called the *quantum cartpole environment* [P1, 180]. In this environment, we intend to replicate the balance by replacing the pole with a free particle, which will be initialized as a Gaussian wavepacket

$$|\psi\rangle = \frac{1}{\sqrt{2\pi\sigma_{\text{sys}}}} \int dx\, e^{\frac{-(x-x_0)^2}{4\sigma_{\text{sys}}^2}} e^{-ixp}. \tag{6.1}$$

This wavefunction is then placed on top of an inverted potential $V(x)$ as shown in Fig. 6.1. As the system undergoes unitary dynamics governed by the Hamiltonian

$$\hat{H} = \frac{\hat{p}}{2m} + V(\hat{x}) \tag{6.2}$$

the wavepacket will move to either direction of the potential. As the wavefunction is defined over the whole position space, the termination condition differs from that of the classical cartpole. To ensure termination is still achieved even when the wavepacket has shifted far from the center, the condition is set to trigger as soon as at least 50% of the wavefunction's probability density has surpassed a specific threshold value x. To prevent the wavefunction from exceeding the threshold, active feedback control is sued based on the measured values of $\langle x\rangle$ and $\langle p\rangle$ of the wavefunction. This is accomplished through the use of weak measurements, as described in Sec. 5.2.1. Following the measurement results, a controlling algorithm, which may be either classical or machine learning-based, applies a force to the system in the form of a unitary "kick operator" u_F. This kick operator is implemented using a shift operator $e^{-iF\hat{x}}$, which adds momentum to the system. We have decided to impose a maximum force limit of $|F| < F_{\max}$ on the system to be closer to actual experimental setups.

There is an additional problem that arises from the quantum nature which is not observed in classical cases. Once the wavefunction is placed on the potential, it begins to delocalize, independent of any applied unitary force u_F. To counteract this problem, it is necessary to apply state reduction to the system, maintaining its shape over time. Fortunately, the weak measurement already has the ability to keep the wavefunction stable over time.

Under the condition that our wavefunction is in form of a Gaussian probability distribution, the post-measurement state follows from Eq. (5.10) to be the product of two Gaussian probability distribution which is again a Gaussian probability distribution. Assuming that $\lambda = 1$, the resulting variance is given as

$$\sigma_{\text{post}} = \left(\frac{\sigma_{\text{sys}}^2\sigma_{\text{anc}}^2}{\sigma_{\text{sys}}^2 + \sigma_{\text{anc}}^2}\right)^{1/2} \qquad \langle x\rangle_{\text{post}} = \frac{\langle x\rangle\sigma_{\text{anc}}^2 + m\sigma_{\text{sys}}^2}{\sigma_{\text{sys}}^2 + \sigma_{\text{anc}}^2}, \tag{6.3}$$

where σ_{sys} represents the initial variance of the system wavefunction and σ_{sys} is the width of the ancilla wavefunction. From this follows, that the resulting width has to be $\sigma_{post} \leq \sigma_{\text{sys}}$,

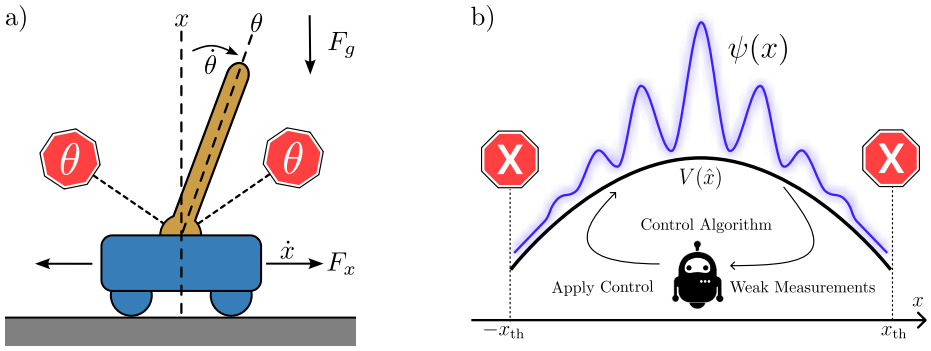


Figure 6.1 – **Cartpole environment for classical and quantum system.** For the classical cartpole, a pole is placed on top of a cart, which is movable in one dimension. Based on some random initial deviation, the pole will drop to one of the two sides. The observer has perfect information about the angle and velocity of the pole and can therefore apply a force on the cart to keep the pole in an upright position. In the quantum case, a wavefunction is placed on some inverted potential (a), with the goal to keep it inside the selected area $[-x_{\text{th}}, x_{\text{th}}]$ marked by the red signs. Once $> 50\%$ of the probability distribution move outside, the run is considered failed and reset. To avoid this periodic measurements are performed on the wavefunction with active control to stabilize the wavefunction inside.

demonstrating the state reduction in the position space. A similar derivation leading to the resulting measurement can be obtained by applying a momentum measurement

$$\sigma_{\text{post}} = 2\left(\frac{\frac{4}{\sigma_{\text{sys}}^2}\sigma_{\text{anc}}^2}{\frac{4}{\sigma_{\text{sys}}^2}+\sigma_{\text{anc}}^2}\right)^{-1/2} \qquad \langle p\rangle_{\text{post}} = \frac{\langle p\rangle\sigma_{\text{anc}}^2 + m\frac{4}{\sigma_{\text{sys}}^2}}{\frac{4}{\sigma_{\text{sys}}^2}+\sigma_{\text{anc}}^2}. \tag{6.4}$$

This now shows a state reduction in the momentum space and therefore $\sigma_{post} \geq \sigma_{\text{sys}}$ after the momentum measurement, thereby working against the state reduction of the position weak measurement. As the magnitude of the state reduction varies depending on the σ_{sys} and σ_{anc}, the state reduction of both measurements generally does not cancel itself out. The difference between the initials and post position width after sequentially applying both weak measurements to the wavefunction is shown in 6.2. The heatmap shows $\Delta\sigma_{\text{sys}} = \sigma_{\text{post}} - \sigma_{\text{sys}}$ depending on σ_{sys} and σ_{sys}. The white line marks the area, where the pre and post-measurement wavefunctions have the same width, therefore stabilizing the wavefunction to constant shape. Looking at the areas further away shows that the resulting state reduction moves the wavefunction closer to the white line, until it is stabilized. So depending on the applied potential, the stabilized shape of the wavefunction differs, but the wavefunction will be stopped from delocalzing over time. A more detailed derivation can be found in Ap. D.1.

While this describes the general function of the environment, we need to formulate it correctly in terms of a RL environment as defined in Sec. 2.3. The system's state is determined by the wavefunction $|\psi\rangle$, which is initialized at the center of the potential with a random momentum $\langle p\rangle_{\text{init}}$, sampled from a zero mean Gaussian distribution if the width $\sigma_p = 0.1$. The wavepacket is also initialized with a width of $\sigma_{\text{sys}} = 1.0$. The wavepacket forms the state $s(t)$ of the RL environment as it completely describes the system.

The interaction with the environment at every time step Δt consists of the following three parts:

- The unitary evolution based on the Hamiltonian from Eq. (6.2) for the time intercal Δt
- The sequential application of the weak position and momentum measurement, and the therefore resulting backaction on the system

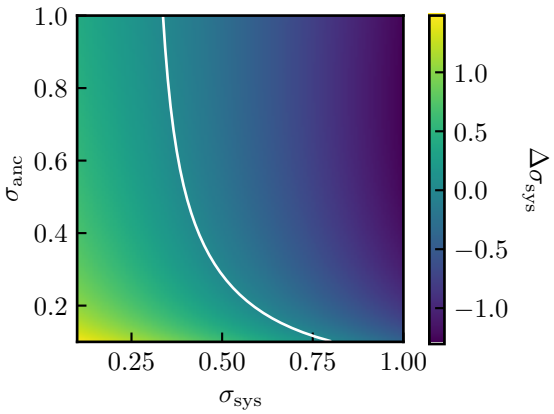


Figure 6.2 –
Weak measurement based state reduction. The heatmap demonstrates the change in the width of the wavefunction $\Delta\sigma_{\mathrm{sys}}$, when applying a position weak measurement followed by a momentum weak measurement. Both weak measurements use the same ancilla wavefunction with σ_{anc}, and the initial wavefunction has a width of σ_{sys}. The white line marks $\Delta\sigma_{\mathrm{sys}} = 0$.

- The application of a controlling force via the kick operator u_F

The force applied is determined by some control algorithm or RL agent, where we first have to provide some information about the system. While in RL often the state $s(t)$ is used as input information, which here is the wavefunction, in actual experimental setups this information is not available. Therefore we use the observation $o(t)$ provided by the weak measurements.

Here we also want to introduce another technique called *framestacking* [82]. This technique provides the option of performing multiple measurements to obtain mean values of the position and momentum observables. The idea behind this technique is that by performing multiple measurements, the observation passed to the agent appears less noisy, which would allow for an easier and more stable training process of the agent, but at the cost of a longer waiting time before an action can be taken.

In a system without measurement backaction, with the time step $dt \to 0$ between two measurements, performing multiple measurements of a noisy observable described by σ_{noise} is the same as performing a single measurement of the observable with

$$\sigma'_{\mathrm{noise}} = \frac{\sigma_{\mathrm{noise}}}{\sqrt{N}}, \tag{6.5}$$

where N_{meas} is the number of measurements. In actual applications this state is merely an approximation, as the system, and therefore the observables, change because of time evolution and backaction of the measurement. While this technique was taken from the RL world, it is also possible to see this in light of feedback control theory, as it can be viewed as a simple state estimation technique. Therefore, the observation passed to the control algorithm will be described via $o = (\overline{x}_{\mathrm{est}}, \overline{p}_{\mathrm{est}})$ as shown in Fig. 6.3. The action returned is the force used by the kick operator, and based on the state of the system a reward is returned to the agent. The exact shape of the reward will be discuss in detail in Sec. 6.1.3.

6.1.1 Classical surrogate model

As our goal is to create a benchmark environment for quantum systems, it is necessary to give a context for provided benchmarks. This is why we previously discussed the LQGC, as it is known to be optimal on the linear classical system. When applied to a quantum system, it can still achieve high performance [181], but the optimality statement is no longer preserved. Thus, we cannot accurately judge the performance based on the quantum version alone and

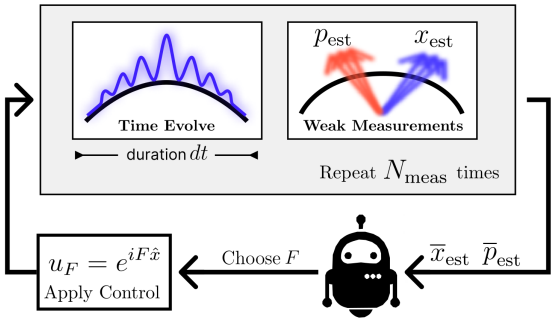


Figure 6.3 –
Quantum cartpole environment. In the actual simulation (b) this cycle is discretized in time steps dt, which starts with the time evolution of the wavefunction over the time dt. Then, the position and weak measurements are applied sequentially. This block is repeated N_{meas} times. Afterwards, the measurement means $\overline{x}_{meas}, \overline{p}_{meas}$ are passed to the state estimator, which in turn returns $\overline{x}_{\text{est}}$ and $\overline{p}_{\text{est}}$ to the control algorithms and an appropriate force F is chosen to apply a kick to the wavefunction. Then the cycle starts anew. Figures taken and resized from [P1].

have therefore decided to create a classical surrogate model for comparison, which behaves as closely as possible to quantum system.

The surrogate model can be described using a combination of the classical formulation for feedback control for linear Eq. (5.3) and nonlinear system Eq.5.1. The model is described by the equation

$$s_{t+1} = f(s_t) + Bu_t + w_t \tag{6.6}$$
$$y_t = Cs_t + v_t, \tag{6.7}$$

where $f(s_t)$ is determined by the chosen potential of the system $V(x)$ and can very well be a nonlinear function[1]. On the other hand the matrices B, C are fixed in the form

$$B = \begin{pmatrix} 0 & 0 \\ 0 & \Delta t \end{pmatrix} \qquad C = \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix}. \tag{6.8}$$

As the control matrix B is designed to mimic the kick operator, it only alters the momentum depending on the selected force. Likewise, the measurement matrix C produces the initial state multiplied by the factor λ, based on the weak measurement results specified in Eqs. 5.29 and 5.34. This approximation assumes that position measurement-induced backaction is negligible for momentum measurement.

Now both Wiener increments w_t and v_t have to be defined in a way to capture the measurement uncertainty of the weak measurement and the backaction of the system, while still being a zero mean Gaussian. As the backaction depends on the measurement, the Wiener increments relation is now given by

$$\mathbf{E}\left[\begin{pmatrix} w_t \\ v_t \end{pmatrix} \begin{pmatrix} w_{t'}^T & v_{t'}^T \end{pmatrix}\right] = \begin{pmatrix} Q & S \\ S^T & R \end{pmatrix} \delta_{tt'}\,, \tag{6.9}$$

where Q and R are the process and measurement covariance matrices and S is the crosscovariance matrix.

Starting with the measurement covariance matrix, the Wiener increment follows directly from Eq. (5.34), allowing it to be described as

$$R = \begin{bmatrix} \sigma_{\text{anc}}^2 & 0 \\ 0 & \sigma_{\text{anc}}^2 \end{bmatrix}. \tag{6.10}$$

[1] Actually, the quantum cartpole is more closely related to the inverted pendulum environment, but the quantum cartpole term was coined by [180]

As the process noise should replace the backaction and its actual value depends on the measurement result, we express the process noise Wiener increment with the measurement Wiener increment

$$dw = \begin{pmatrix} \langle dw_x \rangle \\ \langle dw_p \rangle \end{pmatrix} = \begin{pmatrix} m_{xx} & m_{xp} \\ m_{px} & m_{pp} \end{pmatrix} \begin{pmatrix} dv_x \\ dv_p \end{pmatrix} = M dv. \tag{6.11}$$

Based on Eq. (6.3) and Eq. (6.4), we know that the resulting backaction is also dependent on the width of the wavefunction. Depending on the potential and current position of the wavepacket inside it, σ_{sys} fluctuates around some fixed value. Therefore, when creating the surrogate model, we perform 10^6 measurements on the quantum system, recording the measurements outcomes $x_{\text{meas}}, p_{\text{meas}}$ as well as the resulting change in position and momentum dx, dp. From this we can approximate M by fitting the backaction as a linear combination of the measurement results with the factors $m_{i,j}$.

From this the process covariance and crosscovariance matrices follows as

$$Q = \begin{bmatrix} (m_{x,x} + m_{x,p})\,\sigma^2_{\text{anc}} & (m_{x,x}m_{x,p} + m_{p,x}m_{p,p})\,\sigma^2_{\text{anc}} \\ (m_{x,x}m_{x,p} + m_{p,x}m_{p,p})\,\sigma^2_{\text{anc}} & (m_{p,x} + m_{p,p})\,\sigma^2_{\text{anc}} \end{bmatrix} \tag{6.12}$$

$$S = \begin{bmatrix} m_{x,x}\sigma^2_{\text{anc}} & m_{p,x}\sigma^2_{\text{anc}} \\ m_{x,p}\sigma^2_{\text{anc}} & m_{p,p}\sigma^2_{\text{anc}} \end{bmatrix} \tag{6.13}$$

based on covariance calculations for linear combinations.

6.1.2 Numerical realization

The task is now the numerical implementation of the previously described system. Starting from the RL interaction loop, the dynamics will be discrete in time steps Δt. Every time step consists of three different parts, the time evolution of the wavefunction based on the potential, the weak measurement on the system and the feedback application on the system. It is possible to formulate this problem as QSDE as shown Eq. (5.43). However it is not only necessary to calculate the equation of motion for $\langle x \rangle$ and $\langle p \rangle$, but also for the covariances of the system C_{xx}, C_{xp}, C_{pp}, since the backaction and measurement results of the weak measurement depends on this. These derivations are non-trivial and depend heavily on the chosen potential. Since we want to create a benchmark environment, that can be used with an arbitrary potential, we will instead use some numerical algorithms to calculate the dynamics of the wavefunction.

The first step is the approximation of the dynamics based on the potential. The dynamics are guided by the Hamiltonian in Eq. (6.2), which leads to the Schrödinger equation

$$-i\hbar \frac{\partial}{\partial t}\psi = H\psi \tag{6.14}$$

from which the unitary time evolution follows

$$\psi(t + dt) = e^{-iHdt}\psi(t). \tag{6.15}$$

This can be viewed as an initial value problem of the form $\frac{dy}{dt} = f(t, y),\ y(t_0)$, which can be approximately solved using Runge-Kutta methods [182]. For this it is necessary to discretize the wavefunction in the space and time. This results in the second-order position derivatives

$$\nabla^2\psi(x_j, t^n) \approx \frac{\psi^n_{j+1} + \psi^n_{j-1} - 2\psi^n_j}{\Delta t^2}, \tag{6.16}$$

where we use the second-order numerical differentiation method, with j being the position index and n the time index.

Using the most widely known representation of the Runge-Kutta method, often referred to as **RK4**, the next time step is given by

$$\psi_j^{n+1} = \psi_j^n + \frac{\Delta t}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right), \tag{6.17}$$

with Δt being the time step size and k_i are defined as

$$\begin{aligned} k_1 &= f(t_n, \psi_n) \\ k_2 &= f(t_n + \frac{\Delta t}{2}, \psi_n + \Delta t \frac{k_1}{2}), \\ k_3 &= f(t_n + \frac{\Delta t}{2}, \psi_n + \Delta t \frac{k_2}{2}), \\ k_4 &= f(t_n + \Delta t, \psi_n + \Delta t k_3). \end{aligned}$$

Based on the Schrödinger equation Eq. (6.14) and the numerical differentiation Eq. (6.16) we can express the function f as

$$f(t_n, \psi_n) = \frac{i}{2m\hbar}\frac{\psi_{j+1}^n + \psi_{j-1}^n - 2\psi_j^n}{\Delta t^2} - \frac{i}{\hbar} V_n \psi_n^t \tag{6.18}$$

and are able to numerically simulate the time evolution.

