Universiteit Leiden

Reinforcement Learning for Quantum Error Correction

Alma Rosenmann

Supervisors:
Dr. Thomas Moerland & Lindsay Spoor

Bachelor Thesis– Data Science and Artificial Intelligence

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

July 8, 2026

Abstract

Quantum error correction (QEC) is a prerequisite for large-scale quantum computing. In this work, we investigate the use of reinforcement learning (RL) to perform QEC on a rotated surface code. We formulate the problem as a partially observable Markov decision process (POMDP), in which an agent observes the syndromes and applies qubit corrections to prevent a logical error from occurring.

We study the effect of providing the agent with a history of past syndrome observations on its ability to decode the rotated surface code and prevent a logical error from occurring. This is done using two classes of RL methods: a tabular method (Q-learning) and a function approximation method (Proximal Policy Optimization). We find that, while both methods benefit from additional history, the tabular agent is limited by the exponential growth of the history-including state-action space, whereas PPO is able to generalize across similar observations and therefore make better use of the additional information to increase the survival length before a logical error occurs.

Acknowledgments

I would like to thank my first supervisor, Dr. Thomas Moerland, for his guidance, support and encouragement throughout the making of this work, for providing me with his insights and vast knowledge about the subjects of this thesis, and for giving valuable feedback. Furthermore, I would like to thank my second supervisor, Lindsay Spoor, who helped me by answering all of the questions concerning the content and writing of this thesis. I would also like to sincerely thank Jan Krzywda for providing the code for the environment, and for guiding the quantum aspect of this paper.

Contents

1 Introduction

The field of quantum computing (QC) has experienced significant growth in recent years, achieving advances in solving computational problems that are infeasible or have long run times on classical (non-quantum) computers [BBB+26]. As quantum hardware continues to improve, the number of available qubits (quantum bits) is also expected to increase substantially.

This significant growth in QC has, however, due to its high degree of instability, led to new performance-related problems and limitations that did not exist in the realm of classical computers [Spo24]. Therefore, we bring forward the question of how we are able to counteract this instability. Similarly to classical computers, for which error correction has long been established, quantum error correction (QEC) was introduced in 1995 in order to address this problem, and is since considered a requirement for the development of large-scale, controllable quantum computers [DMN13].

Nevertheless, QEC is not as straightforward as classical error correction, where a single bit is copied multiple times. This is due to a number of quantum-specific limitations, all of which revolve around the fragility of the quantum state [CPG23]. That is, an unknown quantum state cannot be perfectly copied, and cannot be measured directly without collapsing the superposition (as will be further explained in Section 2) on which the computation relies. Furthermore, quantum errors are of two different kinds: bit-flip and phase-flip errors. To address these limitations, QEC does not measure the qubits directly, but rather introduces a set of parity checks, known as stabilizers, whose measurement reveals where errors have occurred without collapsing the qubits [Got97]. In this work, we consider the rotated surface code (topological QEC as visualized in Section 2) and focus specifically on the correction of bit-flip errors.

Given a syndrome, which is the measure of the previously mentioned stabilizers, the task of inferring which physical errors occurred and how to undo them is performed by a decoder [Spo24]. The standard choice is minimum-weight perfect matching (MWPM), which considers decoding as a graph-matching problem. MWPM is fast and effective, yet is hand-designed for a fixed, assumed noise model and does not adapt to noise change. Since real hardware is subject to drifting noise (as will be described in Section 4) and is only partially known, these assumptions do not hold, which is one of the main motivations for learning-

based decoders. Supervised neural-network decoders, such as Google's AlphaQubit, train a network to map syndromes directly to corrections and have been shown to outperform matching-based decoders on real hardware [BSH+24]. Such methods, however, require large labeled datasets of syndrome-error pairs.

An alternative approach that avoids the requirement for a labeled dataset, and is also not reliant on a known noise model is reinforcement learning (RL). In RL, an agent learns a correction policy directly from interaction with the QEC environment. RL decoders have been studied for both the surface and toric codes [SKvNE20, AJLG19, Spo24], and it is this learning-from-interaction property that makes RL a promising approach to QEC in a setting where the noise is non-stationary.

In this work we therefore take an RL approach to perform QEC. We treat the lattice of physical qubits as the environment of a game, in which an agent applies corrections by flipping individual physical qubits, with the goal of protecting the encoded information for as long as possible. The performance of such an agent is measured by its survival length, the number of time-steps for which it avoids a logical error. Because the qubits cannot be measured directly, the agent never observes the true underlying error configuration, but only the syndrome. QEC can therefore be formulated as a partially observable Markov decision process (POMDP). Many studies of QEC decoders, as previously stated have a fixed-noise assumption. In this paper, in order to consider the more realistic setting, the per-qubit physical error rate drifts over time according to a simplified Ornstein-Uhlenbeck process [CMR22], with an initial value of 0.01, which is roughly the error rate expected of current real-world physical qubits [Mic24].

Given the setup of a physical qubit lattice, many distinct error configurations produce the same syndrome. Therefore, a single syndrome observation is insufficient to identify the true underlying state with certainty. A natural way to reduce this ambiguity is to provide the agent with previous observations, which we refer to as 'history', from which it may reconstruct a better approximation of the hidden state. The idea is that a lack of syndrome does not truly guarantee that there is no underlying physical error, since the syndromes of separate errors can cancel one another out. Therefore, given the history of previously observed syndromes, the agent may be better able to judge whether a lack of syndrome is simply caused by a secondary physical error, which should generally reveal itself by triggering a different syndrome.

The main research question of this thesis is: **To what extent can previous syndrome observations improve a reinforcement learning agent's performance as a quantum error correction decoder?**

We study this question across two distinct classes of RL methods, namely, a tabular method of Q-learning, which stores a separate value for every state-action pair [SB18], and a function-approximation method, Proximal Policy Optimization (PPO), which parametrizes its policy with a neural network and is therefore a form of deep reinforcement learning [SWD+17]. Comparing the two allows us to assess the role that generalization plays in exploiting observation history.

This work makes the following two contributions. First, and most importantly, we measure the effect of observation history on the performance of an RL agent decoding QEC, by varying the number of previous observations supplied to the agent. Secondly, we compare the tabular and function-approximation methods on this question. We then show that each exploits history to very different extents. The tabular agent is limited by the exponential growth of its state space, and the function-approximation agent is able to generalize across observations and is thereby able to benefit from the additional history.

The following seven sections of this thesis are structured as follows. Section 2 introduces the necessary theory of QEC and RL and formalises the problem as a partially observable Markov Decision Process. Section 3 discusses related work on classical and learning-based decoders and situates our contribution within it. Section 4 describes two methods, Q-learning and PPO, in detail, and Section 5 specifies the configuration of the environment and of both algorithms. Section 6 presents our results, first for the tabular setting, then for PPO, and finally a comparison of all agents. Section 7 discusses our findings and the limitations of our approach, and Section 8 concludes and outlines directions for future work.

2 Background and theory

This section introduces the main terms and concepts which will be referenced throughout the rest of the thesis. We first introduce quantum error correction (QEC) and explain why it is necessary, before formalizing the QEC problem as a Markov decision process (MDP) so that it can be approached with RL. We then explain the relevant RL concepts that the

methods in Section 4 build on.

2.1 Quantum Error Correction

Let us begin by defining quantum errors and discussing why quantum error correction (QEC) is of interest. To understand QEC, we first need to consider what quantum information is, using the famous thought experiment of Schrödinger's cat [Ion17].

The idea of Schrödinger's cat is as follows: Suppose there is, in a box, a cat, with an excited atom in state $|0\rangle$ and a detector, which is coupled to this atom [Ion17]. In the case that the excited atom falls to its ground state $|1\rangle$, a poison is released by the detector, which kills the cat. This, all within a black box, such that there is no indication what state the cat is in to an observer.

This thought experiment illustrates the property of superposition, where before observation, the system does not occupy a single definite state, but a combination of both, as described in Equation (1). The same equation defines the basic unit of quantum information, which is called a qubit,

$$|\Psi_f\rangle = \alpha|0\rangle|\text{alive}\rangle + \beta|1\rangle|\text{dead}\rangle \tag{1}$$

[Ion17], where $|\Psi\rangle$ is the joint state, or superposition, of the system after the atom and cat have become coupled. Subscript f marks it as the final state, namely, after the interaction, and before observation. $|0\rangle$, and $|1\rangle$ are the two states of the atom in the box, where $|0\rangle$ means it is not decayed, and $|1\rangle$ means it has decayed. $|\text{alive}\rangle, |\text{dead}\rangle$ are the two possible states of the cat. α, β are the complex probability amplitudes, where $|\alpha|^2$ is the probability of finding the system in the alive branch, and $|\beta|^2$ the probability of the dead branch, such that $|\alpha|^2 + |\beta|^2 = 1$.

Returning to our definition of quantum information, we use this equation to define the state of a qubit. While classical computers store information in bits, which can take the value 0 or 1, quantum computers use qubits, which can exist in a superposition of 0 and 1 at the same time [CPG23]. This is analogous to Schrödinger's cat being both alive and dead before observation. The closed box corresponds to the qubit being isolated from measurement. A qubit can only exist in a superposition of 0 and 1 until it is measured. Measurement forces it to assume one outcome, just as opening the box reveals the cat's state. This way of storing information allows quantum systems to represent information

much more richly, enabling certain computations to be performed far more efficiently [SS24].

However, this power comes at the cost that quantum states are very fragile [Spo24]. Very small interactions with the environment can already disturb a qubit and introduce errors, which makes the information unreliable. This fragility is the reason that quantum error correction is needed.

Similarly to classical error correction, QEC protects information by redundantly encoding it across multiple qubits, which we refer to as physical qubits. Instead of storing information in a single physical qubit, quantum error correction distributes it across several physical qubits. The qubit carrying the actual information is called a logical qubit [Got97]. Then, the logical qubit represents the abstract information, while the physical qubits are the hardware-level qubits that jointly store it. Although an individual physical qubit is fragile, encoding information across many physical qubits makes it more robust against noise.

However, QEC must address several additional challenges. First, the no-cloning theorem states that an unknown quantum state cannot be perfectly copied [CPG23]. As a result, the information must be distributed across physical qubits rather than duplicated. Second, direct measurement causes wave function collapse, destroying the quantum superposition required for computation. Third, quantum systems are subject to two main types of errors; bit-flip errors and phase-flip errors. A bit-flip error changes the qubit state in a way analogous to a bit flip in a classical computer. The final problem is that quantum noise is not purely discrete, and even small physical disturbances can slightly rotate a qubit's state, leading to gradual errors that accumulate over time [CPG23].

QEC operates by defining a set of consistency checks for the encoded qubits, known as stabilizers [Got97]. By measuring whether these checks still hold, we can detect where noise has affected the system without disturbing the stored quantum information. The pattern of detected noise is called a syndrome. X-stabilizers are used to detect phase-flip errors, while Z-stabilizers are used to detect bit-flip errors [TFG+25]. Rather than measuring individual qubits directly, they check collective properties of neighbouring physical qubits [Got97]. This allows errors to be identified without disturbing the logical quantum information stored in the code.

Figure 1 shows the simulation of quantum error correction, where the circles represent the

physical qubits, the blue squares represent the X-stabilizers, and the red squares represent the Z-stabilizers.

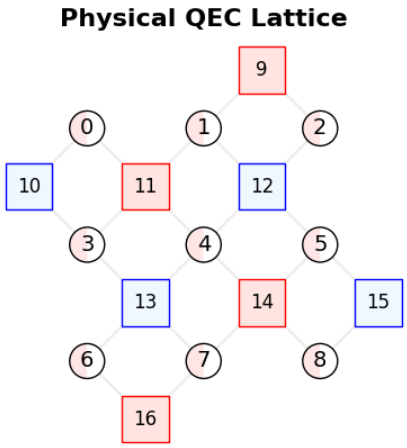


Figure 1: Simulated rotated surface code at distance $d = 3$. The circles are the $d^2 = 9$ physical qubits, and the coloured squares are the stabilizer ancillas. Red squares are the Z-stabilizers (bit-flip / X-error detectors) and blue squares the X-stabilizers (phase-flip / Z-error detectors). There are $(d^2 - 1)/2 = 4$ of each stabilizer type.

The syndrome is interpreted by a decoder, the standard decoder being minimum-weight perfect matching (MWPM) [Spo24]. This method, however, is hand-designed for a fixed, assumed noise model (error rate and error distribution) and does not adapt when the noise changes. This motivates learning-based decoders, and reinforcement learning in particular, which can learn a correction policy directly from interaction without assuming a fixed noise model. We discuss this approach in Section 2.3.

2.2 Markov Decision Process Formulation

Let us now consider this problem as a game for an RL agent. We therefore first formalize the reinforcement learning setting of QEC as a Markov Decision Process (MDP). QEC is a natural fit for this framework since error correction is inherently sequential. Noise accumulates over repeated rounds and a correction at one round changes the state faced at the next, so decisions cannot be treated in isolation. At each round, the decoder takes an

action, choosing which qubit to correct, and there is a natural goal of keeping the logical qubit intact for as long as possible, that can be encoded as a reward.

MDPs are a model used for sequential decision making under uncertainty, defined by a state space $\mathcal{S}$, action space $\mathcal{A}$, reward space R, and state transitions P [SBB+24].

We must note that an MDP does not require the transition probabilities to be precisely known [SBB+24], which is why we are also able to use them for the QEC problem. Here, we only have access to a simulator from which transitions can be sampled. We later discuss how we make use of this with a model-free reinforcement learning (MFRL) method, Q-learning, which learns directly from sampled interactions with the environment rather than from an explicit model.

Furthermore, because of measurement collapse and the use of syndromes rather than direct measurements, this problem is formulated as a partially observable Markov decision process (POMDP). In that case, we introduce an observation space $\mathcal{O}$ and an observation function Z [Spa12]. The transition probability and observation function depend on the final configuration chosen for the QEC environment.

Let us formalize the tuple

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma), \tag{2}$$

where $\mathcal{S} : \{0, 1\}^N$ is the state space consisting of the physical qubits, where 0 denotes a correct qubit state, 1 denotes a flipped state, and N is the number of physical qubits, equal to the squared dimension, d^2.

$\mathcal{A}$: flip 0, flip 1, ..., no-operation, truncate is a finite action space. The agent may apply an X-correction to any of the d^2 physical qubits, take no action, or truncate, giving $|\mathcal{A}| = d^2 + 2 = 11$ actions at $d = 3$.

$P(s' \mid s, a)$ is the transition probability to the next state given the current state s and action a, which is unknown in our environment. $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. It is important to note that the reward formulation will determine whether the agent is able to learn over time. In our environment, we define a training reward, and an evaluation reward. The difference is that during training, we want to punish actions which result in more physical errors, and reward actions which remove them. This means that during

training, the agent has implicit access to the underlying true error configuration through the reward function, so it is somewhat able to consider the physical errors rather than the syndromes only. Contrastingly, during evaluation, the agent no longer has access to the hidden true state, and thereby the shaping reward. Rather, we use these evaluation episodes to test how long the agent is able to survive without a logical error occurring.