The Runge-Kutta method is a fourth order method, meaning that the local truncation error is of order $\mathcal{O}\left(\Delta t^5\right)$ with the total error being of order $\mathcal{O}\left(\Delta t^5\right)$ [182]. By selecting a reduced time step size for the Runge-Kutta simulation, which is a fraction of the environmental time step $\Delta t_{\mathrm{RK4}} = \frac{1}{N}\Delta t_{\mathrm{env}}$ where $N \in \mathbb{Z}$, we can reduce the numerical uncertainty, but at the cost of increased computation time.

The application of the kick operator can be done exactly, as the unitary kick operator simplifies to a diagonal matrix. Therefore, it is possible to perform this evolution as a simple vector-vector multiplication.

The calculation of a weak measurement is performed in two steps. First, we have to determine the measurement result, which can be sampled as a Gaussian distribution $\mathcal{N}(\lambda\langle A\rangle, 2\sigma_{\mathrm{anc}}^2)$, based on Eq. (5.29) and Eq. (5.34), assuming that λ is small so that λ^2 is negligible. Since σ_{anc} is known and constant in time, as the ancilla wavefunction has to be reinitialized for every measurement, only $\langle A\rangle$ has to be determined. In the numerical simulation we have access to the complete wavefunction. Therefore we can calculate the expectation value of the position via

$$\langle x\rangle = \int dx \psi^* x \psi \tag{6.19}$$

$$\approx \sum \Delta x \psi^* x \psi, \tag{6.20}$$

with Δx being the step size used to discretize the position space. The same can be done for the momentum as

$$\langle p\rangle = \int dx \psi^* \left(-i\hbar \frac{\partial}{\partial x}\psi\right) \tag{6.21}$$

$$\approx \sum -i\hbar \Delta x \psi^* \psi'. \tag{6.22}$$

After sampling of the measurement result, we can calculate the post-measurement state of the wavefunction according to Eq. (5.10). Using that λ is small we perform a series expansion of 5.17

$$U = I \otimes I - i\lambda H - \frac{1}{2}\lambda^2 H^2 + O\left(\lambda^3\right) \tag{6.23}$$

$$\approx I \otimes I - i\lambda A \otimes p - \frac{1}{2}\lambda^2 A^2 \otimes p^2, \tag{6.24}$$

which is cut off after λ^2. Here the operator A represents either the position or momentum operator. Performing the projective measurement on the ancilla wavefunction will return the conditional state

$$|\Psi_q\rangle = \frac{I\langle q|\phi\rangle - i\lambda A\langle q|p|\phi\rangle - \frac{1}{2}\lambda^2 A^2\langle q|p^2|\phi\rangle}{\mathcal{N}}|\psi\rangle \otimes |q\rangle, \tag{6.25}$$

where $\mathcal{N}$ is the normalization. The post-measurement state of the wavefunction is returned by tracing of combined state, returning

$$|\psi_q\rangle = \frac{I\langle q|\phi\rangle - i\lambda A\langle q|p|\phi\rangle - \frac{1}{2}\lambda^2 A^2\langle q|p^2|\phi\rangle}{\mathcal{N}}|\psi\rangle. \tag{6.26}$$

We can easily calculate this expression, as the terms $\langle q|\phi\rangle$, $\langle q|p|\phi\rangle$ and $\langle q|p^2|\phi\rangle$ simply correspond to the different derivations of the ancilla wavefunction $\phi\left(q\right)$, $-i\hbar\phi'\left(q\right)$ and $-\hbar^2\phi''\left(q\right)$. These functions can be determined analytically beforehand based on Eq. (5.16). Similary, the two remaining terms $A|\psi\rangle$ and $A^2|\psi\rangle$ can be calculated via vector multiplication or numerical differentiation, depending whether the position or momentum weak measurements are used.

6.1.3 Control strategies

After defining the quantum environment as well as the classical surrogate, it is still necessary to define the control algorithms. The control algorithms can be generally differentiated based on their approach, thereby separating them into classical and RL-based algorithms, as well as their area of application, which includes state estimation and feedback control. This structure allows us to combine different state estimation techniques with different feedback control.

Classical control algorithm For classical algorithm controller, we use the LQGC algorithm described in Sec. 5.3, which is made up from the Kalman filter as well as the LQR.

The Kalman filter provides the state estimation part and in the case of the surrogate model, can be applied following the steps provided in Sec. 5.3.1, as all the dynamics and covariances are known. However, it is necessary to adopt the same step correlation extensions provided in Sec. 5.3.2 and, depending on the chosen potential, use either the Kalman or Extended Kalman model. In the case of the quantum model, we assume that the classical surrogate model is an accurate description of the quantum model and therefore apply the Kalman filter based on the surrogate dynamics.

A similar procedure is used for the LQR. In the surrogate model, the LQR can be implemented based on its description in Sec. 5.3.4, as all of the necessary information is known. In the quantum case, we apply the LQR based on the Kalman filter. This leaves us with the quadratic cost function of the LQR, which we want to model after the Hamiltonian, so that the LQR minimizes $|V(x)| + T$, with T being the kinetic energy. This means that in the beginning, we

set the weights W_2, W_3 to zero, as we do not want to penalize the application of the controlling force. This leaves only W_1, which we define as

$$W_1 = \begin{bmatrix} \frac{|V(x)|}{x^2}|_{x=\overline{x}_{meas}} & 0 \\ 0 & \frac{1}{2m} \end{bmatrix}, \tag{6.27}$$

where the potential is evaluated based on the latest measurement.

Reinforcement learning control As the LQGC optimal only in the case of classical linear system, we want to create a RL-based control algorithm that can achieve comparable or better performance. The main advantage of RL control would be the model-free nature. While LQGC requires extensive knowledge of system dynamics and noise sources, RL can be trained solely on measurement results while observing the runtime of the wavefunction.

Taking inspiration from the design of the LQGC, we will split the RL controller into two parts. The first part is the *reinforcement learning controller* **RLC**, which takes on the task to convert state estimates into a controlling force. The other part is the *reinforcement learning estimator* **RLE**, which calculates the most probable state based on the measurements. For both agents we will utilize the **PPO** algorithm, introduced in Sec. 2.3.4.

The RLC is trained with the goal of giving the most optimal control force based on the input, which can either be the raw measurement results $\overline{x}_{\text{meas}}$, $\overline{p}_{\text{meas}}$ or be the estimate giving by some state estimation technique $\overline{x}_{\text{est}}$ and $\overline{p}_{\text{est}}$. The output of the agent is the controlling force u_F in the range $[-F_{\max}, F_{\max}]$. Both the observation and action space are defined as continuous spaces. In order to allow the agent to find a control strategy without experiencing a bias based on the implementation, we use a reward function which punishes if the wavefunction is pushed outside the thresholds without offering any rewards for keeping the wavefunction inside a certain area. The reward function is given as

$$r_t = \begin{cases} 0 & \text{if } \int_{-x_{\text{th}}}^{x_{\text{th}}} dx |\psi(x)| > 0.5 \\ -1 & \text{else} \end{cases}. \tag{6.28}$$

The disadvantage of this is an increased difficulty in training the agents, since it is necessary to run multiple completed runs to get sensible statistics for the agent. Consequently, training times are considerably increased for environments where the wavefunction survives for an extended period of time. Additionally, the agents are more likely to experience performance degradation and get stuck in local minima, as discussed in Sec. 2.3.5.

Now switching to the RLE, whose goal is to mimic the Kalman filter by copying its input and output. Therefore, the observation space contains the state estimation $\overline{x}^{\text{est,t}_1}, \overline{p}^{\text{est,t}-1}$ from the previous time steps, the current measurement results $\overline{x}_{\text{meas}}$ and $\overline{p}_{\text{meas}}$ and the last used control force F_{t-1}. In return, the output is given by a new state estimation $\overline{x}_{\text{est,t}}, \overline{p}_{\text{est,t}}$. Compared to the LQGC, the RLE has the increased difficulty of learning the system dynamics, noise covariances, as well as the Kalman filter functionalities.

The reward function chosen for this problem is

$$r_t = -\sum (y_{\text{meas}} - s_{\text{est}}), \tag{6.29}$$

based on [183].

The results of training and testing the RLE in combination with the RLC can be seen in Fig. 6.4 b). Showcasing the performance depending on the N_{meas}, it is visible that the RLC + RLE has a similar behavior as the LQR + Kalman, as the maximum performance is reached

at N_{emas}. This shows that the RLE is able to learn some state estimation, but since there is a performance difference of $\approx 50\%$ to the LQR + Kalman, the RLE is not able to reliable predict the most probable state.

More details about the training process as well as the used hyper parameters can be found in D.2.

6.2 Controlling the quantum cartpole

Applying different controlling algorithms, we try to quantify their performance via the average termination time $t_{\text{termination}}$. This observable is determined by simulating N runs and recording the time step when the termination condition is fulfilled. Since each individual run is independent from each other, the average termination time is simply expressed by a mean and its standard error

$$\bar{t}_{\text{termination}} = \frac{1}{N}\sum_{N} t_{\text{termination}} \qquad \Delta t = \frac{1}{N}\sqrt{\sum_{N}\left(t_{\text{termination}} - \bar{t}_{\text{termination}}\right)^2}. \tag{6.30}$$

In the case of RL agents, we are able to observe a variation in the performance between agents, overshadowing the errors of the simulation. In order to minimize these training induced errors, we train a number of different agents on the same set of environmental parameters and training hyperparameters and only pick the M best performing agents to calculate the performance of the RL controller. It is important to mention that the training of these agents is performed completely independent of each other. This follows the argument that even if training might be volatile and produces agents with a wide range of performance, only the best performing agent will actually be deployed later and the rest will be discarded. Because of this, we have to amend the calculation of the average termination time to

$$\bar{t}_{\text{termination}} = \frac{1}{M}\sum_{i=1}^{M} \bar{t}_m \qquad \Delta\bar{t}_{\text{termination}} = \frac{1}{M}\sqrt{\sum_{M}\frac{\sigma_m^2}{N}}. \tag{6.31}$$

In the following, all results were created using this standard set of environmental parameters as listed in Tab. 6.1, unless otherwise specified.

6.2.1 Linear surrogate system

Using the surrogate model, we want to run some initial test to evaluate the performance of the controllers depending on the measurement noise level as well as to investigate the influence of the framestacking technique.

For this, we will test different control algorithms on a linear system with an inverse quadratic potential $V(x) = -\frac{k}{2}x^2$ with $k = \pi$. For the first test, we compare the performance of the LQR against the RLC for different values of σ_{meas} and vary the number of weak measurements N_{meas} performed per action. The results are shown in Fig. 6.4 a), where every single datapoint

Δt	m	$\hbar$	σ_{sys}	$\langle p^2\rangle_{\text{init}}$	x_{th}	$F_{\max}$
$0.01/\pi$	$\frac{1}{\pi}$	1	1.0	0.1	8	8π

Table 6.1 – **Quantum cartpole environmental parameters.** This set of parameters is chosen for comparability to [180] and all units were set to 1.

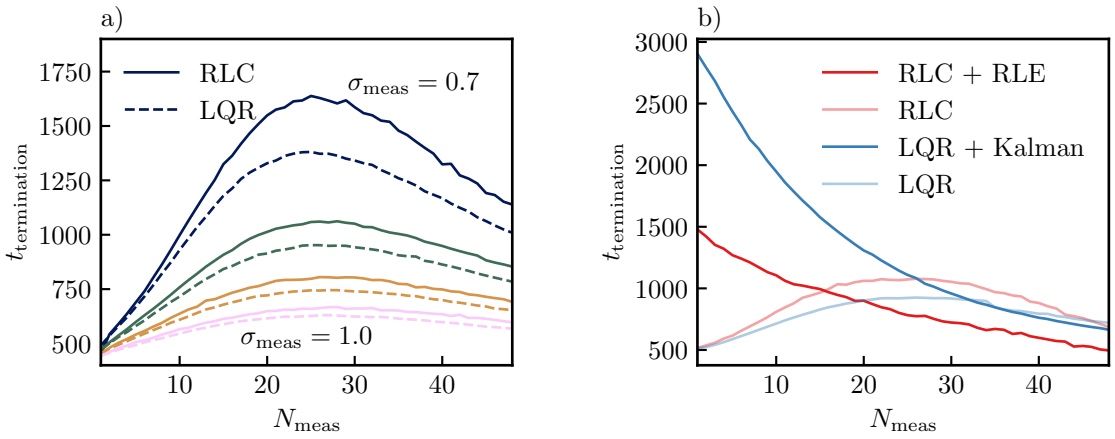


Figure 6.4 – **Importance of state estimation.** Testing the LQR (dashed) against the RLC (solid) on the classical surrogate model with measurement noise varying from $\sigma_{\text{noise}} = 0.7$ to $\sigma_{\text{noise}} = 1.0$ (a) by comparing the average runtime $t_{\text{termination}}$ against the number of measurements performed to obtain the position and momentum observables. Fixing the measurement noise to $\sigma_{\text{noise}} = 0.8$ (b) and adding state estimation in form of the Kalman filter (blue) and RLE (red) to the LQR and RLC, respectively. All four different combinations are compared based on the average termination time against the number of measurements. This figure was taken and modified from [P1].

was determined using 10^5 individual runs of the surrogate model and, in case of RL based controllers, the 10 best performing agents were used to determine the results.

Starting with the LQR, one can see based on the $t_{\text{termination}}$ at $N_{\text{meas}} = 1$ that the algorithm on its own struggles to stabilize the wavefunction for a meaningful amount of time. Increasing the number of measurements according to the framestacking technique allows the LQR to increase the performance by more than a factor of three, reaching a maximum at $N_{\text{meas}} = 25$. This shows the importance of state estimation when process noise is present in the system. Increasing the number of measurements further the performance starts to drop again, which can be seen as a consequence of the increasing delay between taking controlling actions to stabilize the system. Furthermore this behavior can be seen independently of the level of measurement noise applied to the system, where, as expected, the performance drops when the measurement noise is increased.

Looking at the performance of the RLC for the same environment, it follows the same performance behavior as the LQR, but is able to increase the top performance by up to $\approx 20\%$ for $\sigma_{\text{meas}} = 0.7$. This increase in performance is most likely a result of the framestacking returning not the most probable state, but only an approximation of it. The LQR is designed to work with the most probable state, whereas the RLC can adopt to the suboptimal input given.

Now we also want to include the Kalman filter as well as the RLE on the same setting to compare the performance when proper state estimation techniques are used. The results are shown for $\sigma_{\text{meas}} = 0.8$ in Fig. 6.4 b), where the previous results for the LQR and RLC are included for comparison. Starting again with the classical control algorithm, the LQR + Kalman filter, it is evident that the peak performance has increased by a factor of ≈ 3 compared to the LQR, and as well that the maximum has shifted to $N_{\text{meas}} = 1$. Here the LQGC is known to be an optimal controller, therefore it is expected to achieve the highest performance at that point. When increasing the number of measurements, the same statements regarding LQR also apply to the LQR + Kalman filter. The performance drops due to the delay between actions as well as the approximation introduced by performing multiple measurements. Therefore, it

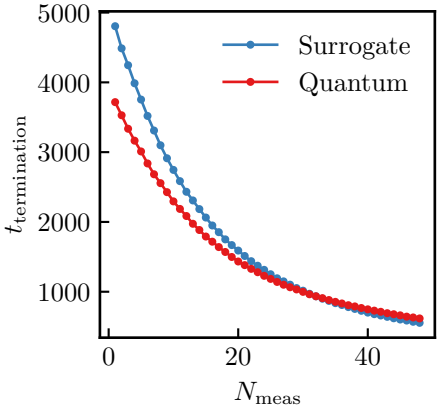


Figure 6.5 –
Comparison of the surrogate and quantum environment. The plots show the $t_{\text{termination}}$ of the LQR + Kalman controller for different numbers of measurements. The controller was applied for both the classical surrogate environment (blue) and the quantum environment (red).

is important to note that the LQR + Kalman filter is *not* optimal for $N_{\text{meas}} \geq 2$. This is also shown by the fact that for $N_{\text{meas}} > 30$ other controlling algorithms are able to outperform it.

In the case of the RL variant of the LQGC, the RLE + RLC controller also shows a maximum at $N_{\text{meas}} = 1$, but only achieves approximately 50% performance compared to LQGC at that point, while also staying below the LQGC at all times.

6.2.2 Linear quantum system

Now we want to turn our attention to the quantum environment and first establish the comparability between the quantum case and the surrogate model. For this, we are going to continue to use the inverted quadratic potential with $\sigma_{\text{anc}} = \sigma_{\text{meas}} = 0.7$ specifying the width of the ancilla wavefunction and the measurement noise for both systems, and apply LQGC as the control algorithm. The decision to use LQGC is made to eliminate the possibility of performance difference based on agent training. In Fig. 6.5, the LQGC algorithm's performance on the surrogate and quantum systems can be observed. The LQGC is able to demonstrate a similar performance on the quantum system as it does on the surrogate models, by exhibiting a maximum at $N_{\text{meas}} = 1$, followed by a continuous decline of performance with increasing measurements. This suggest that the surrogate model is a good approximation to the quantum cartpole. Still, because of the inherently differences between the classical and quantum environment discrepancies in performance can be observed, and therefore for the quantum system we lose the statement that LQGC is optimal, even at $N_{\text{meas}} = 1$.