We define the shaped training reward as follows:

Let Δe denote the net change in physical bit-flip errors on the physical qubits resulting from the agent's correction action. The reward at each time-step is defined as:

$$r_t = \begin{cases} 1 - 0.4\,\Delta e & \text{if } \Delta e > 0 \quad \text{(error introduced)} \\ 1 - 0.2\,\Delta e & \text{if } \Delta e \leq 0 \quad \text{(error corrected)} \end{cases} \tag{3}$$

where $\Delta e = \sum_i \left(e_i^{(t)} - e_i^{(t-1)} \right)$ is computed over all physical qubits, with $e_i^{(t)} \in \{0, 1\}$ indicating the presence of a bit-flip error on qubit i after the agent's action but before environment noise is applied. The weights are asymmetric, where introducing an error is penalised twice as heavily as correcting one is rewarded. This prevents the agent from exploiting a loop in which it deliberately introduces errors only to correct them later for reward. The constant term of 1 rewards survival at each step, independent of the change in errors.

As previously mentioned, we must also consider the partially observable nature of QEC. The observation space consists of the stabilizers placed between the qubits. The observation is derived from the Z-type ancilla measurements (the bit-flip syndrome), since only X-errors are considered in this paper. The agent then observes the list of stabilizers in each round, with 0 meaning either no physical errors detected, or an even number of physical errors detected (maximum 4 in the rotated surface QEC environment), and 1 meaning either a single physical error or three errors detected, which we refer to as the syndromes. For a distance d surface code, there are $\frac{d^2-1}{2}$ Z-stabilizers. The observation space can therefore be formulated $\mathcal{O} : \{0, 1\}^{\frac{d^2-1}{2}}$, where 0 indicates an even parity of physical errors, and 1 indicates an odd parity of physical errors. When setting $d = 3$ this gives 4 Z-ancillas, resulting in $2^4 = 16$ possible syndrome patterns. When including history, we concatenate the current observation to the previous observation(s), so that the state space grows

exponentially (i.e. 256 possible syndrome patterns for $d = 3$). For that reason, we consider the smallest possible lattice, with $d = 3$.

The observation function $Z : \mathcal{S} \to \mathcal{O}$ maps the underlying physical qubit error configuration to the syndrome vector observed by the agent. At each time-step t, the Z-type ancilla qubits are measured, giving a raw measurement vector $m_t \in \{0, 1\}^{(d^2-1)/2}$. The agent observes the raw syndrome relative to the error-free reference measurement m_0,

$$o_t = Z(s_t) = m_t \oplus m_0, \tag{4}$$

where $\oplus$ denotes the bitwise XOR operation. An entry $o_t[i] = 1$ therefore indicates that stabilizer i is currently triggered, signalling a detected defect in its neighbourhood, while $o_t[i] = 0$ indicates that it is not.

The goal is to learn an optimal policy (π^*). A policy is a mapping from observations to actions, in terms of probability [CM23]. That is, given the current state, how likely is the agent to select action a. An optimal policy is one that maximizes the total long-term reward given the MDP formulation. In our case, the goal is to learn the policy which allows the agent to survive as long as possible in the QEC environment without incurring a logical error. A logical error occurs when errors on the physical qubits corrupt the encoded (logical) qubit in a way the reference MWPM decoder (in our environment) cannot undo. Different error patterns create the same syndrome, meaning that the decoder can infer the wrong correction. When its correction does not cancel the true errors but instead combines with them to flip the logical qubit, the stored information is lost, which is checked for every five rounds (see Section 5). Therefore, we measure the performance given the total time-steps in the environment before such an error occurring.

2.3 Reinforcement Learning

In this work, we consider the setting of the QEC physical qubit lattice as a game environment for a reinforcement learning (RL) agent. We must therefore first understand what is meant by RL. RL is one of the three types of machine learning (ML)[ME22], a class of algorithms that enable a computational system to learn patterns from data and, without explicit instructions (though different from unsupervised learning, which will not be discussed in detail in this paper), improve its performance on a given task[AFL21].

In RL, an agent discovers its environment through interaction [SB18]. This learning process is sequential, where at each time-step, the agent considers the current state and selects an action. More specifically, the action is chosen according to a policy, which is the mapping from states to actions. The action results in feedback in the form of a positive or negative reward. To evaluate the success of an action sequence and change its policy accordingly, the agent calculates a cumulative reward, which is influenced by the immediate feedback of the action, and prior knowledge. The RL cycle ends with the agent adjusting the policy to maximize long-term expected returns.

We must also consider that rewards which are received far into the future are often of less value to the agent than immediate ones. Therefore, we introduce the discount factor $\gamma \in [0, 1]$, which weights each reward according to how far into the future it is received. The sum of rewards received from time-step t onward is known as the return G_t, as in Equation (5).

$$\begin{aligned} G_t &= \gamma^0 r_{t+1} + \gamma^1 r_{t+2} + \gamma^2 r_{t+3} + \dots \\ &= \sum_{k=0}^{T} \gamma^k r_{t+k+1}. \end{aligned} \tag{5}$$

Where T is the maximum number of time-steps in the episode. Besides weighing the relative importance of future rewards, the discount factor also ensures the mathematical convergence of the return, since for $\gamma < 1$ the very distant rewards converge to zero.

Building on the return, we define the action-value function, or Q-function. For a policy π, the value $Q^\pi(s, a)$ is the expected return obtained by taking action a in state s and following π [AJLG19], as in Equation (6).

$$Q^\pi(s, a) = \mathbb{E}_\pi[\, G_t \mid s_t = s,\ a_t = a \,]. \tag{6}$$

An optimal policy π^* is one that maximises this value in every state; its action-value function is $Q^*(s, a) = \max_\pi Q^\pi(s, a)$, and acting greedily with respect to it, $a = \arg\max_a Q^*(s, a)$, yields the highest expected return. The tabular agent in Section 4.1 learns an estimate of this Q-function via the update in Equation (9) and selects actions according to Equation (10).

Due to the dependency on hand-crafted features in earlier ML [FLHI+18], and tabular representation (Q-table) in earlier RL, the two were unable to scale to raw, high-dimensional

observations such as images, or in our case, the QEC environment when considering previous observations. This issue can be addressed by deep learning (DL). DL models are built from many layers of interconnected neurons, and training adjusts their weights and biases to yield more accurate outputs [Ber].

Then, in order to continue the use of RL while improving sequential decision making in problems with high dimensional state-space, we can combine the two methods, also known as deep reinforcement learning (DRL) [SB18]. This can be done by using the previously described deep neural networks as function approximators for the value function, or policy [FLHI+18]. The reinforcement learning framework is still kept by updating the neural network weights based on rewards that are received from interactions with the environment. DL is the function approximator, and RL provides the learning objective. Together, they allow policies and value functions to be learned from high-dimensional inputs.

3 Related Work

Before learning-based approaches, the decoding of surface and toric codes was, and to a large extent still is, performed by classical algorithms. Given a syndrome, such a decoder infers the most likely underlying physical errors and the correction that removes them.

The most widely used is minimum-weight perfect matching (MWPM), which interprets the syndrome as a graph-matching problem [PZN26]. This method, however, is only suitable when applied to fixed, known noise models and is not naturally adaptive. Therefore, we are more interested in a method which is able to adapt to the change in noise, as in Section 4. This is the main limitation that motivates learning-based decoders.

A first response to this limitation has been to learn the decoder from data. Supervised neural-network decoders train a network to map syndromes directly to corrections, such as Google's AlphaQubit which used a recurrent transformer-decoder to outperform matching-based decoders on real hardware [BSH+24]. These methods, however, require large labelled datasets of syndrome-error pairs. We therefore consider the alternative of RL, where the agent learns a correction policy directly from interaction, and frames decoding as a sequential decision problem (see Section 2.3). This step-by-step operation is also an advantage over MWPM, since MWPM computes a single global matching of the defects for

a given syndrome, and the RL agent corrects incrementally and re-observes the syndrome after each action, making it more adaptable as additional errors are introduced over time [AJLG19]. This framing makes RL well-suited to perform QEC and should allow the agent to adapt to noise changes.