After showcasing the ability of LQGC to still work in the quantum environment, we also want to quantify the behavior of the RL based model, namely the RLC and RLC + RLE. For this, we determine the $t_{\text{termination}}$ for the inverted quadratic potential for varying N_{meas} and σ_{anc}. As we now perform weak measurements in the quantum system, the measurement noise and measurement strength are both controlled by σ_{meas}. In Fig. 6.6 (a), the results for the RLC are presented in the form of a heatmap, where every point represent 10^5 runs of the quantum environment, equally spread out on the ten best performing agents. Similar to the surrogate model we can observe the tradeoff between performing multiple measurements and waiting to long to take an action. Additionally, it shows how the control of the system becomes unfeasible, if the measurement is too weak, as for $\sigma_{\text{anc}} = 1.0$ seemingly no tradeoff can be observed and the termination time converges against the same value for all measurement numbers. By adding an estimator in form of the RLE, Fig. 6.6 b) shows once again that the maximum shifts to $N_{\text{meas}} = 1$, similar to the surrogate model. This reinforces the statement that the surrogate

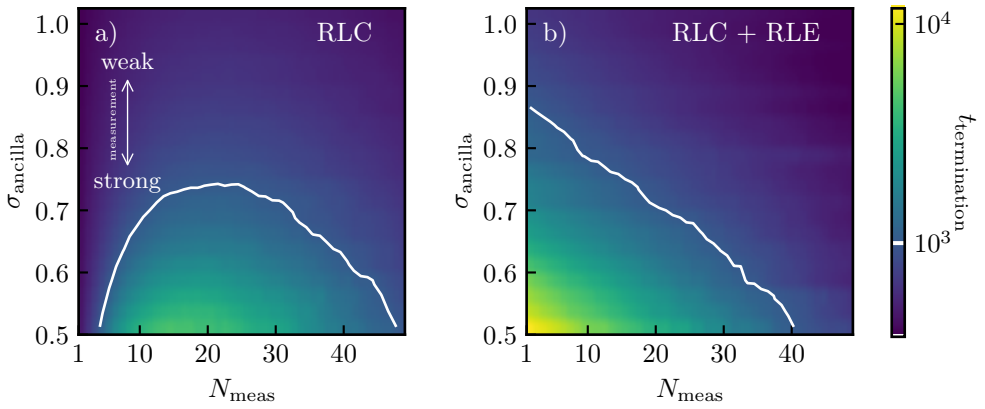


Figure 6.6 – **State estimation on quantum environment.** The figures show the performance of the RLC (a) and RLC + RLE (b) control algorithms tested on the quantum system with an inverted quadratic potential. The different simulation runs are varied in numbers of measurements and width of the ancilla wavefunction σ_{ancilla}, thereby changing the strength of the measurements. The white line indicates an average run time of $t_{\text{termination}} = 10^3$. The plot was taken and resized from [P1].

model is a good approximation of the quantum environment.

6.2.3 Potential variation

Up to this point, only linear systems with the inverse quadratic potential have been examined, but now we want to examine nonlinear systems by varying the used potential. The potentials we now additionally consider are

- A cosine potential $V(\hat{x}) = k_1 \left(\cos\left(\pi\hat{x}/k_2\right) - 1\right)$
- An inverted quartic potential $V\left(\hat{x}\right) = -k\hat{x}^4$,

where we use $k1 = 67$ and $k_2 = 12$ for the cosine potential and $k = \frac{\pi}{100}$ for the quartic potential. Fig. 6.7 shows these potentials together with the inverted quadratic potential. The quadratic and quartic potential use the definition introduced in [180].

Comparing the potential one can see that the cosine potential was chosen to be similar to the inverted quadratic potential, only differing by small margins inside the $[-x_{\text{th}}, x_{\text{th}}]$ and terminating at the same level as the quadratic potential. The inverted quartic potential on the other hand shows a completely different course and will be used as a stress test for the control algorithms to deal with nonlinear potentials.

Using these three potentials, we test the different control algorithms. Compared to previous tests we now only consider control algorithms with the state estimation technique, so excluding the pure LQR and RLC algorithms. Instead, we now also allow for mixing the classical control algorithms with the RL techniques. These results can be seen in Fig. 6.8, where the performance of 3 different controllers are shown, compared to the performance of the LQR + Kalman controller. The investigated controllers are the RLC + RLC, LQR + RLE and RLC + Kalman filter. The raw measurement data as well as the benchmark measurement on the surrogate model can be found in Ap. D.3.

Starting with the inverted quadratic potential in panel a), one can see that the RLC + Kalman controller is able to achieve a performance equal to the LQR + Kalman filter. This is a notable result, as the LQGC is not guaranteed to achieve optimal performance in the quantum case, but this indicates that LQGC is close to the optimal control case as the RLC

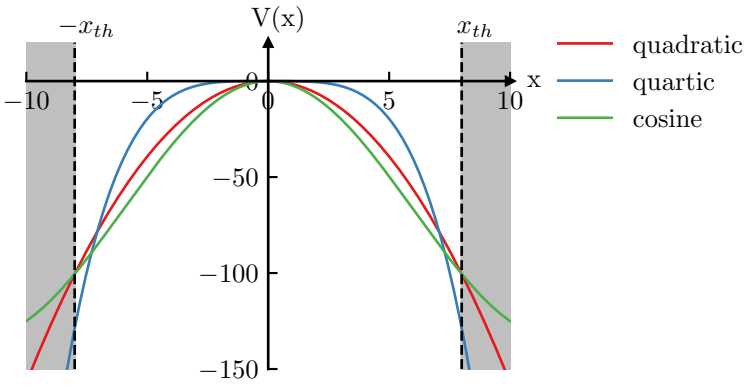


Figure 6.7 – **The different potentials** used in benchmarking the controllers. The inverted quadratic (blue) is used to benchmark the controllers for linear system and the cosine and inverted quartic (green, red) are used for nonlinear systems. At the left and right side of the plot, the threshold values are indicated. This plot was taken and resized from [P1].

+ Kalman filter converges against the same value. Looking at the other controllers, the pure RL based controller (RLC + RLE) is not able to catch up to the performance of the LQGC only achieving $\approx 50\%$ of the LQGC at $N_{\text{meas}} = 1$. Only at $N_{\text{meas}} \approx 40$, the controller is able to converge to similar performance of the LQGC and even surpassing it in some instances. These results reflect the previously seen results on the surrogate model in Fig. 6.4 b). The RLE + LQR controller consistently underperforms and is unable to catch up with the other controllers, demonstrating a poor synergy between the state estimation provided by the RLE and the LQR.

Going to the cosine potential in panel b), we now consider a nonlinear system. We can observe that the RLC + Kalman is now able to outperform the LQGC by a small margin. As both controllers utilize the Kalman filter for state estimation, this suggests that the LQR controller has lost some of its performance because of the nonlinearity, while the RLC is able to adapt to it. Similar the RLE + RLC is able to close the gap in performance, showcasing $\approx 70\%$ of the performance of the LQGC. Because of the machine learning nature of the agents, it is difficult to determine, to what extent the performance increase can be attributed to the RLC or RLE.

The advantage of the RL becomes clear in panel c), where the inverted quartic case is displayed. Now every single controller combination outperforms the LQGC controller for measurements up to $N_{\text{meas}} = 20$. The combination of the RLC + Kalman still reigns supreme achieving an increase of $\approx 60\%$ to the LQGC, while the RLC + RLE is able to achieve a performance only slightly below it. Even the LQR + RLE is able to increase the performance by $\approx 50\%$ compared to the LQGC.

Transfer learning In addition to the mixed classical and RL-based controller, we are also investigating the use of controllers trained on the *classical surrogate* and then applied to the quantum model. The results are presented alongside the other controllers in Figure 6.8. It is evident that the controllers achieve comparable performance to these trained directly on the quantum environment. This further demonstrates that the surrogate model is a reliable approximation of the quantum environment, even for nonlinear systems.

6.2.4 Control strategy interpretation

To further elucidate the control strategies used by the control algorithms, we investigate the output of RLE based on the measurement, as well the distributions of position $\langle x \rangle$ and momentum $\langle p \rangle$ expectation values.

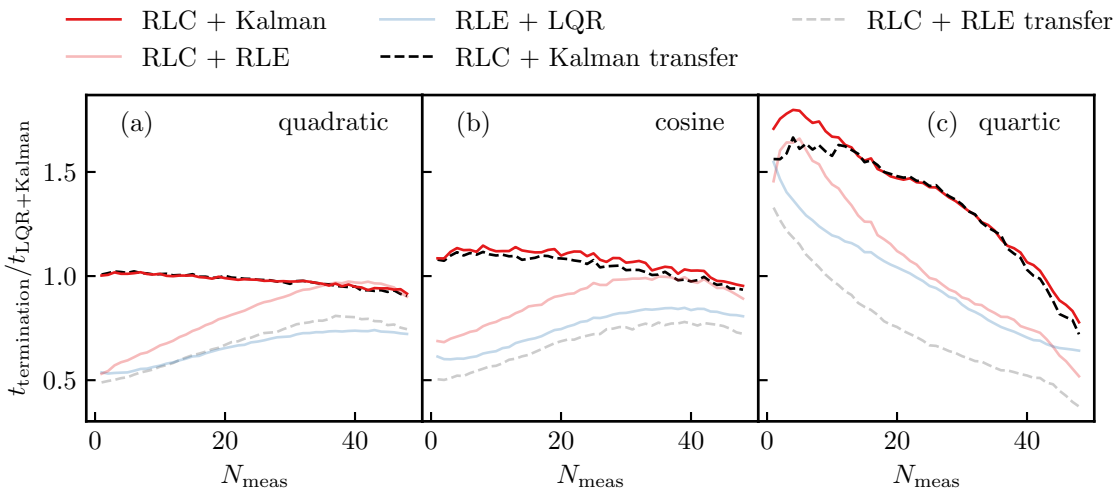


Figure 6.8 – **Improvement rate** of different control algorithm performances compared to the LQR + Kalman controller on the quantum environment. These control algorithm include three different combinations of RL based controllers, namely the RLC + Kalman (solid red), RLC + RLE (transparent red) and LQR + RLE (transparent blue). Additionally, some transfer learning versions (dashed) of RLC + Kalman (black) and RLC + RLE (transparent black) are included, where the RL agents were trained on the classical surrogate and then applied it to the quantum system. The investigated potentials include the inverted quadratic (a), cosine (b) and inverted quartic (c) potential. This plot was taken and resized from [P1].

Understanding what an ANN has learned is usually a difficult task. However, interpreting the ANN of the RLC is comparatively easier since the input space is two-dimensional, while the output space is only one-dimensional. This allows us to map all possible outcomes based on the input to a heat map, as shown in Fig. 6.9. As our potentials are all symmetric we can assume that an optimal control strategy can be described as an odd function and therefore has the property of

$$F(-\overline{x}_{\text{est}}, -\overline{p}_{\text{est}}) = -F(\overline{x}_{\text{est}}, \overline{p}_{\text{est}}). \tag{6.32}$$

If the agents have learned an optimal strategy, this property should be reflected in the ANN.

We have chosen the top performing RLC agents for each of the three potential, trained for $N_{\text{meas}} = 1$ and fed them state estimations between $[-2, 2]$ for both the $\overline{x}_{\text{est}}$ and $\overline{x}_{\text{est}}$. The first agent was trained using the inverted quadratic potential, and it is evident that it attempts to return the wave packet to the center of the potential when it is too far away or when the momentum is too high. This behavior is indicated by the yellow or dark blue areas where the agent applies the maximum force allowed. More interesting are the areas, where the agent chooses $F = 0$. We can see that this is the case when $\overline{x}_{est} \approx \overline{p}_{\text{est}}$, so showing a behavior of always counter acting the highest input value. Therefore, it nearly fulfills the condition in Eq. (6.32).

A different behavior can be seen for panel (b) and (c), where agents for the cosine and inverted quartic potentials are shown. For both agents the $F = 0$ line is pushed far away from the origin, so that $F(0, 0) = F_{\text{max}}$. This result is counter-intuitive as it suggests that the agent is intentionally pushing the wavepacket away from the center. Instead, it seems like it tries to balance the wavepacket on the side of the potential. This assumption is bolstered by the fact that the agent b) and c) show the same behavior only on different sides of the potential. Because of the symmetry of the potentials, there is no incentive for the agents to stabilize the wavepacket on one side over the other. Therefore, the chosen side only depends on the random

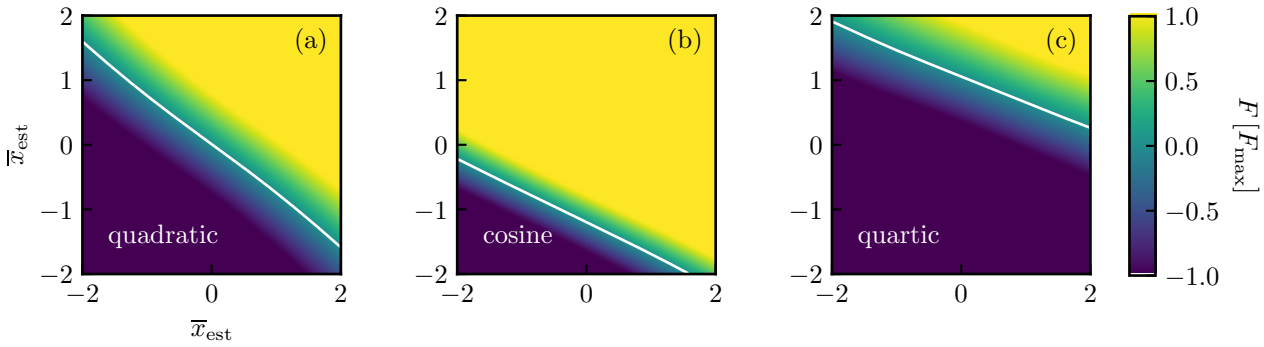


Figure 6.9 – **RLC output.** Output of RLC agent trained on the surrogate model with an inverted quadratic potential for $N_{\mathrm{meas}} = 1$. The agent is fed measurement values between $[-1, 1]$ for both the position and momentum. For each input pair, the corresponding output is visualized in the heatmap, showcasing a seeming linear mapping. The white line indicates the $F = 0$.

symmetry breaking in the training.

In the case of the RLE or the combination of the RLC + RLE, we cannot use the same technique to interpret what the agents have learned. Instead, we will try to get some indirect hint about it by looking at the distribution of $\langle x \rangle$ and $\langle p \rangle$ over time. The distributions were taken by collecting the position and momentum of the wavefunction over 10^6 time steps, but in order to avoid measurement artifacts from the initialization of the wavefunctions, only data from $t = 300$ and onward were taken.

The measured distributions are shown in Fig. 6.10 for the RLC + RLE as well as LQGC for comparison. For the LQGC, the distributions are symmetric and centered around 0 for both $\langle x \rangle$ and $\langle p \rangle$. This shows that the controller tries to balance the wavefunction in the middle of the potential. Comparing the quadratic and cosine potentials, the distributions are similar, but the cosine distributions have a greater width. This is most likely due to the cosine potential flattening near the threshold, allowing the wave packet to be kept near the threshold for longer. The quartic potential has by far the sharpest distribution, showing that it is unable to recover the wavepacket, once it approaches thresholds. Especially the potential distribution shows that once the wavepacket picks up to much momentum, the run will be terminated shortly after.

Now looking at the distributions of the full reinforcement learning case, the RLE + RLC controller, we can firstly see that the position distributions are not centered around 0 nor are they symmetric. As the agents are not forced to learn this behavior based on their received rewards, this, combined with the probabilistic nature of the training, can result in the agents learning to stabilize the wavepackets at any point. This can be seen especially for the quadratic and quartic potentials, where the distributions have their peak left and right from the center. The momentum distributions, on the other hand, are seemingly symmetric and centered around zero. This shows that the agents are actually trying to stabilize the wavepacket and not just pushing it outside in one preferred direction. The cosine potential also shows a unique behavior in the position distributions as is has a broad plateau instead of a clear peak. This indicates that the controller can stabilize the wavepacket far away from the center.

6.3 Summary

In this chapter we have designed a quantum benchmark environment that allows for the rapid development of RL agents that can be used for quantum feedback control, while at the same

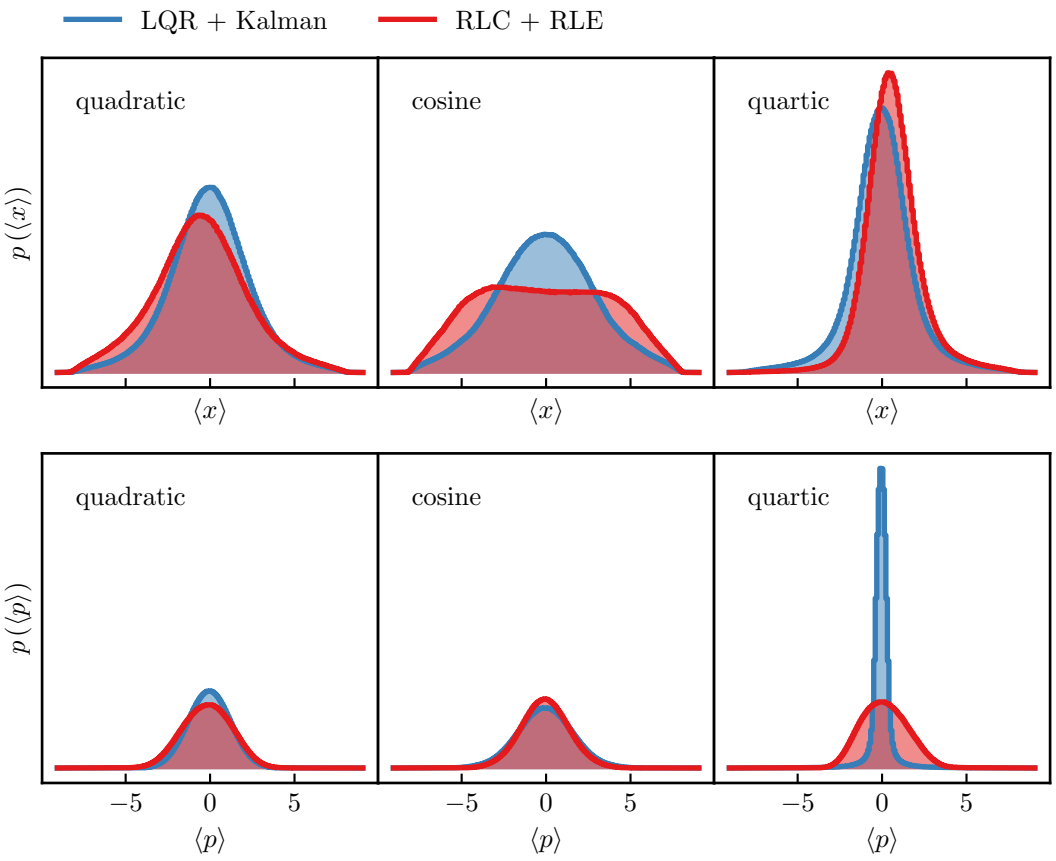


Figure 6.10 – **Distribution of position and momentum of the stabilized wavepacket** on the quantum system for the three different potentials: (I) **quadratic**, (II) **cosine**, (III) **quartic**. The position $\langle x\rangle$ (top row) and momentum $\langle p\rangle$ (bottom row) are traced over 10^6 time steps for the LQR + Kalman (red) and RLC + RLE (red) controllers and converted into histogram of the probability $p(\langle x\rangle)$. This plot was taken and modified from [P1].

time providing flexibility in the quantum problem by allowing any potential to be used for testing.

We have started this chapter with the discussion of the implementation of the quantum cartpole environment as an example of generic quantum feedback control. There we have introduced the numerical approximation for applying sequential weak position and momentum measurements to the system as well as the approximation for the time evolution and control input. At the same time we have also discussed the development of a classical surrogate for comparison. This allows the use of algorithmic control strategies developed for classical systems to set some RL independent benchmarks. We have tested these classical control algorithms as well as reinforcement learning based controllers on both environments as well as different potentials. From this we were able to obtain three notable results.

The first result is that it is possible to replace classical control algorithms in a quantum system, using RL based control algorithms. In the case of the RLC + Kalman filter controller, it was able to successfully replace the LQR algorithm, achieving the same or better performance for every potential. Even the pure RL based controller (RLC + RLE) was able to demonstrate at least 50% of the performance of the LQGC or even surpass it in the case of the quartic potential by finding new novel strategies as shown in Fig. 6.10. This is particularly remarkable when

considering that the RL controller only had access to information based on weak measurements, as opposed to the complete dynamics and noise covariances of the system, which most likely would not be available in an experiment.

The second result is that we were able to create a surrogate model which is able to faithfully mimic the quantum model. This has allowed us to train RL agents on the classical system and then transfer them to the quantum system while preserving their ability to control the system. This can be used to improve and speed up the training of the RL agents by training on the surrogate environment in the first step and then performing a final training on the quantum environment.

The third result is that using a frame stacking technique in the form of performing multiple measurements can be helpful in the training of the agents and can be used as a primitive state estimator which works without knowledge about the noise covariances. Although, if this knowledge exists, using an estimator utilizing this information causes the framestacking to lose its advantage.

Chapter 7

Summary and outlook

In this thesis, we have investigated the use of quantum feedback control to stabilize quantum systems over extended periods of time using quantum error correction as an example. As quantum error correction is necessary to achieve fault-tolerant computing, it is important to attain control strategies that are naturally described by the field of quantum feedback control.

Using machine learning as the basis to obtain these control strategies, we present numerical results demonstrating the ability of ML to learn control strategies which are able to perform comparably or better in terms of precision, speed, and robustness compared to algorithmic decoders.

Our first study focused on the problem of quantum error correction, where the idea of a machine learning decoder has been established and shown to achieve high accuracy decoding. However, a common problem in these cases is that ML approaches have difficulty dealing with large input and output spaces of the decoding problem, leading to either training failures for large code distances [33, 79] or an exponential increase in decoding time [80]. By designing a hierarchical decoder, which uses a neural decoder in the first step and a classical decoding algorithm in the second step, we were able to demonstrate an application of the ML based decoding for code distances up to $d = 255$ while having an almost linear scaling of the computational time.

Studies on the decoding accuracy of the hierarchical decoder revealed improved performance on both the toric code and RSC. The increase in performance on the latter was established even at code distances as low as $d = 7$, where early tests of active error correction on RSC implemented on actual hardware [52, 184] were conducted. Upon studying the behavior of the hierarchical decoder, it has been revealed that the decoding advantage stems from the neural decoder's capability to decode correlated errors. This exhibits the decoder's capacity to acquire novel strategies that have not been realized in classical algorithms such as UF and MWPM, demonstrating the flexibility of the ML approach. Furthermore we have investigated the robustness of the hierarchical decoder, by mismatching the training and test systems in regards to the error rate and code distance. We were able to show that even under these condition the hierarchical decoder was able to provide high accuracy decoding. This suggests that the hierarchical decoder could be used in experimental settings, where the noise varies with time.

Besides the many advantages of the hierarchical decoder, we also identified its problems in dealing with apparently simple errors, which show a high degree of degeneracy with regard to the most probable correction. We have linked this problem to the supervised training approach, which trains the decoder to learn the recovery operation that eliminates all errors, without taking into account equally valid recovery operations. Thus, training bias inhibits the trained model.

This bias can be mitigated through RL techniques which train the ML algorithm solely

based on the final outcome rather than the actions taken to achieve it. This allows for bias-free strategy development, while also being a viable option for training the algorithm in an experimental setting where the information necessary to create appropriate labels may not be available. However, this approach does pose increased training difficulties.

This leads us to our second study, the creation of a quantum benchmark environment which enables the development of RL agents and techniques to be deployed across a wide range of applications. To this end, we have implemented the quantum cartpole environment, a quantum system based on the field of quantum feedback control. By freely varying the potential, it is possible to closely match a specific problem.

In order to establish a benchmark independent of the RL, we introduced a classical surrogate model to mimic the quantum model as closely as possible. This allowed us to transfer classical control algorithms to the quantum environment, while maintaining most of their performance. Using this performance as a benchmark, we trained our own RL agents, where we divided the controlling task between two separate RL agents. One agent learns the state estimation while the other learns the control algorithm. Testing the different agents on linear and nonlinear examples of the quantum environment has shown that the RL agents are able to match or outperform the classical algorithms. When investigating what the RL agents have learned, we showed that they were able to learn novel strategies.

In addition to the performance advantage shown, the RL agent was able to achieve this performance without any knowledge of system dynamics or noise covariance, both of which were available to the LQGC. Furthermore, the RL agent used the same underlying structure and training process for all variations of the environment, unlike the LQGC, which has to be modified depending on the system and noise. The robustness and flexibility of the RL approach was demonstrated by the fact, that for every single potential the RL control strategies were able to achieve high performances without modifying the underlying approach. LQGC, on the other hand, had to be modified depending on whether the system had a linear or non-linear potential and even then it failed to achieve meaningful control in the case of the quartic potential.

An interesting line of research for the future would be to reformulate the hierarchical decoder using RL. This could allow an increase in performance in the neural decoder part, as it could learn to correct errors with degeneracy with respect to the most probable correction, and thus potentially resolve the correction bottleneck at the boundary of the RSC. In addition, an increase in performance would automatically lead to an increase in computational speed, as the subsequent classical decoder would have less to correct, while the neural decoder's computational time would remain unchanged. Another interesting direction would be to develop an RL agent or RL training technique that achieves high stability during training and is able to cope with long RL runs.

The quantum cartpole environment could also be further extended to allow a wider range of agent development. This could include generally extending the RL agent's capabilities by giving it the option to decide when to measure or with what measurement strength, or providing the agent with the raw measurements instead of just the mean values. Similar, the extension of the environment itself could be an interesting direction. This could include time-depending potentials, non-Markovian noise or interacting systems.

In parallel to our work, a variant of machine learning error correction has been developed that exploits the rational symmetry of the toric code [154]. This could also be implemented on top of the hierarchical decoder to reduce the input space and thus stabilize the training process. The latter would be even more important if implemented as an RL problem.

Another interesting line of research that has opened up is the use of machine learning in high-level control of quantum computing. IBM demonstrated this by using generative ML based on

large language models [185] to automate the development of quantum code for Qiskit [186] or by using RL for circuit compilation, claiming to reduce the needed two-qubit gates by 20-50% compared to heuristic methods [66].

To improve the transparency of the thesis, the presented data has been published [S2, S3] under open source licenses. Additionally, to contribute to the active development of machine learning applications in the field of quantum control, the quantum cartpole environment has been published [S4] and made compatible with the OpenAI Gym framework [187], allowing for easy use and attracting a large audience. The same has been done with the UF decoder implementation, which is freely available at [S1]. The syntax of the UF decoder is based on the PyMatching implementation [145], for ease of use.

Appendix

Appendix for Chapter 2

A

A.1 Expectation Grad-Log-Prob lemma

Here we want to give a short proof following [108] showing that the expectation value of the gradient of a probability is given by

$$\underset{x\sim P_\theta}{E}\left[\nabla_\theta \log P_\theta\left(x\right)\right] = 0, \tag{A.1}$$

where $P_\theta\left(x\right)$ is parametrized over the random variable x.

For this we remember that the probability distribution is normalized

$$\int_x P_\theta\left(x\right) = 1. \tag{A.2}$$

Applying the gradient to both sides of the equation results in

$$\nabla_\theta \int_x P_\theta\left(x\right) = \nabla_\theta 1 = 0. \tag{A.3}$$

Then we can use the log derivative trick to rewrite the equations

$$\begin{aligned} 0 &= \nabla_\theta \int_x P_\theta\left(x\right) && \text{(A.4)}\\ &= \int_x \nabla_\theta P_\theta\left(x\right) && \text{(A.5)}\\ &= \int_x P_\theta\left(x\right) \nabla_\theta \log P_\theta\left(x\right) && \text{(A.6)}\\ &= \underset{x\sim P_\theta}{E}\left[\nabla_\theta \log P_\theta\left(x\right)\right]. && \text{(A.7)} \end{aligned}$$

Appendix

Appendix for Chapter 3

B

B.1 Supplementing error threshold plots

In Sec. 4.2, we have presented the MWPM decoder based on the PyMatching implementation. In Fig. B.1, we present the error threshold plot for the PyMatching v0.7, used to determine the error threshold. Fig. B.2 shows the error threshold plot of the hierarchical decoder, optimized for the highest possible threshold, using UF as the secondary decoder.

Fig. B.2 shows the error threshold plot of the hierarchical decoder, optimized for the highest possible threshold, using UF and MWPM as secondary decoder. The threshold plots are shown for depolarizing on the toric code. Fig. B.3 shows the error threshold plot of the hierarchical decoder variant using lazy decoding as primary decoder and UF as secondary.

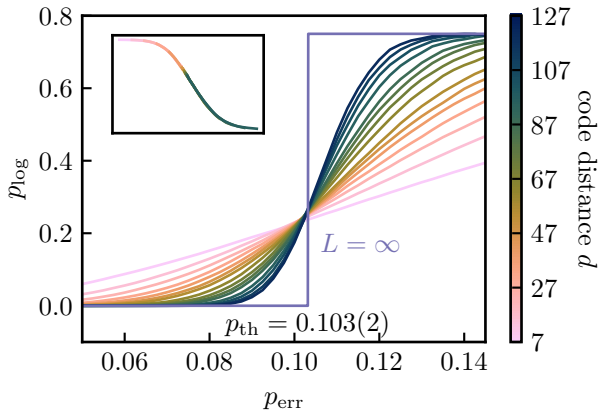


Figure B.1 –
PyMatching v0.7 threshold. Showcase of the error threshold using PyMatching v0.7 as decoder. The blue line shows the thermodynamic limit and the insert showcases the data collapse.

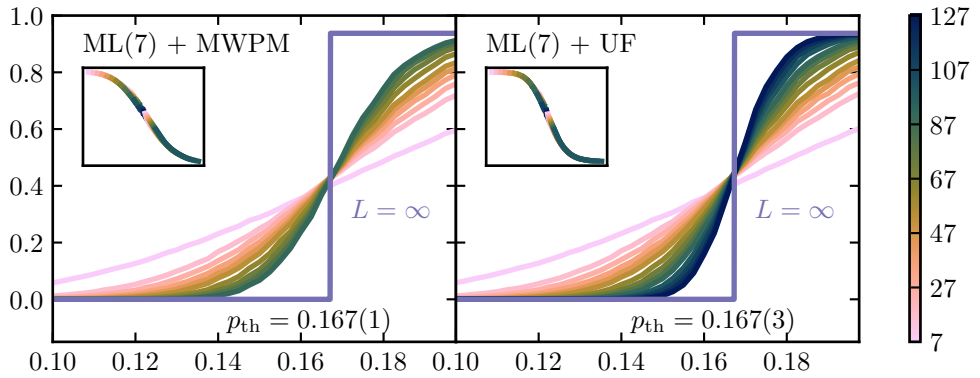


Figure B.2 – **Threshold optimized hierarchical decoder.** Both graphs show the threshold plot for the hierarchical decoder with an input size of $l = 7$ on the toric code for code distances up to $d = 127$. As secondary decoder either the MWPM (left) or UF (right) decoder was used. The blue shows the thermodynamic limit and the insert the data collapse used to determine the threshold.

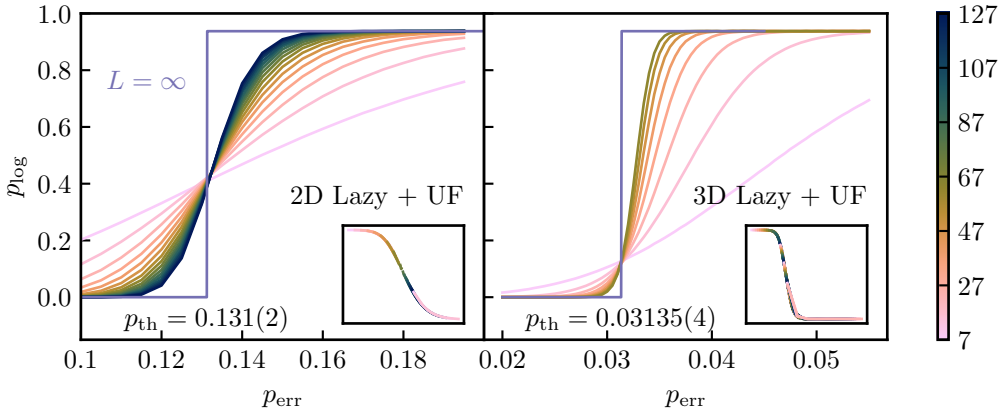


Figure B.3 – **Lazy hierarchical decoder.** The graphs show the error threshold plots for the hierarchical decoder using the Lazy decoder as primary decoder and the UF as secondary decoder. The decoder was tested on the 2D (left) and 3D (right) toric code. The blue line shows the thermodynamic limit and the insert shows the data collapse used to determine the threshold value.

B.2 Stochastic neural decoder

Here we look at the comparison between a deterministic and stochastic neural decoder. For this comparison, both neural decoders utilize the exact same underlying ANN given by the wall-clock time optimized hierarchical decoder in Tab. 4.2. The stochastic neural decoder samples the applied correction based on the returned probabilities instead of applying the correction with the highest probability. In Fig. B.4 the threshold plots of the stochastic decoder on the toric code with depolarizing noise are shown, returning an error threshold of $p_{\text{th}} = 0.14(7)$, which is $\approx 10\%$ below the threshold of the deterministic neural decoder given in Tab. 4.3.

In Fig. B.5, the difference in decoding performance is further illuminated by comparing the effective error rates of both decoders. In the low error rate regimes, both decoders converge against seemingly the same value, showcasing the same decoding performance. This is expected as the appearing errors are dominated by errors of length $l = 1$, which can be corrected with absolute certainty. Increasing the physical error rate, the difference between both decoders starts to form, as the stochastic decoder is unable to match the performance of the deterministic

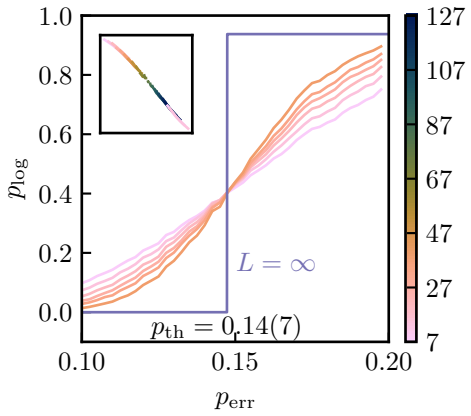


Figure B.4 – **Stochastic error threshold.** The graph shows the threshold plot for the hierarchical decoder with MWPM as secondary decoder and an input size of $l = 5$. The blue line shows the thermodynamic limit and the insert shows the data collapse used to determine the threshold value.

decoder. Moving closer to the error threshold of the stochastic decoder, the decoder reduces the initial error rate by a factor of ≈ 2.8, while the deterministic decoder reduces it by a factor of ≈ 6.1. This showcases that the possibility of correcting degenerate syndromes is outweighed by the loss of performed correction.

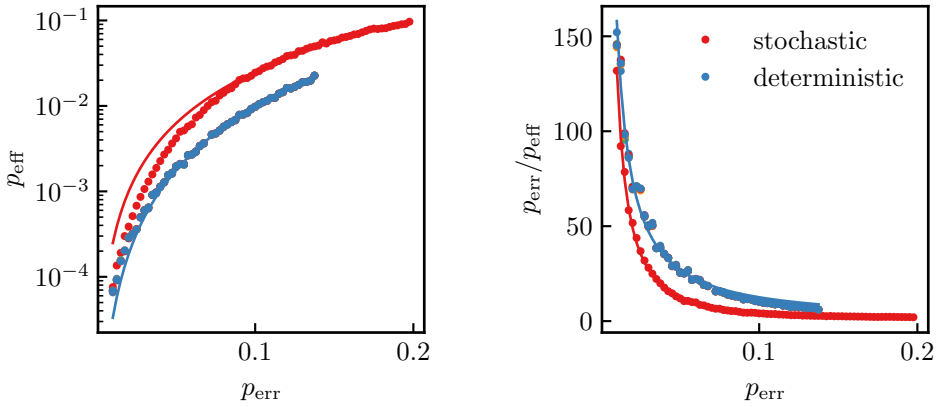


Figure B.5 – **Comparison between stochastic and deterministic corrections.** The plots compare the corrections applied via the effective error rate (left) and the error rate improvement (right). The same ANNs were used for both the deterministic and stochastic decoders.