The idea for using RL as a decoder for Quantum Computing has already been researched using various different setups. For example, we consider a paper, which too, evaluates RL for surface code [SKvNE20], and found that with $d = 5$, with bit-flip noise $p < 1.3 \cdot 10^{-2}$, the DeepQ RL decoding agent is able to prolong the average lifetime of a logical qubit to be longer than that of a single faulty qubit. The idea for this paper was strongly inspired by the paper of Lindsay Spoor [Spo24], which considers toric code, unlike the (rotated) surface code used in this paper.

In this paper, the physical error rate is not held fixed but drifts over time (Section 4), so the agent faces a non-stationary noise model rather than the fixed one assumed by classical decoders. A further distinguishing aspect of this work is the role of observation history. While the most commonly studied RL decoders act on a single syndrome observation, we provide the agent with a history of past observations and study how this affects its survival time before a logical error. We contrast two classes of methods on this question, a tabular agent (Q-learning) and a function-approximation agent (PPO). Our evaluation also differs methodologically from prior work such as Sweke et al., where rather than comparing the logical qubit's lifetime to that of a physical qubit, we measure the agent's survival length, which is the number of time-steps before a logical error occurs, as a function of training. We do so at a fixed code distance of $d = 3$, chosen because the history-including state space grows exponentially with distance, and at an initial physical error rate of 0.01, roughly that expected of current physical qubits [Mic24]. All experiments use the QEC environment provided by Jan Krzywda.[1]

4 Methods

This section describes the methods used to train and evaluate decoding agents in the QEC environment introduced in Section 2. We begin with the noise model, since it defines the conditions the agent must learn to handle, and in particular the error drift that makes the

[1] https://github.com/jaq-lab/qec-control-gym.git

environment non-stationary. We then present the two reinforcement learning algorithms used to train the decoder, tabular Q-learning and Proximal Policy Optimization (PPO), along with the different ways each algorithm represents the observation.

Error Drift

The per-qubit physical error rate is not held fixed but evolves as a simplified Ornstein-Uhlenbeck process [CMR22], described by Equation (7).

$$dX_t = -\theta\, X_t\, dt + \sigma\, dW_t, \tag{7}$$

where X_t is the value of the process at time t (in our case the probability of a bit-flip occurring on a given physical qubit), θ is the mean-reversion rate, σ the volatility, and W_t a Wiener process. The drift term $-\theta X_t$ pulls the rate back toward zero at a speed set by θ, while $\sigma\, dW_t$ inputs Gaussian fluctuations [CMR22].

In the implementation we use (see GitHub), we take, for the bit-flip error rate $p_x^{(t)}$, a unit step ($\Delta t = 1$), a mean-reversion rate $\theta = 1/\tau_c$ with $\tau_c = 50$, and volatility $\sigma = \sigma_0\sqrt{2\theta}$. Lastly, the rate is clipped to $[10^{-9}, 0.4]$ to remain a valid probability, resulting in Equation (8)

$$p_x^{(t+1)} = \mathrm{clip}\big(p_x^{(t)} - \theta\, p_x^{(t)} + \sigma\, \xi_t,\ 10^{-9},\ 0.4\big)\,, \quad \xi_t \sim \mathcal{N}(0,1). \tag{8}$$

4.1 Q-learning

Now that the noise model is known, we consider the learning algorithms used to train the decoding agent. We start with tabular Q-learning, a model-free baseline that lets us characterize the problem in its simplest form for the preliminary experimentation and research.

In tabular RL, the agent maintains a table $Q(s, a)$ estimating the expected cumulative reward of taking action a in state s, updated at each time-step t by the Bellman equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\big], \tag{9}$$

where $\alpha \in (0, 1]$ is the learning rate.

There are three main classes of tabular RL methods, namely, dynamic programming (DP), Monte Carlo methods (MC), and temporal-difference (TD) learning [CM23].

DP relies on a complete and accurate model of the environment, which is not available. MC methods, while they do not require such models, estimate values from full episodic returns, meaning that learning updates can only occur after an episode has completed. In our QEC environment, episodes may last up to 100 time-steps, which means an MC agent would not be able to learn from intermediate experience within an episode, making learning less efficient.

TD methods require no model and are stepwise incremental [SB18]. The Q-learning algorithm, a TD-based model, is off-policy. Off-policy means that the agent splits the policy into two separate ones, the behaviour policy and the target policy. We chose this implementation, as this allows the Q-learning agent to learn the optimal policy independently of the induced exploration noise. The behavior policy dictates how the agent selects actions and navigates the environment, whereas the target policy represents the optimal policy the agent aims to learn and maximize. This allows the agent to explore in the behaviour policy, such that it gains more accurate knowledge of the environment, that it can then exploit in the target policy.

Unknown action values are the main issue for action-selection. Consequently, the agent must use the $Q(s, a)$ estimates. To adjust the estimates such that they are as close to the real values as possible, the agent needs to explore. That is, selecting actions which currently seem un-optimal. Of course, in order to maximize the reward, the agent must indeed select the optimal actions at all times. Here, greedy action selection should result in optimal long-term reward due to the predictive nature of the reward estimates.

This results in the exploration-exploitation problem, of balancing improvement of the agent's knowledge, and making use of its current knowledge. Without sufficient exploration, the agent may fail to discover the correct optimal action for a given state, leading to lower long-term reward. However, without sufficient exploitation, the agent may spend too many time-steps selecting suboptimal actions rather than acting on its current knowledge.

Therefore, for the behaviour policy action selection, we use a standard exploration-exploitation method, ε-greedy, as defined in Equation (10) [SB18]. ε-greedy action selection allows the agent to select the action it currently believes to lead to the highest reward, also known as exploitation, for the majority of the time $(1 - \varepsilon)$. However, with probability ε, the agent must select a random action, which in our specific implementation, allows also

for the best (estimated) action to be drawn from the random selection.

This policy can be formalized in Equation (10):

$$\pi_{\varepsilon\text{-greedy}}(a) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}|} & \text{if } a = \text{argmax}_{a' \in \mathcal{A}} Q(a') \\ \frac{\varepsilon}{|\mathcal{A}|} & \text{if } a \neq \text{argmax}_{a' \in \mathcal{A}} Q(a') \end{cases} \tag{10}$$

where $\pi(a)$ is the probability of selecting action a in state s, $\mathcal{A}$ is the set of all possible actions, such that $|\mathcal{A}|$ is the size of the action space. $Q(a)$ is the estimated value of selecting action a from the current state, and $\varepsilon \in [0, 1]$ is the exploration parameter. The higher we set ε, the more exploration. In our specific implementation, we introduce a decaying epsilon such that we can ensure that the agent begins with higher exploration, but begins to exploit more with time, as its knowledge of the environment becomes more accurate.

The first case of the equation describes the probability of selecting the greedy action, $a = \text{argmax}_{a \in \mathcal{A}} Q(a)$. We refer to this as exploitation, since it takes advantage of the current knowledge of the environment. In the evaluation episodes, which use the target policy, exploitation is the only approach the agent considers. This is done in order to assess the performance that the agent's current knowledge can lead to. The second case of the equation is the random selection, where we give equal probability to the $|\mathcal{A}|$ actions. This is exploration, since the current knowledge is not being used, and instead, the agent possibly chooses actions which it deems as non-optimal. This is important to correct for the fact that, especially in the beginning, the knowledge that the agent has is not fully accurate. This means that actions which seem to have the highest return are not the best actions in the long run.

A syndrome gives only a partial observation of the underlying error state, since the same syndrome pattern can result from multiple different underlying physical error configurations. To study the effect of memory on the performance of the agent, aiming to counter-act this partial observability, we extend the state to include the k most recent previous syndromes, for $k \in \{1, 2, 3\}$.