B.3 Robustness

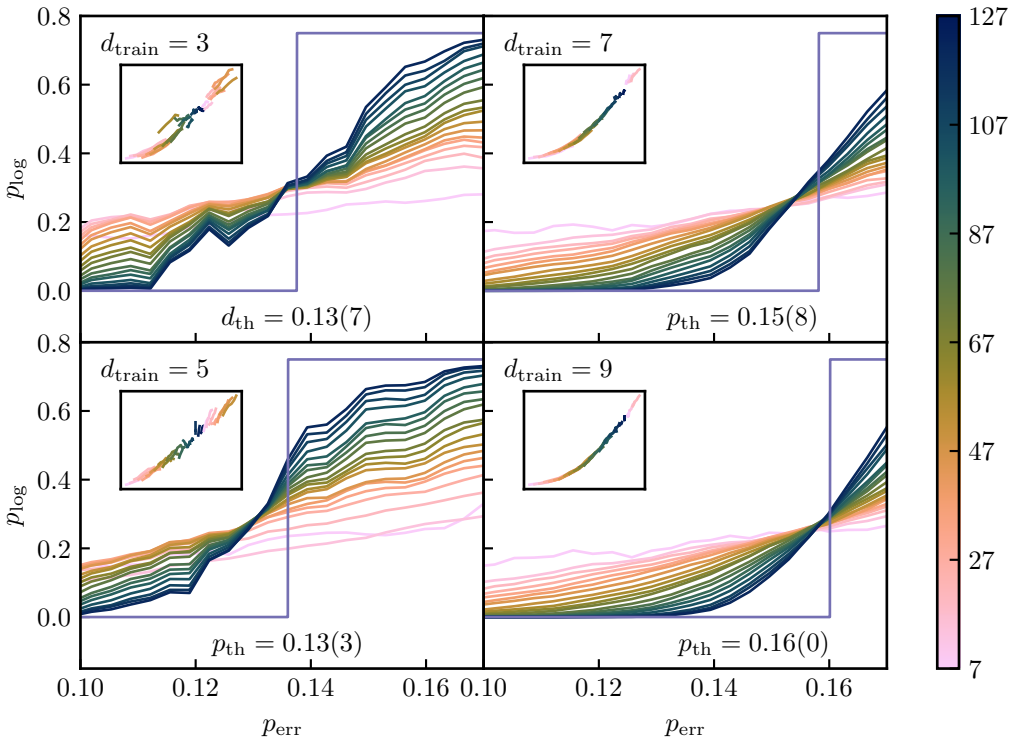


Figure B.6 – **Robustness regarding to the training code distance.** Error threshold plots of the ML(5) + UF decoder applied on the RSC. The different panels show the results of the decoder being trained on different code distances. The blue line indicates the thermodynamic limit and the insert shows the data collapse of the plots.

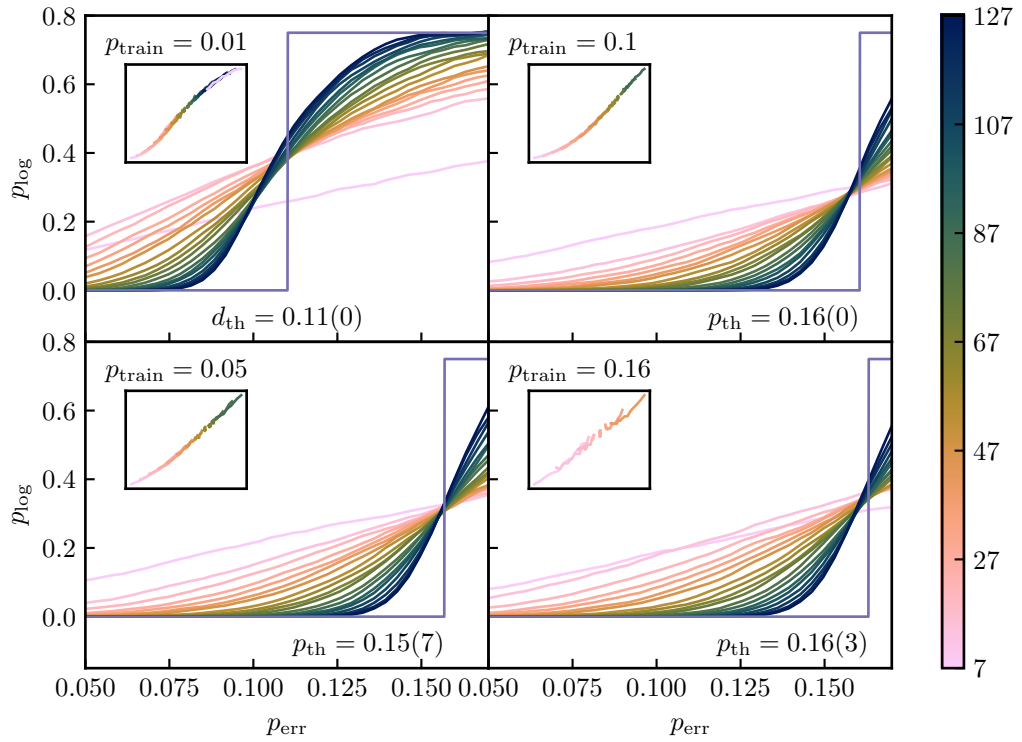


Figure B.7 – **Robustness regarding to the training error rate.** Error threshold plots of the ML(5) + UF decoder applied on the RSC. The different panels show the results of the decoder being trained on different error rates. The blue line indicates the thermodynamic limit and the insert shows the data collapse of the plots.

Appendix

Appendix for Chapter 5

C

C.1 SME calculation

Here we show the derivation of the SME for the weak measurement introduced in Sec. 5.2.1. Following the approach provided in [13], continuous weak measurements are constructed by setting the strength of the measurement, determined by the width of the ancilla wavefunction, proportional to the measured time interval $\sigma^2 = \frac{1}{\kappa\Delta t}$. Substituting this into Eq. (5.21) and Eq. (5.37) yields

$$M_m = \mathcal{N} \exp\left[-\frac{\kappa\Delta t}{4}(\lambda A - m)^2\right] \tag{C.1}$$

$$m = \lambda\langle A\rangle + \frac{\Delta W}{\sqrt{\kappa}\Delta t}, \tag{C.2}$$

ΔW is now a Wiener increment of a zero mean Gaussian with variance Δt. Applying the Kraus operator to the system and expanding the term gives

$$|\psi(t+dt)\rangle \propto \exp\left[-\frac{\kappa\Delta t}{4}(\lambda\alpha - m)^2\right]|\psi(t)\rangle \tag{C.3}$$

$$\propto \exp\left[-\frac{\kappa\Delta t}{4}\lambda^2\alpha^2 + \frac{\kappa\Delta t}{2}\lambda A m\right]|\psi(t)\rangle \tag{C.4}$$

$$\propto \exp\left[-\frac{\kappa\Delta t}{4}\lambda^2 A^2 + \frac{\kappa\Delta t}{2}\lambda^2 A\langle A\rangle + \frac{\sqrt{\kappa}}{2}\lambda A\Delta W\right]|\psi(t)\rangle, \tag{C.5}$$

where all terms independent of A are considered to be part of some pre-factor. Now, the exponential term is expanded in the first order of Δt, which gives

$$|\psi(t+\Delta t)\rangle \propto \left[1 - \frac{\kappa\Delta t}{4}\lambda^2 A^2 + \frac{\kappa\Delta t}{2}\lambda^2 A\langle A\rangle + \frac{\sqrt{\kappa}}{2}\lambda A\Delta W + \frac{\kappa}{8}\lambda^2 A^2\Delta W^2\right]|\psi(t)\rangle. \tag{C.6}$$

Note that we have kept a second order term in ΔW. This is because in the infinitesimal limit $\Delta t \to dt$, $\Delta W^2 \to dW^2 = dt$. Applying this limit to the equation results in

$$|\psi(t+\delta t)\rangle \propto \left[1 - \frac{\kappa\delta t}{4}\lambda^2 A^2 + \frac{\kappa\delta t}{2}\lambda^2 A\langle A\rangle + \frac{\sqrt{\kappa}}{2}\lambda A\delta W + \frac{\kappa}{8}\lambda^2 A^2\delta t\right]|\psi(t)\rangle \tag{C.7}$$

$$\propto \left[1 - \frac{\kappa\delta t}{8}\lambda^2 A^2 + \frac{\kappa\delta t}{2}\lambda^2 A\langle A\rangle + \frac{\sqrt{\kappa}}{2}\lambda A\delta W\right]|\psi(t)\rangle. \tag{C.8}$$

As we have not included the pre-factors of the Kraus operator, the current equation does not preserve the norm of the wavefunction. We can correct this by normalizing $|\psi(t+dt)\rangle$ and

expanding the results in the first order of dt again. Writing down the result in the form $|\psi(t+dt)\rangle = |\psi(t)\rangle + d|\psi\rangle$ gives the stochastic differential equation

$$d|\psi\rangle = \left[-\frac{\kappa\lambda^2}{8}(A-\langle A\rangle)^2 dt + \frac{\sqrt{\kappa}\lambda}{2}(A-\langle A\rangle)dW\right]|\psi(t)\rangle. \tag{C.9}$$

Now we only have to rewrite the equation using the density matrix, where we use $\rho(t+dt) = \rho(t) + d\rho(dt)$, giving us the SME:

$$\begin{aligned}
d\rho &= (d|\psi\rangle)\langle\psi| + |\psi\rangle(d\langle\psi|) + (d|\psi\rangle)(d\langle\psi|) && \text{(C.10)}\\
&= -\frac{\kappa\lambda^2}{8}(A-\langle A\rangle)^2\rho dt + \frac{\sqrt{\kappa}\lambda}{2}(A-\langle A\rangle)dW\rho \\
&\quad -\rho\frac{\kappa\lambda^2}{8}(A-\langle A\rangle)^2 dt + \rho\frac{\sqrt{\kappa}\lambda}{2}(A-\langle A\rangle)dW && \text{(C.11)}\\
&\quad +\frac{\kappa\lambda^2}{4}(A-\langle A\rangle)\rho(A-\langle A\rangle)dW^2 \\
&= -\frac{\kappa\lambda^2}{8}A^2\rho dt + \frac{\sqrt{\kappa}\lambda}{2}(A\rho-\langle A\rangle\rho)dW \\
&\quad -\frac{\kappa\lambda^2}{8}A^2\rho dt + \frac{\sqrt{\kappa}\lambda}{2}(\rho A-\langle A\rangle\rho)dW && \text{(C.12)}\\
&\quad +\frac{\kappa\lambda^2}{4}A\rho A dt \\
&= -\frac{\kappa\lambda^2}{4}\left(A\rho A - \frac{1}{2}(A^2\rho+\rho A^2)\right)dt + \frac{\sqrt{\kappa}\lambda}{2}(A\rho+\rho A-2\langle A\rangle\rho)dW && \text{(C.13)}\\
&= -\frac{\kappa\lambda^2}{4}[A,[A,\rho]]dt + \frac{\sqrt{\kappa}\lambda}{2}(A\rho+\rho A-2\langle A\rangle\rho)dW. && \text{(C.14)}
\end{aligned}$$

It has to be noted that we have done this derivation without any consideration of a Hamiltonian acting on a system. Including a Hamiltonian does not change the derivation and adds the $-\frac{i}{\hbar}[H,\rho]dt$ at the end.

Appendix

Appendix for Chapter 6

D

D.1 Sequential weak measurements

Here we give a short proof for the post measurement variance and mean describing the system given in Eq. (6.3) and Eq. (6.4). Starting with the position measurement, the wavefunction and Kraus operator follow the description of Eq. (6.1) and Eq. (5.21) as

$$\psi(x) = \left(\frac{1}{2\pi\sigma_{\text{sys}}}\right)^{1/4} e^{-\frac{(x-\langle x\rangle)^2}{4\sigma_{\text{sys}}^2}} e^{-ix\langle p\rangle}$$
$$M_m(x) = \left(\frac{1}{2\pi\sigma_{\text{anc}}}\right)^{1/4} e^{\frac{-(x-m)^2}{4\sigma_{\text{anc}}^2}}.$$

Here we have set $\lambda = 1$ in the Kraus operator. The post measurement state follows

$$\psi'(x) = \frac{1}{\mathcal{N}} M_m(x)\psi(x) \tag{D.1}$$
$$= \frac{1}{\mathcal{N}} \left(\frac{1}{2\pi\sigma_{\text{anc}}}\right)^{1/4} \left(\frac{1}{2\pi\sigma_{\text{sys}}}\right)^{1/4} e^{-\frac{(x-m)^2}{4\sigma_{\text{anc}}^2} - \frac{(x-\langle x\rangle)^2}{4\sigma_{\text{sys}}^2}} e^{-ix\langle p\rangle}, \tag{D.2}$$

where $\mathcal{N}$ is some normailization. As we are only interested in σ_{post} and $\langle x\rangle_{\text{post}}$, there is no need to further specify the normalization. The two quantities are determined by the combined exponent of the exponential function

$$\beta = \frac{-\sigma_{\text{sys}}^2 (x-m)^2 - \sigma_{\text{anc}}^2 (x-\langle x\rangle)^2}{4\sigma_{\text{sys}}^2\sigma_{\text{anc}}^2}. \tag{D.3}$$

Expanding this equation and sorting for x yields

$$\beta = \frac{\left(\sigma_{\text{sys}}^2 + \sigma_{\text{anc}}^2\right) x^2 - 2\left(\langle x\rangle\sigma_{\text{anc}}^2 + m\sigma_{\text{sys}}^2\right) x + \langle x\rangle^2\sigma_{\text{anc}}^2 + m^2\sigma_{\text{sys}}^2}{4\sigma_{\text{sys}}^2\sigma_{\text{anc}}^2}. \tag{D.4}$$

Dividing the fraction by $\left(\sigma_{\text{sys}}^2 + \sigma_{\text{anc}}^2\right)$ so that x^2 is isolated

$$\beta = \frac{x^2 - 2\frac{\left(\langle x\rangle\sigma_{\text{anc}}^2 + m\sigma_{\text{sys}}^2\right)}{\left(\sigma_{\text{sys}}^2+\sigma_{\text{anc}}^2\right)} x + \frac{\langle x\rangle^2\sigma_{\text{anc}}^2 + m^2\sigma_{\text{sys}}^2}{\left(\sigma_{\text{sys}}^2+\sigma_{\text{anc}}^2\right)}}{4\frac{\sigma_{\text{sys}}^2\sigma_{\text{anc}}^2}{\left(\sigma_{\text{sys}}^2+\sigma_{\text{anc}}^2.\right)}} \tag{D.5}$$

The resulting term in the numerator almost looks like a square binomial theorem. We complete by adding some constant factor ϵ

$$\epsilon = \frac{\left(\frac{\langle x\rangle\sigma_{\text{anc}}^2 + m\sigma_{\text{sys}}^2}{\sigma_{\text{sys}}^2+\sigma_{\text{anc}}^2}\right)^2 - \left(\frac{\langle x\rangle\sigma_{\text{anc}}^2 + m\sigma_{\text{sys}}^2}{\sigma_{\text{sys}}^2+\sigma_{\text{anc}}^2}\right)^2}{4\frac{\sigma_{\text{sys}}^2\sigma_{\text{anc}}^2}{\left(\sigma_{\text{sys}}^2+\sigma_{\text{anc}}^2\right)}} = 0. \tag{D.6}$$

Adding it to the exponent and reshuffling the equation yields

$$\beta = \frac{\left(x - \frac{\langle x\rangle\sigma_{\text{anc}}^2 + m\sigma_{\text{sys}}^2}{\sigma_{\text{sys}}^2+\sigma_{\text{anc}}^2}\right)^2}{4\frac{\sigma_{\text{sys}}^2\sigma_{\text{anc}}^2}{(\sigma_{\text{sys}}^2+\sigma_{\text{anc}}^2)}} + \frac{(\langle x\rangle - m)^2}{4\left(\sigma_{\text{sys}} + \sigma_{\text{anc}}\right)}. \tag{D.7}$$

Inserting the exponent back to the post measurement wavefunction results in the new form of

$$\psi'(x) = \frac{A}{\mathcal{N}}\exp\left[\frac{\left(x - \frac{\langle x\rangle\sigma_{\text{anc}}^2 + m\sigma_{\text{sys}}^2}{\sigma_{\text{sys}}^2+\sigma_{\text{anc}}^2}\right)^2}{4\frac{\sigma_{\text{sys}}^2\sigma_{\text{anc}}^2}{(\sigma_{\text{sys}}^2+\sigma_{\text{anc}}^2)}}\right], \tag{D.8}$$

here we have expressed the second exponent term as the factor A, since it is independent of x. Now σ_{post} and $\langle x\rangle_{\text{post}}$ follow immediately from calculating the probability density $|\psi'(x)|^2$ as

$$\sigma_{\text{post}} = \left(\frac{\sigma_{\text{sys}}^2\sigma_{\text{anc}}^2}{\sigma_{\text{sys}}^2 + \sigma_{\text{anc}}^2}\right)^{1/2} \qquad \langle x\rangle_{\text{post}} = \frac{\langle x\rangle\sigma_{\text{anc}}^2 + m\sigma_{\text{sys}}^2}{\sigma_{\text{sys}}^2 + \sigma_{\text{anc}}^2}, \tag{D.9}$$

The same calculation can be performed for the momentum weak measurement, where we first have to transform the wavefunction into momentum space representation and update the Kraus operator with the momentum operator

$$\psi(p) = \left(\frac{\sigma}{\pi}\right)^{1/4} e^{-\frac{\sigma_{\text{sys}}^2}{2}(p-\langle p\rangle)^2} e^{-i(p-\langle p\rangle)\langle x\rangle} \tag{D.10}$$

$$M_m(p) = \left(\frac{1}{2\pi\sigma_{\text{anc}}}\right)^{1/4} e^{\frac{-(p-m)^2}{4\sigma_{\text{anc}}^2}} \tag{D.11}$$

Afterwards, the calculation follows the same steps as laid out before, with the final results of

$$\sigma_{\text{post}} = 2\left(\frac{\frac{4}{\sigma_{\text{sys}}^2}\sigma_{\text{anc}}^2}{\frac{4}{\sigma_{\text{sys}}^2} + \sigma_{\text{anc}}^2}\right)^{-1/2} \qquad \langle p\rangle_{\text{post}} = \frac{\langle p\rangle\sigma_{\text{anc}}^2 + m\frac{4}{\sigma_{\text{sys}}^2}}{\frac{4}{\sigma_{\text{sys}}^2} + \sigma_{\text{anc}}^2} \tag{D.12}$$

D.2 Training process

The training of the agents follows the general description in 2.3.4. As we train two different types of agents, the hyperparameters vary between training. The parameters are listed in Tab. D.1 and were determined via extensive grid search. In both cases we utilize tanh function as activation function as the input and output used are normalized. This is a straightforward process regarding the controlling force, as the normalization maps $[-F_{\max}, F_{\max}]$ to $[-1, 1]$. As the measurement can very well lay outside the allowed area, we chose to normalize the measurement and state estimation by a factor of $\frac{1}{2x_{\text{th}}}$.

The RLC are always trained in combination with the used state estimation technique, in order to adapt to the provided input. This means in the case of the RLC + RLE decoder, the RLE have to be trained before the RLC. During the training process using the RLE or Kalman Filter as state estimator, we have noticed that with a small number of measurements, the training process often gets stuck early during the training or experiences performance degradation, which out being able to recover. This resulted in performance nowhere comparable to the

Reinforcement Learning Parameters		
Parameter	RLE	RLC
learning rate	2e-05 $\cdot N_{\text{meas}}$	3e-05
clipping rate	0.7	0.5
epochs	1000	10000
steps per epoch	100000	200000
batchsize	2048	1024
nepochs	20	10
action network	[32, 32]	[32, 32]
value network	[32, 32]	[32, 32]
activation function	tanh	tanh

Table D.1 – **Hyperparameters of the reinforcement learning approach**, specifying the initialization of the RLC and RLE agents and their respective training processes. This table was taken from [P1].

LQGC. As this effect didn't occur as often for higher number of measurements, we deployed transfer learning in the training process. This approach is based on the assumption that the optimal agents of two environments, with only slight differences from each other, also differ only slightly from each other [34]. This approach is implemented by starting the training process at the highest number of investigated measurements $N_{\text{meas}} = 48$. After the training process is completed, the training is moved to the next smaller number of measurements, keeping the previously trained agent as starting point. This is repeated until $N_{\text{meas}} = 1$.

In Fig. D.1 (a) the average performance of the trained agents are shown using $t_{\text{termination}}$. The agents are tested on the quantum system with an inverse quartic potential. Additionally to the transfer learning approach, we also trained multiple agents in parallel. It turns out that for large measurement numbers the training of the agents has a high success rate, as all agents show a similar performance. As the number of measurement increases, the training process becomes more and more unstable and the difference between agents starts to grow. Finally, at $N_{\text{meas}} = 1$, a large spectrum of performance can be seen, where the best performing agent

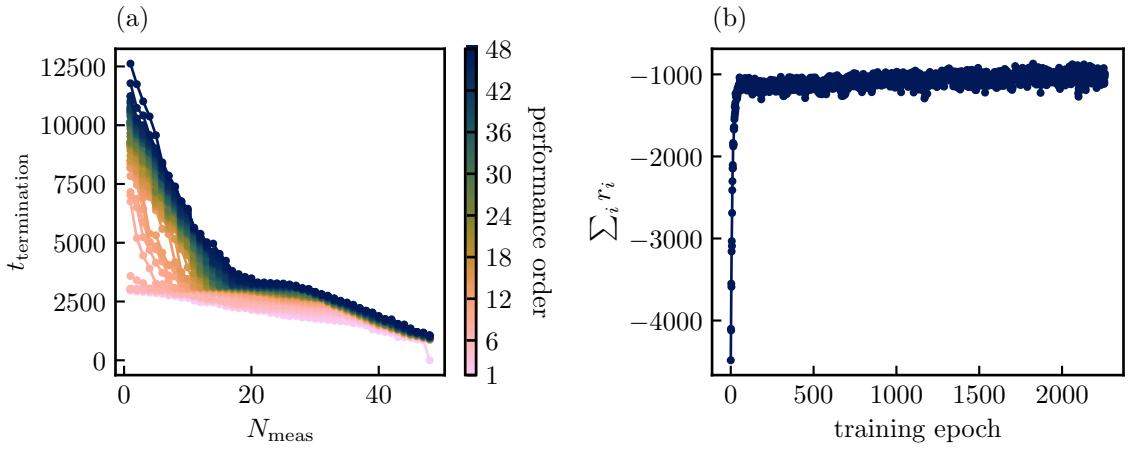


Figure D.1 – **Training stability.** The training process of the RLC and RLE differs from each other as the RLC showcases a more volatile training compared to the RLE. For the RLC, 48 different agents are trained for each number of measurements. The performance of these agents (a) is tested on the quantum system with an inverse quartic potential. For visual clarity the agents are color coded based on the performance. For the RLE (b) the training is visualized using the accumulated rewards $\sum_i r_i$ per epoch.

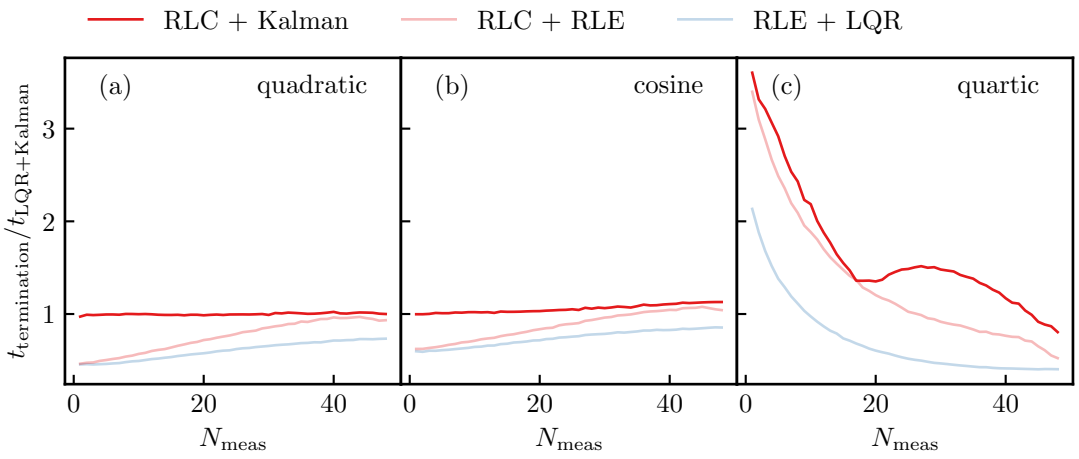


Figure D.2 – **Improvement rate** of different control algorithm performances compared to the LQR + Kalman controller on the classical environment. These control algorithms include three different combinations of RL based controllers, namely the RLC + Kalman (solid red), RLC + RLE (transparent red) and LQR + RLE (transparent blue). The investigated potentials are the inverted quadratic (a), cosine (b) and inverted quartic (c) potentials. This plot was taken and resized from [P1].

surpasses the worst performing agent by a factor of ≈ 5.

Now when we consider the training process of the RLE, it is important to note that the training of the agent does not depend on a following controller. Therefore, we have opted to apply force randomly to the wavefunction during training. This approach ensures that the agent can explore the entire observation space. In contrast to the training of the RLC, the RLE training exhibits a rapid convergence towards the maximum reward value, as indicated in D.1 b).

D.3 Extended benchmarks

In this section we present additional data taken from the benchmark measurement for different potentials in Sec. 6.2.3. In Fig. D.2 the benchmarking results comparing the different decoders on the classical surrogate are shown. Similar to the results on the quantum system, the RLC + Kalman controller manages to match the performance of the LQGC for the quadratic and cosine potential, while increasing the performance in the quartic performance by a factor of 3. The other two decoders, RLC + RLE and RLE + LQR, are unable to perform a similar feat and fall behind the LQGC for the quadratic and cosine potentials. There the RLC + RLE controller only achieves a similar performance when performing 40 or more weak measurements per input. In the quartic potential, both controllers are able to surpass the LQGC by at least a factor of 2 for $N_{\text{meas}} = 1$, while the RLC + RLE decoder is able to show almost the same performance as the RLC + Kalman decoder.

In Fig. D.3 the collected raw measurement benchmarking results are shown, where the results of the classical surrogate are shown in the left row, where as the middle and right rows show results from the quantum system. The results in the middle row were gathered by training the agents on the classical surrogate and then applying them on the quantum system.

The first column presents the benchmarking results on the inverted quadratic potential, showing a complete match between the performance of the LQGC and the RLC + Kalman decoder for both the classical and quantum systems. As the LQGC controller is proven op-

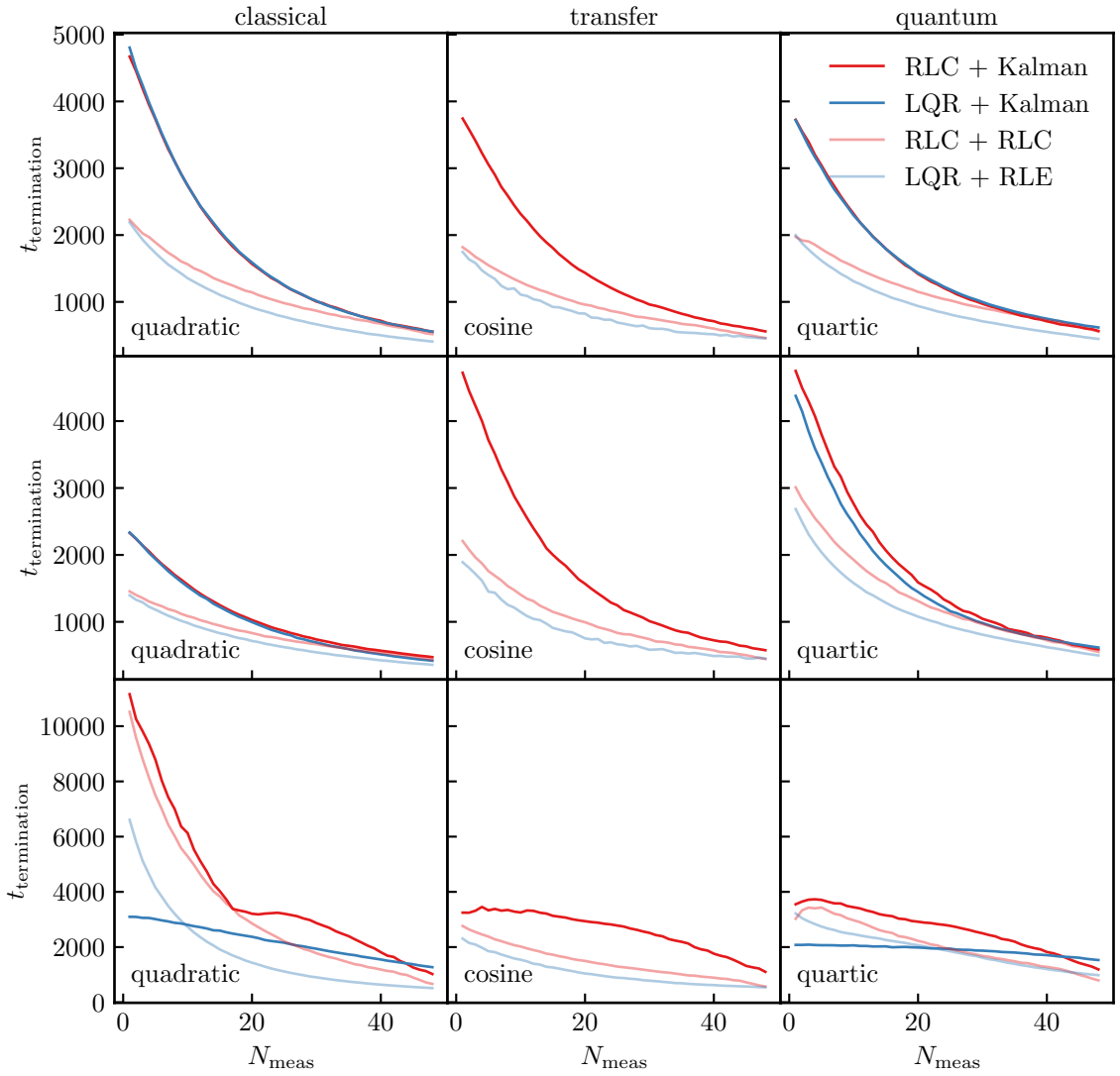


Figure D.3 – **Raw benchmark measurement** results of the four decoder combinations: LQGC (blue), RLC + Kalman (red), RLE + LQR (light blue) and RLC +RLE (light red). The results are sorted according to the different potentials and used environments. The three rows represent from top to bottom the quadratic, cosine and quartic potential, the columns showcase the different system types, namely the classical, transfer and quantum system from left to right. Here the transfer system indicates the performance of the agents trained on the classical system and then applied on the quantum system.

timal in case of $N_{\text{meas}} = 1$ of the linear classical system, the results suggest that the LQGC performance is close to optimal even for a different number of measurements and for the quantum system. Comparing the overall maximum average terminations between the classical and quantum environments, the quantum system reports a termination time about 1000 time steps lower, showing that there are still difference between the quantum and classical system.

A similar, but more exaggerated, behavior can be seen in the quartic potential, where the maximum average termination time is less than half of the classical case. The opposite is true for the cosine potential, where the classical system has less than half the maximum performance of the quantum system. We suspect that these differences results from the fact that the wavepacket extends over the position space, whereas in the classical case we have a point mass. This allows

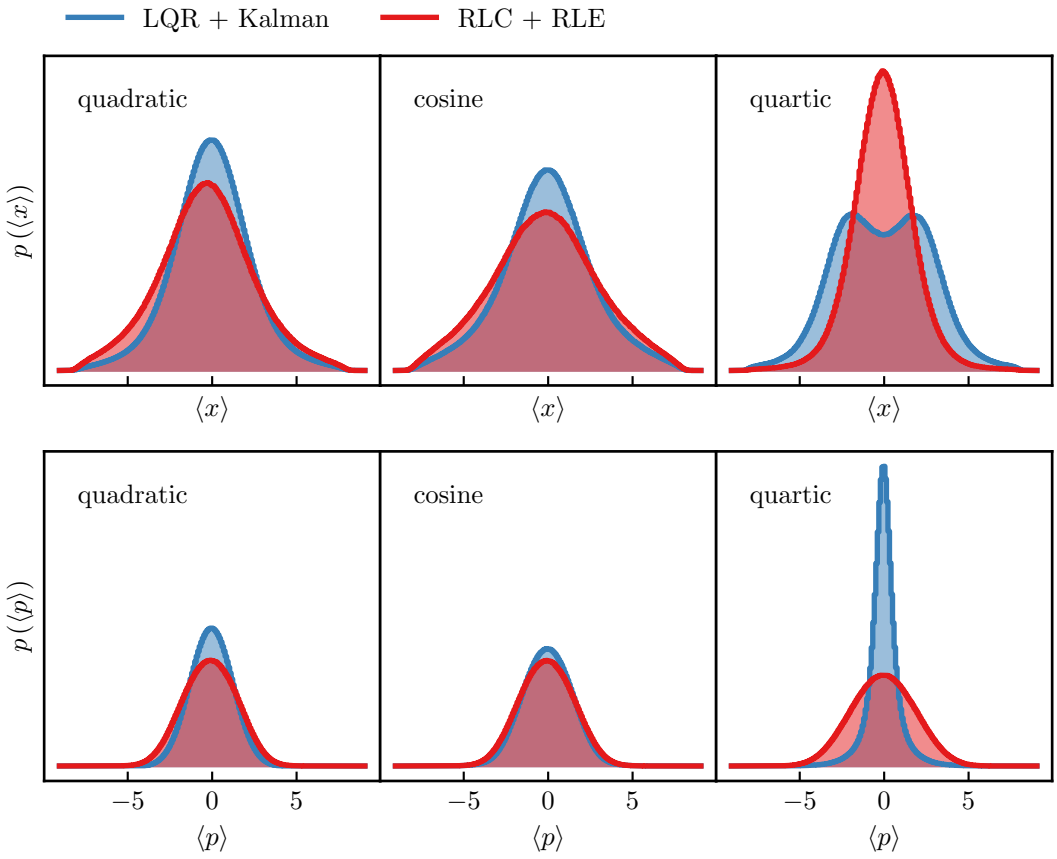


Figure D.4 – **Distribution of position and momentum of the stabilized wavepacket** on the classical surrogate system for the three different potentials: (I) **quadratic**, (II) **cosine**, (III) **quartic**. The position $\langle x \rangle$ (top row) and momentum $\langle p \rangle$ (bottom row) are traced over 10^6 time steps for the LQR + Kalman (red) and RLC + RLE (red) controllers and converted into a histogram of the probability $p(\langle x \rangle)$. This plot was taken and modified from [P1].

the wavefunction to be influenced by the potential outside the threshold boarder. In the case of the cosine potential, the potential forms a minimum outside the border, which influences the wavefunction and helps in stabilizing the wavepaket close to the thresholds.

D.4 Classical strategy analysis

Fig. D.4 shows the probability distribution of $\langle x \rangle$ and $\langle p \rangle$ for the LQGC and RLC + RLE controller and the classical surrogate system. Similar to the results shown for the quantum system in Fig. 6.10, the LQGC shows a perfectly symmetrical distribution for the position and momentum, independent from the sued potential, while the RLC + RLE decoder shows a slightly asymmetrical distribution, best seen in the position distribution of the inverted quadratic potential.

A notable difference to the quantum environment behavior is shown by the RLC + RLE decoder in the cosine potential. While in the quantum system the position probability showed a wide plateau, indicating the ability to stabilize the potential near the threshold, here the position shows a clear maximum at the center of the potential. This reinforces the argument that in the quantum case the wavefunction is influenced by the potential outside of the threshold.

Most interestingly, the LQGC is unable to stabilize the wavepacket in the center of the potential for the inverted quartic potential, but instead tries to stabilize it left and right from it. Together with the momentum distribution, being a sharp peak, this is a hint that the LQGC is no longer able to properly control the wavepacket. Based on the results from Fig. D.3 in the classical system, where the RLC + Kalman control delivers the highest average termination time, we can deduce that the LQR forms the bottleneck in the LQGC.