In our implementation, we refer to each state as an index. To do this, we simply convert the binary syndrome observation to the decimal number system.

Given $N = 2^4 = 16$ possible single-step states, the combined state index for $k+1$ consecutive observations results in Q-table sizes of $16^{k+1} \times 11$:

History	States	Q-table entries
$k = 0$ (no history)	16	176
$k = 1$ (1-step)	256	2,816
$k = 2$ (2-step)	4,096	45,056
$k = 3$ (3-step)	65,536	720,896

Table 1: Q-learning state-space growth with inclusion of k previous observations

As the Q-table grows exponentially with history depth, we have a problem that with a fixed episode budget, the fraction of states that receive enough visits to reach reliable Q-value estimates decreases. For the tabular agent, we restrict the history to $k \leq 1$. Instead, the deeper history depths are studied only with PPO.

4.2 PPO

Moving beyond tabular methods, we now consider Proximal Policy Optimization (PPO) [SWD+17], a family of policy gradient algorithms introduced by OpenAI. These are designed to overcome the limitations of standard policy gradient methods, which are often sensitive to step size choices and need many interactions with the environment to reach a certain performance level. PPO improves on these issues by using a clipped surrogate objective such that policy updates do not deviate too far from the previous policy.

At each update, PPO considers the probability ratio, r, between the new policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$:

$$ratio_t(\theta) = \frac{\pi_\theta(a_t \mid o_t)}{\pi_{\theta_{\text{old}}}(a_t \mid o_t)}, \tag{11}$$

where θ denotes the policy parameters, a_t is the action taken at time-step t, o_t is the observation at time-step t, and $\pi_\theta(a_t \mid o_t)$ denotes the probability of selecting action a_t given observation o_t under the policy parameterised by θ.

While standard policy gradient methods use an objective that can lead to unstable, large updates, PPO defines a clipped surrogate objective, L^{CLIP}, that penalizes changes which

move $ratio_t(\theta)$ too far away from 1:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min\big(ratio_t(\theta)\, \hat{A}_t,\ \text{clip}(ratio_t(\theta), 1-\varepsilon, 1+\varepsilon)\, \hat{A}_t \big) \right], \tag{12}$$

where $\hat{\mathbb{E}}_t$ is the sample average over time-steps, $\hat{A}_t$ is the estimated advantage at time t, and $\varepsilon \in (0, 1)$ is a hyperparameter that determines the clipping range. By taking the minimum between the clipped and unclipped versions, this objective forms a pessimistic lower bound on the policy's performance. That is, changes in the probability ratio are ignored when they would improve the objective beyond the clip threshold, but included when they make the objective worse. In this way, PPO encourages stable updates.

Given that in PPO, the policy and value function share parameters, we combine the clipped surrogate loss with a value function error term and an entropy bonus encouraging exploration [MBM+16]. The total objective becomes:

$$L_t^{\text{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{\text{CLIP}}(\theta) \ - \ c_1\, L_t^{\text{VF}}(\theta) \ + \ c_2\, S[\pi_\theta](s_t) \right], \tag{13}$$

where $L_t^{\text{VF}}(\theta) = \big(V_\theta(s_t) - V_t^{\text{targ}} \big)^2$ is the squared-error value loss, $S[\pi_\theta](s_t)$ is the entropy of the policy at state s_t, and c_1, c_2 are coefficients weighting each term (in our case, 0.5 and 0.005 respectively). The value loss trains the critic to predict expected returns, while the entropy bonus prevents the policy from becoming deterministic too quickly, similar to what ε does in Equation (10). Furthermore, in our code implementation, the signs were flipped in order to minimize the loss rather than maximize it.

To reduce variance in the advantage estimates, PPO also uses a cut-off version of Generalized Advantage Estimation (GAE) [SML+15]:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t-1}\delta_{T-1}, \tag{14}$$

where at time t, the temporal difference (TD) error δ_t is:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t), \tag{15}$$

where V is the value estimate produced by the critic, $\gamma \in [0, 1]$ is the discount factor, T is the total number of time-steps left in the episode, and $\lambda \in [0, 1]$ is the GAE smoothing parameter that trades off bias and variance in the advantage estimate, and the advantage

estimate is then the exponentially-weighted sum of these residuals. Then, $\lambda = 0$ is the one-step TD estimator (low variance, and high bias), while $\lambda = 1$ is the Monte Carlo return (high variance, low bias).

As previously stated, PPO stands out in that it requires much fewer interactions with the environment in order to reach a high performance, as it is able to generalize.

Observation representation in PPO versus Q-learning

An important difference between the tabular Q-learning agent and PPO, is the manner in which the observation is represented as input to the algorithm. In the Q-learning setting, we collapse the syndrome observation into a single discrete state index, converting raw binary observations to the decimal number system. We are able to do this because Q-learning does not generalize. That is, the agent learns a specific action value for every individual state-action pair separately, and does not make use of information from other states when estimating the value of the current state. Therefore, the exact structure of the observation does not matter to the agent, but rather only its identity as an index.

In the case of PPO, however, this would be harmful to the learning of the agent, and taking away the main advantage of the algorithm in contrast to the tabular setting. PPO learns a policy and value function which generalize across similar states, meaning that the agent does not learn a separate action for each state, but rather learns shared weights which apply across the entire input space. Naturally, if we were to collapse the observation into a single index, we would lose the structure that the network relies on in order to generalize. Therefore, in PPO we provide the raw syndrome vector to the network directly, either as a single observation or as a concatenation of the most recent observations when history is included. The agent is then able to exploit the similarities between observations. So, when two syndrome patterns are similar, the learned features of the network respond to them similarly, meaning that the agent is able to generalize to states which it has rarely, or even never, visited.

5 Experimental Setup

In this section we describe the configuration of the QEC environment, the two learning algorithms, and the method with which we evaluate them. All experiments are run on a

distance $d = 3$ rotated surface code.

Environment configuration

The QEC environment is fully specified by the parameters in Table 2. We study only bit-flip (X) errors, such that the initial phase-flip rate $p_z^{(0)}$ is set to zero, and the agent is given a single correcting action per data qubit accordingly. As described in the Errors Drift section, the bit-flip rate is not held fixed but evolves according to Equation (8), with standard deviation σ_0 and mean-reversion timescale τ_c. The environment retains a single syndrome round in each observation. This observation is the syndrome pattern, i.e. the set of stabilizers currently triggered relative to the error-free starting point, and the history depth k studied in this work is applied on top of this single-round observation at the level of the agent, rather than within the environment itself.

Parameter	**Value**	**Description**
d	3	Rotated surface code distance
$p_x^{(0)}$	0.01	Initial bit-flip (X) error rate
$p_z^{(0)}$	0.00	Initial phase-flip (Z) error rate
σ_0	0.01	Standard deviation of the error-rate drift
τ_c	50	Mean-reversion timescale of the drift (time-steps)
readout error	0.00	Ancilla measurement error rate
rounds_history	1	Syndrome rounds retained in the environment observation
check_period	5	Time-steps between logical-error checks
max_steps	100	Maximum time-steps per episode
use_hook_errors	false	Hook errors disabled

Table 2: Configuration of the QEC environment used in all experiments.

We further implement a maximum of 100 time-steps per episode, as listed in Table 2. This provides sufficient steps per episode to evaluate a clear learning curve, yet maintains a shorter run-time for the experiments.