Bibliography

[P1] **K. Meinerz**, S. Trebst, M. Rudner, and E. van Nieuwenburg, The quantum cartpole: A benchmark environment for non-linear reinforcement learning, 2023. arXiv: 2311.00756 [quant-ph].

[P2] **K. Meinerz**, C.-Y. Park, and S. Trebst, Scalable neural decoder for topological surface codes, Phys. Rev. Lett. **128** (8), 080505, 2022.

[S1] C.-Y. Park and **K. Meinerz**, Open-source c++ implementation of the union-find decoder, https://github.com/chaeyeunpark/UnionFind, 2020.

[S2] **K. Meinerz**, S. Trebst, M. Rudner, and E. van Nieuwenburg, Dataset underpinning: "the quantum cartpole", version 1.0.0, 2023.

[S3] **K. Meinerz**, C.-Y. Park, and S. Trebst, Dataset underpinning: "Scalable Neural Decoder for Topological Surface Codes", 2023.

[S4] **K. Meinerz**, Quantum cartpole environment, version v0.1.0, 2023.

[1] H. Rabitz, R. de Vivie-Riedle, M. Motzkus, and K. Kompa, Whither the future of controlling quantum phenomena? Science **288** (5467), 824–828, 2000. eprint: https://www.science.org/doi/pdf/10.1126/science.288.5467.824.

[2] A. Weiner, D. Leaird, J. Patel, and J. Wullert, Programmable shaping of femtosecond optical pulses by use of 128-element liquid crystal phase modulator, IEEE Journal of Quantum Electronics **28** (4), 908–920, 1992.

[3] I. Petersen, Quantum control theory and applications: A survey, IET Control Theory & Applications **4** (12), 2651–2671(20), 2010.

[4] S. Lloyd, Coherent quantum feedback, Phys. Rev. A **62** (2), 022108, 2000.

[5] A. G. J. MacFarlane, J. P. Dowling, and G. J. Milburn, Quantum technology: The second quantum revolution, Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **361** (1809), 1655–1674, 2003. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2003.1227.

[6] A. Acín *et al.*, The quantum technologies roadmap: A european community view, New Journal of Physics **20** (8), 080201, 2018.

[7] A. C. Doherty, S. Habib, K. Jacobs, H. Mabuchi, and S. M. Tan, Quantum feedback control and classical control theory, Phys. Rev. A **62** (1), 012105, 2000.

[8] J. Preskill, Quantum Computing in the NISQ era and beyond, Quantum **2**, 79, 2018.

[9] L. D. Landau and L. M. Lifshitz, *Quantum Mechanics Non-Relativistic Theory, Third Edition: Volume 3*, 3rd ed. Butterworth-Heinemann, 1981.

[10] N. Khaneja, B. Luy, and S. J. Glaser, Boundary of quantum evolution under decoherence, Proceedings of the National Academy of Sciences **100** (23), 13162–13166, 2003. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.2134111100.

[11] N. Khaneja, T. Reiss, B. Luy, and S. J. Glaser, Optimal control of spin dynamics in the presence of relaxation, Journal of Magnetic Resonance **162** (2), 311–319, 2003.

[12] N. Khaneja, J.-S. Li, C. Kehlet, B. Luy, and S. J. Glaser, Broadband relaxation-optimized polarization transfer in magnetic resonance, Proceedings of the National Academy of Sciences **101** (41), 14742–14747, 2004. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.0404820101.

[13] K. Jacobs and D. A. Steck, A straightforward introduction to continuous quantum measurement, Contemporary Physics **47** (5), 279–303, 2006.

[14] G. Lindblad, On the generators of quantum dynamical semigroups, Communications in Mathematical Physics **48** (2), 119–130, 1976.

[15] B. Qi, On the quantum master equation under feedback control, Science in China Series F: Information Sciences **52** (11), 2133–2139, 2009.

[16] B. Chen, J. Wang, and Y. Zhou, Quantum control and its application: A brief introduction, Journal of Physics: Conference Series **1802** (2), 022068, 2021.

[17] B. Qi, A two-step strategy for stabilizing control of quantum systems with uncertainties, Automatica **49** (3), 834–839, 2013.

[18] D. Dong and Y. Wang, *Several recent developments in estimation and robust control of quantum systems*. 2017, pp. 190–195.

[19] N. Khaneja, R. Brockett, and S. J. Glaser, Time optimal control in spin systems, Phys. Rev. A **63** (3), 032308, 2001.

[20] D. D'Alessandro and M. Dahleh, Optimal control of two-level quantum systems, IEEE Transactions on Automatic Control **46** (6), 866–876, 2001.

[21] H. Zhang and H. Rabitz, Robust optimal control of quantum molecular systems in the presence of disturbances and uncertainties, Phys. Rev. A **49** (4), 2241–2254, 1994.

[22] C. D'Helon and M. R. James, Stability, gain, and robustness in quantum feedback networks, Phys. Rev. A **73** (5), 053803, 2006.

[23] N. Yamamoto, H. I. Nurdin, M. R. James, and I. R. Petersen, Avoiding entanglement sudden death via measurement feedback control in a quantum network, Phys. Rev. A **78** (4), 042339, 2008.

[24] E. J. Griffith, C. D. Hill, J. F. Ralph, H. M. Wiseman, and K. Jacobs, Rapid-state purification protocols for a cooper pair box, Phys. Rev. B **75** (1), 014511, 2007.

[25] W. C. Myrvold and J. Christian, *Quantum Reality, Relativistic Causality, and Closing the Epistemic Circle*. Springer Netherlands, 2009.

[26] L. Bouten, R. van Handel, and M. James, An introduction to quantum filtering, 2006. arXiv: math/0601741 [math.OC].

[27] A. Strikis, D. Qin, Y. Chen, S. C. Benjamin, and Y. Li, Learning-based quantum error mitigation, PRX Quantum **2** (4), 040330, 2021.

[28] M. Y. Niu, S. Boixo, V. N. Smelyanskiy, and H. Neven, Universal quantum control through deep reinforcement learning, npj Quantum Information **5** (1), 33, 2019.

[29] M. Bukov, A. G. R. Day, D. Sels, P. Weinberg, A. Polkovnikov, and P. Mehta, Reinforcement learning in different phases of quantum control, Phys. Rev. X **8** (3), 031086, 2018.

[30] M. Dalgaard, F. Motzoi, J. J. Sørensen, and J. Sherson, Global optimization of quantum dynamics with alphazero deep exploration, npj Quantum Information **6** (1), 2020.

[31] Z. An and D. L. Zhou, Deep reinforcement learning for quantum gate control, EPL (Europhysics Letters) **126** (6), 60002, 2019.

[32] H. P. Nautrup, N. Delfosse, V. Dunjko, H. J. Briegel, and N. Friis, Optimizing quantum error correction codes with reinforcement learning, Quantum **3**, 215, 2019.

[33] R. Sweke, M. S. Kesselring, E. P. L. van Nieuwenburg, and J. Eisert, Reinforcement learning decoders for fault-tolerant quantum computation, Machine Learning: Science and Technology **2** (2), 025005, 2020.

[34] K. Weiss, T. M. Khoshgoftaar, and D. Wang, A survey of transfer learning, Journal of Big Data **3** (1), 9, 2016.

[35] D. Xu, A. B. Özgüler, G. Di Guglielmo, N. Tran, G. N. Perdue, L. Carloni, and F. Fahim, Neural network accelerator for quantum control, 43–49, 2022.

[36] D. Deutsch and R. Penrose, Quantum theory, the church–turing principle and the universal quantum computer, Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences **400** (1818), 97–117, 1985. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rspa.1985.0070.

[37] O. Goldreich, On promise problems (a survey in memory of shimon even [1935-2004]), Electron. Colloquium Comput. Complex. **TR05**, 2005.

[38] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM Journal on Computing **26** (5), 1484–1509, 1997. eprint: https://doi.org/10.1137/S0097539795293172.

[39] P. Shor, *Algorithms for quantum computation: discrete logarithms and factoring*. 1994, pp. 124–134.

[40] D. Harvey and J. van der Hoeven, Integer multiplication in time $O(n \log n)$, Annals of Mathematics **193** (2), 563–617, 2021.

[41] D. Coppersmith, Modifications to the number field sieve, Journal of Cryptology **6** (3), 169–180, 1993.

[42] R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM **21** (2), 120–126, 1978.

[43] R. P. Feynman, Simulating physics with computers, International Journal of Theoretical Physics **21** (6), 467–488, 1982.

[44] J. Lee, D. W. Berry, C. Gidney, W. J. Huggins, J. R. McClean, N. Wiebe, and R. Babbush, Even more efficient quantum computations of chemistry through tensor hypercontraction, PRX Quantum **2** (3), 2021.

[45] M. Reiher, N. Wiebe, K. M. Svore, D. Wecker, and M. Troyer, Elucidating reaction mechanisms on quantum computers, Proceedings of the National Academy of Sciences **114** (29), 7555–7560, 2017. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1619152114.

[46] M. Capdevila-Cortada, Electrifying the haber–bosch, Nature Catalysis **2** (12), 1055–1055, 2019.

[47] B. M. Bolten and T. DeGregorio, Trends in development cycles, Nature Reviews Drug Discovery **1** (5), 335–336, 2002.

[48] J. J. Goings, A. White, J. Lee, C. S. Tautermann, M. Degroote, C. Gidney, T. Shiozaki, R. Babbush, and N. C. Rubin, Reliably assessing the electronic structure of cytochrome p450 on today's classical computers and tomorrow's quantum computers, Proceedings of the National Academy of Sciences **119** (38), 2022.

[49] H. Zu, W. Dai, and A. de Waele, Development of dilution refrigerators—a review, Cryogenics **121**, 103390, 2022.

[50] N. Ezzell, B. Pokharel, L. Tewala, G. Quiroz, and D. A. Lidar, Dynamical decoupling for superconducting qubits: A performance survey, 2023. arXiv: 2207.03670 [quant-ph].

[51] D. Gottesman, An introduction to quantum error correction and fault-tolerant quantum computation, 2009. arXiv: 0904.2557 [quant-ph].

[52] R. Acharya *et al.*, Suppressing quantum errors by scaling a surface code logical qubit, Nature **614** (7949), 676–681, 2023.

[53] Y. Wu *et al.*, Strong quantum computational advantage using a superconducting quantum processor, Physical Review Letters **127** (18), 2021.

[54] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Topological quantum memory, Journal of Mathematical Physics **43** (9), 4452–4505, 2002.

[55] C. Ahn, A. C. Doherty, and A. J. Landahl, Continuous quantum error correction via quantum feedback control, Phys. Rev. A **65** (4), 042301, 2002.

[56] Z. Hu, Y. Lin, Q. Guan, and W. Jiang, Battle against fluctuating quantum noise: Compression-aided framework to enable robust quantum neural network, 2023. arXiv: 2304.04666 [quant-ph].

[57] F. Arute *et al.*, Quantum supremacy using a programmable superconducting processor, Nature **574** (7779), 505–510, 2019.

[58] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, and R. Wisnieff, Leveraging secondary storage to simulate deep 54-qubit sycamore circuits, 2019. arXiv: 1910.09534 [quant-ph].

[59] F. Pan, K. Chen, and P. Zhang, Solving the sampling problem of the sycamore quantum circuits, Phys. Rev. Lett. **129** (9), 090502, 2022.

[60] Y. Kim *et al.*, Evidence for the utility of quantum computing before fault tolerance, Nature **618** (7965), 500–505, 2023.

[61] N. Herrmann, D. Arya, M. W. Doherty, A. Mingare, J. C. Pillay, F. Preis, and S. Prestel, *Quantum utility – definition and assessment of a practical quantum advantage*. 2023, pp. 162–174.

[62] A. Kandala, K. Temme, A. D. Córcoles, A. Mezzacapo, J. M. Chow, and J. M. Gambetta, Error mitigation extends the computational reach of a noisy quantum processor, Nature **567** (7749), 491–495, 2019.

[63] E. van den Berg, Z. K. Minev, A. Kandala, and K. Temme, Probabilistic error cancellation with sparse pauli–lindblad models on noisy quantum processors, Nature Physics **19** (8), 1116–1121, 2023.

[64] D. Bluvstein *et al.*, Logical quantum processor based on reconfigurable atom arrays, Nature, 2023.

[65] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Surface codes: Towards practical large-scale quantum computation, Phys. Rev. A **86** (3), 032324, 2012.

[66] J. Gambetta, Ibm quantum system two: The era of quantum utility is here, 2023.

[67] S. Krinner *et al.*, Realizing repeated quantum error correction in a distance-three surface code, Nature **605** (7911), 669–674, 2022.

[68] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review **65** (6), 386–408, 1958.

[69] M. Minsky and S. Papert, An introduction to computational geometry, Cambridge tiass., HIT **479** (480), 104, 1969.

[70] A. L. Fradkov, Early history of machine learning, IFAC-PapersOnLine **53** (2), 1385–1390, 2020, 21st IFAC World Congress.

[71] K. Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, Biological Cybernetics **36** (4), 193–202, 1980.

[72] S. Linnainmaa, Taylor expansion of the accumulated rounding error, BIT Numerical Mathematics **16** (2), 146–160, 1976.

[73] M. Bojarski *et al.*, End to End Learning for Self-Driving Cars, arXiv e-prints, 2016. arXiv: 1604.07316 [cs.CV].

[74] S. Shahriar and K. Hayawi, Let's have a chat! a conversation with chatgpt: Technology, applications, and limitations, Artificial Intelligence and Applications, 2023.

[75] P. Kaur, K. Krishan, S. K. Sharma, and T. Kanchan, Facial-recognition algorithms: A literature review, Medicine, Science and the Law **60** (2), 131–139, 2020, PMID: 31964224. eprint: https://doi.org/10.1177/0025802419893168.

[76] R. D. King *et al.*, The automation of science, Science **324** (5923), 85–89, 2009.

[77] M. Schmidt and H. Lipson, Distilling free-form natural laws from experimental data, Science **324** (5923), 81–85, 2009.

[78] J. Carrasquilla and R. G. Melko, Machine learning phases of matter, Nature Physics **13** (5), 431–434, 2017.

[79] G. Torlai and R. G. Melko, Neural decoder for topological codes, Phys. Rev. Lett. **119** (3), 030501, 2017.

[80] S. Krastanov and L. Jiang, Deep neural network probabilistic decoder for stabilizer codes, Scientific Reports **7** (1), 11003, 2017.

[81] S. Varsamopoulos, B. Criger, and K. Bertels, Decoding small surface codes with feed-forward neural networks, Quantum Science and Technology **3** (1), 015004, 2017.

[82] V. Mnih *et al.*, Human-level control through deep reinforcement learning, Nature **518** (7540), 529–533, 2015.

[83] D. Silver *et al.*, Mastering the game of go with deep neural networks and tree search, Nature **529** (7587), 484–489, 2016.

[84] D. Silver *et al.*, Mastering the game of go without human knowledge, Nature **550** (7676), 354–359, 2017.

[85] H. P. Nautrup, N. Delfosse, V. Dunjko, H. J. Briegel, and N. Friis, Optimizing Quantum Error Correction Codes with Reinforcement Learning, Quantum **3**, 215, 2019.

[86] W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, The bulletin of mathematical biophysics **5** (4), 115–133, 1943.

[87] A. G. Ivakhnenko and V. G. Lapa, *CYBERNETIC PREDICTING DEVICES*. 1966.

[88] M. A. Nielsen, Neural networks and deep learning, Determination Press, 2015.

[89] S. Basodi, C. Ji, H. Zhang, and Y. Pan, Gradient amplification: An efficient way to train deep neural networks, Big Data Mining and Analytics **3** (3), 196–207, 2020.

[90] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals and Systems **2** (4), 303–314, 1989.

[91] K. Fukushima, Visual feature extraction by a multilayered network of analog threshold elements, IEEE Transactions on Systems Science and Cybernetics **5** (4), 322–333, 1969.

[92] J. Han and C. Moraga, *The influence of the sigmoid function parameters on the speed of backpropagation learning*, J. Mira and F. Sandoval, Eds. Springer Berlin Heidelberg, 1995, pp. 195–201.

[93] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[94] X. Glorot, A. Bordes, and Y. Bengio, *Deep Sparse Rectifier Neural Networks*, G. Gordon, D. Dunson, and M. Dudík, Eds. PMLR, 2011, vol. 15, pp. 315–323.

[95] P. Ramachandran, B. Zoph, and Q. V. Le, Searching for activation functions, 2017. arXiv: `1710.05941 [cs.NE]`.

[96] M. Valueva, N. Nagornov, P. Lyakhov, G. Valuev, and N. Chervyakov, Application of the residue number system to reduce hardware costs of the convolutional neural network implementation, Mathematics and Computers in Simulation **177**, 232–243, 2020.

[97] M. Hashemi, Enlarging smaller images before inputting into convolutional neural network: Zero-padding vs. interpolation, Journal of Big Data **6** (1), 98, 2019.

[98] W. Zhang, K. Itoh, J. Tanida, and Y. Ichioka, Parallel distributed processing model with local space-invariant interconnections and its optical architecture, Appl. Opt. **29** (32), 4790–4797, 1990.

[99] H. Robbins and S. Monro, A Stochastic Approximation Method, The Annals of Mathematical Statistics **22** (3), 400–407, 1951.

[100] A. Amini, A. Soleimany, S. Karaman, and D. Rus, Spatial uncertainty sampling for end-to-end control, 2018. arXiv: `1805.04829 [cs.AI]`.

[101] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, 2017. arXiv: `1412.6980 [cs.LG]`.

[102] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning representations by back-propagating errors, Nature **323** (6088), 533–536, 1986.

[103] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, *On the importance of initialization and momentum in deep learning*, S. Dasgupta and D. McAllester, Eds. PMLR, 2013, vol. 28, pp. 1139–1147.

[104] S. N. Hinton G. and S. K, Lecture 6a: Overview of mini-batch gradient descent, 2020.

[105] L. Ciampiconi, A. Elwood, M. Leonardi, A. Mohamed, and A. Rozza, A survey and taxonomy of loss functions in machine learning, 2023. arXiv: 2301.05579 [cs.LG].