Q-learning configuration

For the tabular Q-learning agent, we use the hyperparameters listed in Table 3. It is often the case with regular ε-greedy agents, that sufficient knowledge of the environment is gained after some time, yet the agent is still forced to take non-optimal actions (with a probability of ε). For this reason, we implement ε decay. That is, the exploration rate begins at $\varepsilon = 0.5$ and is multiplied by a decay rate of 0.99997 after each time-step, until it reaches a minimal value of 0.05, such that the agent explores broadly at the start of training and increasingly exploits its learned values as training progresses. The final hyperparameter values for the Q-learning agent are as in Table 3.

Hyperparameter	**Value**
Learning rate α	0.1
Discount factor γ	0.99
Initial exploration ε	0.5
Exploration decay rate	0.99997
Minimum exploration ε	0.05
Training time-steps	1,000,001
History depth k	$\{1, 2\}$

Table 3: Hyperparameters of the tabular Q-learning agent.

PPO configuration

For the PPO agent we refer to the CleanRL implementation, with modifications to the network architecture to suit the comparatively small state space of our problem. The policy and value function are parametrized by a shared multi-layer perceptron with two hidden layers of 512 units each and ReLU activations, followed by two separate output heads, as in Equation (16), known as the actor-critic setup. That is, the actor head outputs a probability distribution over the $|\mathcal{A}|$ actions, which is the policy itself, and the critic head outputs a single scalar estimate of the value of the current state, which is used to compute

the advantage of each action.

$$\text{Input} \;\to\; \mathrm{L}(d_{\text{obs}}, 512) \;\to\; \text{ReLU} \;\to\; \mathrm{L}(512, 512) \;\to\; \text{ReLU} \;\to\; \begin{cases} \text{Actor: } \mathrm{L}(512, |\mathcal{A}|) \\ \text{Critic: } \mathrm{L}(512, 1) \end{cases} \tag{16}$$

where L is a linear layer. The full set of PPO hyperparameters is given in Table 4.

Hyperparameter	**Value**
Training time-steps	1,000,001
Rollout size	2,048
Minibatch size	512
Update epochs	10
Learning rate	2.5×10^{-4}
Discount factor γ	0.99
GAE parameter λ	0.95
Clip coefficient ε	0.1
Entropy coefficient	0.005
Value loss coefficient	0.5
Maximum gradient norm	0.5
Hidden layer size	512

Table 4: Hyperparameters of the PPO agent.

Evaluation method

Throughout training, we periodically evaluate the agent in order to measure its progress. During an evaluation episode, the agent no longer learns and no longer explores, but rather selects actions greedily, so that it always chooses the action which it currently believes to be best. The performance measure is the survival length, the number of time-steps the agent survives in the environment before incurring a logical error, capped at the maximum of 100 time-steps per episode. For both the Q-learning agent, and PPO agent, we evaluate every 5000 time-steps, and in both cases we average the survival length over 30 evaluation episodes. To account for the variance between training runs, every experiment is repeated over 15 random seeds, and the resulting learning curves are averaged across these seeds. For

readability of the presentation, all curves are smoothed using a Savitzky-Golay filter with smoothing window 35 and polyorder 1. Lastly, all learning curves are run over 1,000,001 training time-steps.

6 Results

All results can be re-created following the code posted on GitHub: https://github.com/alma-rosenmann/qec-control-gym.git. Instructions for running the RL agents are under README_RL.md.

We began with our preliminary experiments in the tabular Q-learning.

6.1 Q-learning

As previously stated, the main question that this research paper would like to address is the performance difference in an RL agent performing QEC when provided only the current time-step's observation, as opposed to an agent which has not only information about the current time-step, but also the previous one. In all of our experiments, we will refer to these two agents as "no history agent", and "history agent", respectively.

Now, let us consider the overall comparison of the history versus no history agent, as seen in Figure 2. We believe that the history agent will outperform the no history agent. Although the syndrome observation already reports which of the stabilizers are currently triggered, it remains only a partial observation of the underlying physical error configuration, since many distinct configurations produce the same syndrome (or lack thereof). A history of recent syndromes provides the sequence of syndromes over consecutive rounds, which we expect to help the agent disambiguate the configuration that a single syndrome leaves ambiguous.

Figure 2: Q-learning algorithm performance on a 3x3 lattice with a bit-flip error rate of 0.01 with 0.01 drift, comparing a Q-learning agent that uses only the current observation with a Q-learning agent that uses both the current and previous observation as a state representation. We see that the performance of both agents is very similar towards the final evaluations, but the history agent is slightly outperformed.

In contrast to our original hypothesis that the history agent will outperform the no history agent, we see in Figure 2 that the performance of the two agents is relatively equal. Indeed, after a million training time-steps, the Q-learning agent with no history reaches a smoothed, survival length of 66.60, while the Q-learning agent with a time-step of history reaches one of 64.01, such that the no history agent marginally outperforms the other agent. However, this is a rather marginal difference which may also be amplified by the specific seed selection, and smoothing. Regardless, we can say with certainty that our original hypothesis, that the history agent will outperform the no history agent, does not hold for the tabular RL agent.

Taking a further look at the visitation tables of these two agents, as in Figure 3 and considering Table 1, we may hypothesize that the history agent is unable to outperform the no history agent due to exponential growth of the state space that comes with the inclusion of the previous observation into consideration. That is, in the same number of

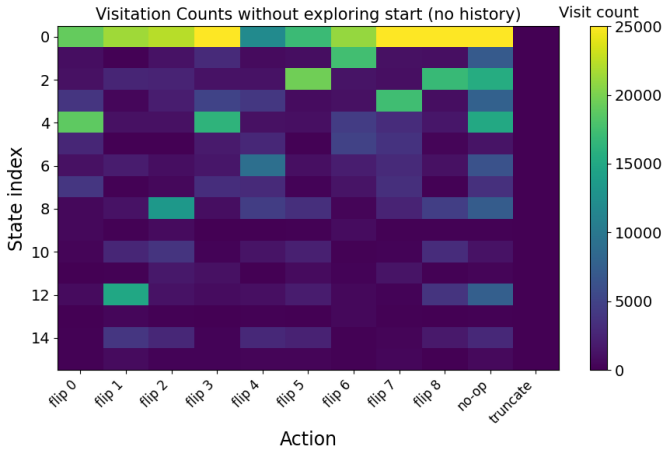
time-steps as the no history agent, the history agent simply is unable to visit all of the relevant/possible states enough times to have sufficient knowledge of the optimal action to take in that state. We know that the more frequently visited state-action pairs will have more accurate values, and therefore result in an agent that is able to survive for longer.

In order to correctly analyze these visitation tables, let us first re-consider Figure 1. As previously stated, we choose to collapse the state-space to indices. For each state index, we can re-convert it to the length-4 binary representation. For example, state 0 can be re-written 0000, state 1 as 0001, and so on. In our environment, Z-ancilla 9 is the first (left-most) binary, measuring physical qubits 1 and 2, followed by Z-ancilla 11, with the full state description in Table 5.

Syndrome position	**Z-ancilla index**	**Physical qubits checked**
0	9	$\{1, 2\}$
1	11	$\{0, 1, 3, 4\}$
2	14	$\{4, 5, 7, 8\}$
3	16	$\{6, 7\}$

Table 5: Syndrome positions and corresponding Z-ancilla index and checked physical qubits.

Considering this information, we can now see which state-action pairs are more or less frequently visited.



(a) No history agent

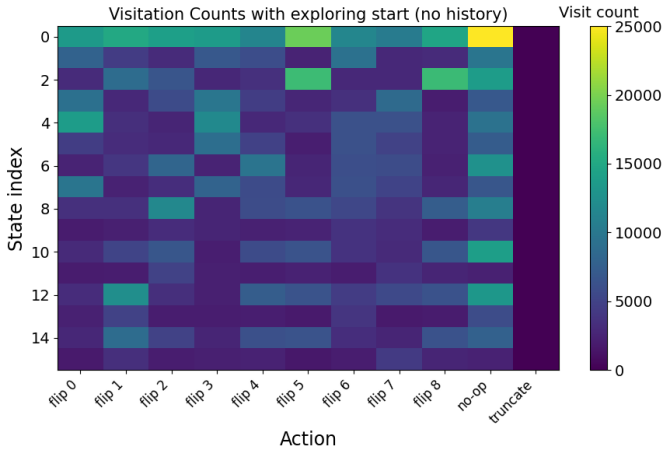
(b) History agent

Figure 3: Q-table visitation of the Q-learning agent's training on a 3x3 lattice with a bit-flip error rate of 0.01 with 0.01 drift. In both figures, yellow indicates a highly visited state, blue a semi-highly visited state, and purple a rarely-visited state. In (a), high visitation means around 25,000 instances, semi-highly around 15,500, and rarely-visited is under 1,000. In (b), high visitation means around 5,000 instances, semi-highly visited around 2,500, and rarely-visited is around 0. We see in (b) that much of the state-action space is rarely-visited.

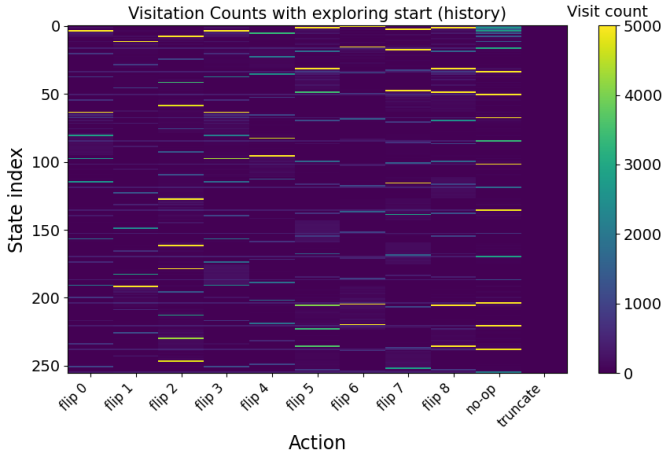
We see, indeed, that in Figure 3a, most of the table has been visited frequently enough that the agent's estimate of the optimal action in each state can be assumed reasonably accurate. However, Figure 3b shows that much of the table is visited only rarely, so the

agent's value estimates there are less reliable, leading to poor action selection in those states. Of course, given the large state-action space, we rescale the legend for the history agent. Still, even with this change, much of the state-action space is purple, indicating that the knowledge provided by the additional observation input is not able to overcome the limitation in frequency of visitation per state-action. While any additional knowledge, or input, cannot actually harm the performance of the agent, it does slow it significantly, such that the expected cumulative reward even after 1 million time-steps is slightly lower than without it. The need to cover more of the possible state-action pairs in order to reach better knowledge of the environment motivates the implementation of exploring starts, a method for enforcing exploration in which each episode begins with a randomly chosen state and action, and only afterwards considers the behaviour policy [WYSR22].

Implementing this change in the Q-learning algorithm results in the following visitation tables Figure 4. Indeed, we see in Figure 4 that with the exploring starts, many more state-action pairs have high visitation counts. We may therefore assume that the exploring-starts implementation will improve the survival length of the Q-learning agent, given that we assume that a more uniform distribution of visitations across the state-action space indicates better knowledge of the environment. We note that the truncate action is recorded as unvisited because the visitation counter logs an action only when the episode continues to a further step, which is naturally not the case with the truncate action. Using this method, we compare the survival of the history and no history agent with and without these exploring starts. In our implementation, exploring starts meant that each physical qubit had equal probability of having a bit-flip or not for the start state, followed by equal probability for all actions in the first action selection.

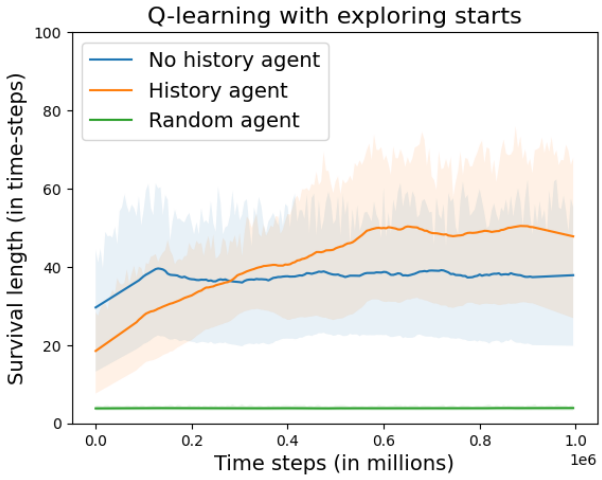


(a) No history agent

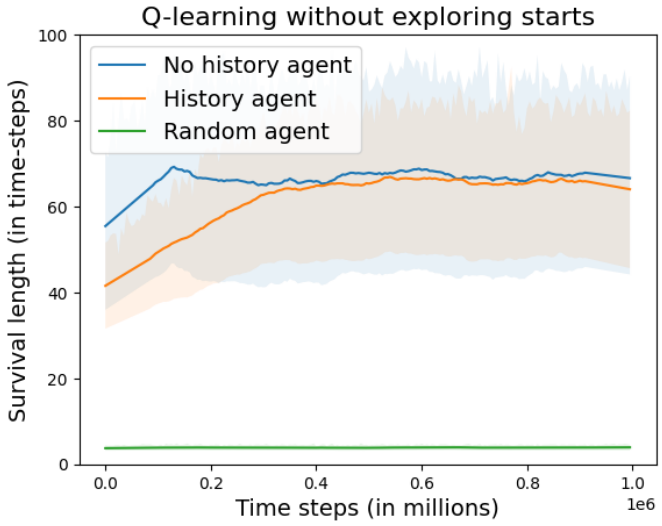


(b) History agent

Figure 4: Q-table visitation of the exploring start Q-learning agent's training on a 3x3 lattice with a bit-flip error rate of 0.01 with 0.01 drift. In both figures, yellow indicates a highly visited state, blue a semi-highly visited state, and purple a rarely-visited state. In Figure 4a, high visitation means around 25,000 instances, semi-highly around 15,500, and rarely-visited is under 1,000. In Figure 4b, high visitation means around 5,000 instances, semi-highly visited around 2,500, and rarely-visited is around 0. We see that much more of the state-action space has now been at least semi-highly visited.



(a) Q-learning agent performance with exploring starts implemented



(b) Q-learning agent performance without exploring starts

Figure 5: Q-learning algorithm performance with (a) and without (b) exploring starts on a 3x3 lattice with a bit-flip error rate of 0.01 with 0.01 drift, both with standard deviation plotted. As can be seen, the exploring starts lowers the survival length of both agents significantly in the given time-steps.

As can be seen in Figure 5, the Q-learning agent that begins with exploring starts, both in the case of the history agent, and the no history agent, performs worse than the original

Q-learning agent. In fact, the averaged and smoothed survival length of the two agents at the end of their training is 37.86 for the no history Q-learning agent, and 47.79 for the Q-learning agent with a history of one time-step. While the history and no history agent clearly outperform the random agent in both Figure 5a and Figure 5b, the exploring starts still seem to disadvantage the agent. As this goes against the initial intuition of introducing the exploring starts, we can inspect why this is the case by referring back to the visitation tables in Figure 4.