[106] E. Gordon-Rodriguez, G. Loaiza-Ganem, G. Pleiss, and J. P. Cunningham, Uses and abuses of the cross-entropy loss: Case studies in modern deep learning, 2020. arXiv: 2011.05231 [stat.ML].

[107] D. Michie and R. A. Chambers, *BOXES: AN EXPERIMENT IN ADAPTIVE CONTROL*. 2013.

[108] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. A Bradford Book, 2018.

[109] T. Schaul, D. Horgan, K. Gregor, and D. Silver, *Universal Value Function Approximators*, F. Bach and D. Blei, Eds. PMLR, 2015, vol. 37, pp. 1312–1320.

[110] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, High-dimensional continuous control using generalized advantage estimation, 2018. arXiv: 1506.02438 [cs.LG].

[111] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal policy optimization algorithms, CoRR **abs/1707.06347**, 2017. arXiv: 1707.06347.

[112] V. Dasagi, J. Bruce, T. Peynot, and J. Leitner, Ctrl-z: Recovering from instability in reinforcement learning, 2019. arXiv: 1910.03732 [cs.LG].

[113] S. Thrun and A. Schwartz, *Issues in Using Function Approximation for Reinforcement Learning*. 1999.

[114] O. Anschel, N. Baram, and N. Shimkin, Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning, 2017. arXiv: 1611.01929 [cs.AI].

[115] S. Fujimoto, H. van Hoof, and D. Meger, Addressing function approximation error in actor-critic methods, 2018. arXiv: 1802.09477 [cs.AI].

[116] S. Thrun, *Efficient Exploration In Reinforcement Learning.* 1992.

[117] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, Deep Exploration via Bootstrapped DQN, arXiv e-prints, 2016. arXiv: 1602.04621 [cs.LG].

[118] M. Fortunato *et al.*, Noisy Networks for Exploration, arXiv e-prints arXiv:1706.10295, arXiv:1706.10295, 2017. arXiv: 1706.10295 [cs.LG].

[119] M. Jaderberg *et al.*, Population Based Training of Neural Networks, arXiv e-prints arXiv:1711.09846, arXiv:1711.09846, 2017. arXiv: 1711.09846 [cs.LG].

[120] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, arXiv e-prints, 2018.

[121] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, *Asynchronous Methods for Deep Reinforcement Learning*, M. F. Balcan and K. Q. Weinberger, Eds. PMLR, 2016, vol. 48, pp. 1928–1937.

[122] T. G. Karimpanal and R. Bouffanais, Self-organizing maps for storage and transfer of knowledge in reinforcement learning, Adaptive Behavior **27** (2), 111–126, 2019.

[123] D. Gottesman, Stabilizer codes and quantum error correction, 1997. arXiv: quant-ph/9705052 [quant-ph].

[124] A. Kitaev, Fault-tolerant quantum computation by anyons, Annals of Physics **303** (1), 2–30, 2003.

[125] Y. Tomita and K. M. Svore, Low-distance surface codes under realistic quantum noise, Phys. Rev. A **90** (6), 062320, 2014.

[126] A. Bookstein, V. Kulyukin, and T. Raita, Generalized hamming distance, Information Retrieval **5**, 2002.

[127] K. Svore, A. Cross, I. Chuang, and A. Aho, A flow-map model for analyzing pseudothresholds in fault-tolerant quantum computing, Quantum Information and Computation **6**, 193–212, 2006.

[128] F. Jazaeri, A. Beckers, A. Tajalli, and J.-M. Sallese, A review on quantum computing: Qubits, cryogenic electronics and cryogenic mosfet physics, 2019. arXiv: 1908.02656 [quant-ph].

[129] J. Roffe, Quantum error correction: An introductory guide, Contemporary Physics **60** (3), 226–245, 2019.

[130] W. K. Wootters and W. H. Zurek, A single quantum cannot be cloned, Nature **299** (5886), 802–803, 1982.

[131] P. W. Shor, Scheme for reducing decoherence in quantum computer memory, Phys. Rev. A **52** (4), R2493–R2496, 1995.

[132] E. Knill, R. Laflamme, A. Ashikhmin, H. Barnum, L. Viola, and W. H. Zurek, Introduction to quantum error correction, 2002. arXiv: quant-ph/0207170 [quant-ph].

[133] C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross, Topological and subsystem codes on low-degree graphs with flag qubits, Phys. Rev. X **10** (1), 011022, 2020.

[134] A. G. Fowler, A. M. Stephens, and P. Groszkowski, High-threshold universal quantum computation on the surface code, Phys. Rev. A **80** (5), 052312, 2009.

[135] A. Karim, A. Mishra, M. H. Newton, and A. Sattar, Machine learning interpretability: A science rather than a tool, 2018. arXiv: 1807.06722 [cs.LG].

[136] D. Kroese, T. Taimre, and Z. Botev, *Handbook of Monte Carlo Methods*. 2011.

[137] G. E. P. Box and M. E. Muller, A note on the generation of random normal deviates, Annals of Mathematical Statistics **29**, 610–611, 1958.

[138] C. Wang, J. Harrington, and J. Preskill, Confinement-higgs transition in a disordered gauge theory and the accuracy threshold for quantum memory, Annals of Physics **303** (1), 31–58, 2003.

[139] A. Sorge, Pyfssa 0.7.6, version 0.7.6, 2015.

[140] M.-H. Hsieh and F. ç. Le Gall, Np-hardness of decoding quantum error-correction codes, Phys. Rev. A **83** (5), 052331, 2011.

[141] P. Iyer and D. Poulin, Hardness of decoding quantum stabilizer codes, IEEE Transactions on Information Theory **61**, 2013.

[142] P. Iyer and D. Poulin, Hardness of decoding quantum stabilizer codes, IEEE Transactions on Information Theory **61** (9), 5209–5223, 2015.

[143] J. Edmonds, Paths, trees, and flowers, Canadian Journal of Mathematics **17**, 449–467, 1965.

[144] V. Kolmogorov, Blossom v: A new implementation of a minimum cost perfect matching algorithm, Mathematical Programming Computation **1** (1), 43–67, 2009.

[145] O. Higgott, PyMatching: A Python package for decoding quantum codes with minimum-weight perfect matching, arXiv e-prints, 2021. arXiv: 2105.13082 [quant-ph].

[146] O. Higgott and C. Gidney, Sparse blossom: Correcting a million errors per core second with minimum-weight matching, arXiv preprint arXiv:2303.15933, 2023.

[147] A. M. Stephens, Fault-tolerant thresholds for quantum error correction with the surface code, Phys. Rev. A **89** (2), 022321, 2014.

[148] A. Honecker, M. Picco, and P. Pujol, Universality class of the nishimori point in the 2d $\pm J$ random-bond ising model, Phys. Rev. Lett. **87** (4), 047201, 2001.

[149] G. Duclos-Cianci and D. Poulin, Fast decoders for topological quantum codes, Phys. Rev. Lett. **104** (5), 050504, 2010.

[150] N. Delfosse and N. H. Nickerson, Almost-linear time decoding algorithm for topological codes, Quantum **5**, 595, 2021.

[151] B. A. Galler and M. J. Fisher, An improved equivalence algorithm, Commun. ACM **7** (5), 301–303, 1964.

[152] R. E. Tarjan, Efficiency of a good but not linear set union algorithm, J. ACM **22** (2), 215–225, 1975.

[153] N. Delfosse and G. Zémor, Linear-time maximum likelihood decoding of surface codes over the quantum erasure channel, Phys. Rev. Res. **2** (3), 033042, 2020.

[154] T. Wagner, H. Kampermann, and D. Bruß, Symmetries for a high-level neural decoder on the toric code, Phys. Rev. A **102** (4), 042411, 2020.

[155] N. Delfosse, Hierarchical decoding to reduce hardware requirements for quantum computing, 2020. arXiv: 2001.11427 [quant-ph].

[156] H. Bombin, R. S. Andrist, M. Ohzeki, H. G. Katzgraber, and M. A. Martin-Delgado, Strong resilience of topological codes to depolarization, Phys. Rev. X **2** (2), 021004, 2012.

[157] D. S. Wang, A. G. Fowler, A. M. Stephens, and L. C. L. Hollenberg, Threshold error rates for the toric and planar codes, Quantum Info. Comput. **10** (5), 456–469, 2010.

[158] D. Fitzek, M. Eliasson, A. F. Kockum, and M. Granath, Deep q-learning decoder for depolarizing noise on the toric code, Phys. Rev. Res. **2** (2), 023230, 2020.

[159] S. Varona and M. A. Martin-Delgado, Determination of the semion code threshold using neural decoders, Phys. Rev. A **102** (3), 032411, 2020.

[160] L. Domingo Colomer, M. Skotiniotis, and R. Muñoz-Tapia, Reinforcement learning for optimal error correction of toric codes, Physics Letters A **384** (17), 126353, 2020.

[161] J. Gambetta, Ibm's roadmap for scaling quantum technology, 2022.

[162] P. Das, C. A. Pattison, S. Manne, D. Carmean, K. Svore, M. Qureshi, and N. Delfosse, A scalable decoder micro-architecture for fault-tolerant quantum computing, 2020. arXiv: 2001.06598 [quant-ph].

[163] N. P. Jouppi *et al.*, In-datacenter performance analysis of a tensor processing unit, SIGARCH Comput. Archit. News **45** (2), 1–12, 2017.

[164] M. Zhang, X. Ren, G. Xi, Z. Zhang, Q. Yu, F. Liu, H. Zhang, S. Zhang, and Y.-C. Zheng, A scalable, fast and programmable neural decoder for fault-tolerant quantum computation using surface codes, 2023. arXiv: 2305.15767 [quant-ph].

[165] T. Ben-Nun and T. Hoefler, Demystifying parallel and distributed deep learning: An in-depth concurrency analysis, 2018. arXiv: 1802.09941 [cs.LG].

[166] A. Fowler, Minimum weight perfect matching of fault-tolerant topological quantum error correction in average o(1) parallel time, Quantum Information and Computation **15**, 145–158, 2014.

[167] R. Durrett, *Probability: Theory and Examples*, 5th ed. Cambridge University Press, 2019.

[168] R. E. Kalman, A New Approach to Linear Filtering and Prediction Problems, Journal of Basic Engineering **82**, 35–45, 1960.

[169] Y. Bar-Shalom, X. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation: Theory, Algorithms and Software*. 2001.

[170] R. E. Kálmán, Contributions to the theory of optimal control, 1960.

[171] Y. Aharonov and L. Vaidman, Properties of a quantum system during the time interval between two measurements, Phys. Rev. A **41** (1), 11–20, 1990.

[172] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 10th. Cambridge University Press, 2011.

[173] H. Margenau, Measurements and quantum states: Part i, Philosophy of Science **30** (1), 1–16, 1963.

[174] S. L. Adler, D. C. Brody, T. A. Brun, and L. P. Hughston, Martingale models for quantum state reduction, Journal of Physics A: Mathematical and General **34** (42), 8795, 2001.

[175] J. Humpherys, P. Redd, and J. West, A fresh look at the kalman filter, SIAM Review **54**, 801–823, 2012.

[176] J. Suddath, R. Kidd, and A. Reinhold, *A Linearized Error Analysis of Onboard Primary Navigation Systems for the Apollo Lunar Module*. National Aeronautics and Space Administration, 1967.

[177] D. Gaylor and G. Lightsey, Gps/ins kalman filter design for spacecraft operating in the proximity of the international space station, 2003.

[178] A. Lindquist, On feedback control of linear stochastic systems, SIAM Journal on Control **11** (2), 323–343, 1973.

[179] P. Lancaster and L. Rodman, *Algebraic Riccati Equations*. Clarendon Press, 1995.

[180] Z. T. Wang, Y. Ashida, and M. Ueda, Deep Reinforcement Learning Control of Quantum Cartpoles, Phys. Rev. Lett. **125**, 100401, 2020.

[181] A. C. Doherty and K. Jacobs, Feedback control of quantum systems using continuous state estimation, Phys. Rev. A **60** (4), 2700–2711, 1999.

[182] E. Hairer and C. Lubich, *Numerical Analysis of Ordinary Differential Equations*. 2015, pp. 1053–1059.

[183] M. Fu, Reinforcement learning approach to estimation in linear systems, 2022. arXiv: 2205.03504 [eess.SY].

[184] B. Barber *et al.*, A real-time, scalable, fast and highly resource efficient decoder for a quantum computer, 2023. arXiv: 2309.05558 [quant-ph].

[185] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, Finetuned language models are zero-shot learners, 2022. arXiv: 2109.01652 [cs.CL].

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

[187] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, Openai gym, 2016. eprint: arXiv:1606.01540.

Acknowledgement

First and foremost, I would like to express my deepest gratitude to my supervisor, Simon Trebst, for mentoring me over the years and providing me with an interesting and challenging PhD topic. His guidance and the opportunities he has provided have supported both my academic and personal growth throughout my journey. I also want to thank David Gross and Lucas Labadie for kindly agreeing to being the second referee and chairman of this thesis.

I also want to thank the various collaborators who have worked side by side with me. I would like to thank Chae-Yeun Park for the interesting discussions on quantum error correction and machine learning, and Mark Rudner for all his help on the Quantum Cartpole project. A special thanks goes to Evert van Niuewenburg, who not only offered his help in countless video chats, but also invited me to spend 6 months in Copenhagen, where I was received with utmost hospitality.

None of these opportunities would have been possible without the support of the CRC183 and ML4Q, which not only paid for my position, but also allowed me to attend several conferences and a long stay in Copenhagen. I would like to thank everyone involved, with a special thanks to Rita Kottmeier for her support in organising the stay. Furthermore, I am grateful for the administrative support provided by Andreas Sindermann, Petra Neubauer-Guenther and the Bonn-Cologne Graduate School of Physics and Astronomy.

I would also like to acknowledge the indispensable contributions of the RRZK in Cologne and the Jülich Forschungszentrum. Their provision of access to the CHEOPS and JUWELS supercomputers has been instrumental in allowing this thesis to be completed. The availability of these resources has contributed significantly to the depth and quality of the research performed.

I would like to express my sincere gratitude to all my colleagues over the years for fostering a welcoming and open working environment defined by mutual support. This atmosphere encouraged not only insightful scientific discussions, but also moments of shared laughter and office banter that made the journey truly enjoyable. Special thanks go to Christoph, Sagar, Lasse, Theo who helped me proofread this thesis.

Finally, I am grateful to my friends, whose companionship during the challenging times of the pandemic was invaluable. Also I am grateful to my family for their unconditional support and encouragement throughout this academic pursuit. Their belief in me has been a constant source of motivation.

Erklärung zur Dissertation

Hiermit versichere ich an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel und Literatur angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind als solche kenntlich gemacht. Ich versichere an Eides statt, dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie - abgesehen von unten angegebenen Teilpublikationen und eingebundenen Artikeln und Manuskripten - noch nicht veröffentlicht worden ist sowie, dass ich eine Veröffentlichung der Dissertation vor Abschluss der Promotion nicht ohne Genehmigung des Promotionsausschusses vornehmen werde. Die Bestimmungen dieser Ordnung sind mir bekannt. Darüber hinaus erkläre ich hiermit, dass ich die Ordnung zur Sicherung guter wissenschaftlicher Praxis und zum Umgang mit wissenschaftlichem Fehlverhalten der Universität zu Köln gelesen und sie bei der Durchführung der Dissertation zugrundeliegenden Arbeiten und der schriftlich verfassten Dissertation beachtet habe und verpflichte mich hiermit, die dort genannten Vorgaben bei allen wissenschaftlichen Tätigkeiten zu beachten und umzusetzen. Ich versichere, dass die eingereichte elektronische Fassung der eingereichten Druckfassung vollständig entspricht.

Köln, der 25. Oktober 2024

(Kai Meinerz)

(Eine Liste der Teilpublikationen findet sich auf der nächsten Seite.)

Publications

(Peer-reviewed)

[P1] **K. Meinerz**, S. Trebst, M. Rudner, and E. van Nieuwenburg, The quantum cartpole: A benchmark environment for non-linear reinforcement learning, 2023. arXiv: 2311.00756 [quant-ph].

[P2] **K. Meinerz**, C.-Y. Park, and S. Trebst, Scalable neural decoder for topological surface codes, Phys. Rev. Lett. **128** (8), 080505, 2022.

(Selected open-source publications)

[S1] C.-Y. Park and **K. Meinerz**, Open-source c++ implementation of the union-find decoder, https://github.com/chaeyeunpark/UnionFind, 2020.

[S2] **K. Meinerz**, S. Trebst, M. Rudner, and E. van Nieuwenburg, Dataset underpinning: "the quantum cartpole", version 1.0.0, 2023.

[S3] **K. Meinerz**, C.-Y. Park, and S. Trebst, Dataset underpinning: "Scalable Neural Decoder for Topological Surface Codes", 2023.

[S4] **K. Meinerz**, Quantum cartpole environment, version v0.1.0, 2023.

Quantum feedback control is a field of research that deals with the manipulation of quantum systems towards a goal, based on continuous observation of the systems. These control strategies have been identified as an essential part of the development of future quantum technologies. An example of this is the formulation of quantum error correction strategies for fault-tolerant quantum computing. In this thesis, we focus on the use of machine learning based approaches to find quantum control feedback strategies.

In a first study, we develop a control strategy for quantum error correction on topological surface codes in the form of a hierarchical decoder. Using a combination of machine learning and combinational decoding, we were able to demonstrate scalable decoding. We achieved nearly linear-time scaling, while still maintaining a high precision decoding comparable to state-of-the-art conventional decoding strategies.

In a second study, we investigate the use of reinforcement learning for quantum feedback control. Since reinforcement learning encompasses model-free approaches, it is a prime candidate for finding robust strategies that are not hindered by unknown uncertainties in the experimental system. A well-known problem of reinforcement learning is the sometimes unstable training, caused by the encountered “exploration versus exploitation" dilemma during the process. Therefore, we have implemented a toy model, the quantum cartpole, designed to serve as a benchmark environment for the development of reinforcement learning based quantum feedback strategies, allowing to demonstrate the ability of reinforcement to find novel strategies that outperform conventional control strategies in highly nonlinear systems.

University of Cologne

thp Institute for Theoretical Physics University of Cologne