We hypothesize that this is caused by a mismatch between the states visited during training with exploring starts and the states encountered during evaluation. At an error rate of 0.01, the agent naturally spends almost all of its time in low-error states. Exploring starts causes each episode to begin from a random state-action pair, but, especially given the extremely low error rate, this includes states that are very rarely reached when following the policy at the true error rate. For example, as seen in Figure 4, state 15 of the no history agent already corresponds to all stabilizers being triggered, so the agent spends a portion of its updates learning about states it will almost never encounter during evaluation. We also see that for the no history agent without exploring starts, nearly all actions in state 0 have been visited frequently. Contrastingly, the exploring-start no-history agent, and the upper right corner of the history agent shows the same phenomenon. It is important to note that in the tabular setting, each $Q(s, a)$ is stored and updated independently, with no generalization between states. Exploring starts therefore cannot cause the agent to learn a worse policy on the states it actually reaches, but only changes how the fixed budget of value updates is distributed across the state-action space. What the agent learns about a high-error state such as state 15 is less impactful for the very low-error states. This means that the worsened performance caused by exploring starts is due to the redistribution of updates rather than from interference between states. Although both agents train for the same number of time-steps, the distribution of the updates across the state space differs significantly. With exploring starts, episodes that begin in high-error states reach a logical error quickly and terminate, and the agent then resets to another random, often high-error, state. As a result, over the course of 1,000,001 time-steps, a large fraction of the Q-value updates are spent on states in the high-error region that the agent will never naturally reach. Without exploring starts, every episode begins from a clean lattice and the agent remains in the low-error states for longer before a logical error occurs, so that nearly all updates are concentrated on the more reachable states.

This effect is clearer when considering the specific state-actions visitation when compared side-by-side. Without exploring starts, the low-index states of the no history agent still receive sufficient coverage, whereas with exploring starts (Figure 4), the visits are distributed more uniformly but the per-state counts on these low-index states are lower, in particular the actions one would logically try given the state. Consider, for example, state 1 (representing syndrome measurement 0001). The agent without exploring starts most frequently visited flip 6. Looking at Figure 1, we see indeed that considering the fourth Z-ancilla measurement checks physical qubits 6 and 7, and that if 7 were the flipped qubit, the observation would result in 0011 (due to Z-ancilla 14 measuring qubit 7), the agent checks the (most likely) correct physical qubit. However, the exploring starts agent visits this action much less frequently for state 1. Instead, it is nearly equally likely to flip qubit 0, 3, or choose to perform no operation.

That is to say, in this setting, the decisive factor for the agent's performance is not the breadth of coverage across the full state space, but the depth of coverage on the reachable states. This also means that if we were to run the exploring starts agents for longer, they would eventually reach, and possibly surpass the performance of the no exploring starts agents. We also see that with the exploring start, the history Q-learning agent is able to quite significantly outperform the no history agent. While this was not further experimented with in this paper due to the overall lower performance, it would make for an interesting next step to consider whether this would have stronger implications when used for the PPO agent, or for a larger lattice.

Future research may consider adding an agent that never corrects, as a reference point. The bit-flip rate of 0.01 per data qubit per round (with drift) causes errors to accumulate uncorrected and should cause a logical error to occur earlier. The learned agents' survival should be read against this baseline to measure how much the agent's interventions prolong the survival.

As established in this section, the tabular Q-learning agent is limited by the exponential growth of the state space when history is included. Even with exploring starts, the history agent is unable to sufficiently cover its Q-table to reach a very high performance. This motivates the use of Proximal Policy Optimization (PPO), which parametrizes the policy with a neural network and can therefore generalize across similar syndrome patterns without requiring explicit visits to every state-action pair [SWD+17].

We now investigate whether this ability to generalize allows the history agent to outperform the no history agent. If the history agent's poor performance in the tabular setting was indeed caused by insufficient coverage of the state space rather than the history observation being uninformative, then PPO should be able to overcome this limitation.

History and generalization in PPO versus Q-learning

Although both the Q-learning and PPO agents are able to make use of history, the benefit which they gain from it is of a different nature. In the case of the Q-learning agent, the inclusion of most history increases the state space from 256 to 4096 for $k = 2$. As discussed, the agent must visit each state-action pair a sufficient number of times in order to learn a reliable value estimate, such that this requirement becomes unfeasibly expensive as the state space grows. As we indeed saw in Figure 2, the history agent takes around 400 thousand time-steps in order to match the performance of the no history agent in the tabular setting, and is entirely unable to outperform the no history agent within the given 1 million time-step budget.

In the case of PPO, however, history allows for a more powerful form of learning. Namely, the agent is able to generalize across syndrome patterns regardless of their position in the history window. Because the network learns shared weights rather than memorizing each state individually, it can recognize a syndrome configuration, or a short sub-sequence of syndromes, wherever it occurs within the window. So, if we consider a history length of $k = 3$, such that the agent observes four consecutive syndromes, the agent is able to learn that a pattern appearing in the earlier part of the window carries a similar implication when it reappears later in the window. The network learns this implicitly through its weights, and does not require an explicit visit to every possible sequence of observations. PPO's ability to learn patterns is a significant advantage of the function approximation over the tabular setting. Then, PPO is able to recover more of the benefit which history provides, without incurring the same visitation requirement as the tabular agent.

6.2 PPO

For the implementation of the PPO algorithm, we refer to the CleanRL code, with slight modifications regarding the structure of the neural network, to adjust for the small state space of our problem when compared to the Atari state space.

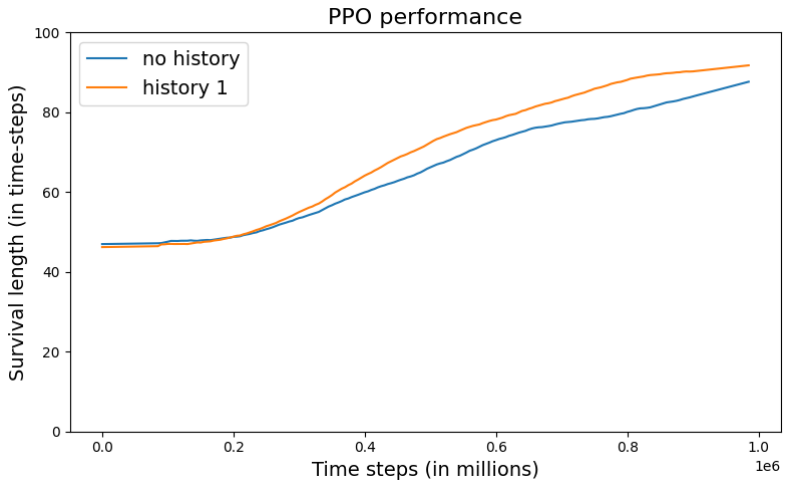


Figure 6: PPO algorithm on 3x3 lattice with a bit-flip error rate of 0.01 with 0.01 drift, comparing a PPO agent that uses only the current observation with a PPO agent that uses both the current and previous observation as a concatenated input. Here, we see that history is able to improve the survival length of the agent after around 200 thousand time-steps.

We see in Figure 6 that, firstly, the PPO agent achieves a much higher survival length than the tabular Q-learning agent. Furthermore, we see that the history agent reaches a final survival length of approximately 91.76 time-steps, whereas the no history agent reaches approximately 87.65 time-steps. As hypothesized, in contrast to the tabular setting (without exploring starts), the history agent now outperforms the no history agent.

Whereas the tabular history agent was unable to outperform the no history agent, due to the exponential growth of the state space and the resulting insufficient coverage, the PPO history agent is able to generalize across the high-dimensional history input using the neural network, and therefore makes effective use of the additional information which the history provides. Therefore, the poor performance of the history agent in the tabular setting was indeed caused by insufficient coverage of the state space, rather than by the history observation being uninformative.

We must also consider that the improvement in survival is not very large. Therefore, and having established that the PPO history agent is able to outperform the no history agent, we now investigate whether providing further history continues to improve performance. We extend the history depth to two and three previous time-steps. It should first be noted

that throughout this section, the history-k agent refers to the agent which is given the k previous syndrome observations in addition to the current observation, so the history-2 agent observes the two previous time-steps together with the current one, and the history-3 agent the three previous time-steps.

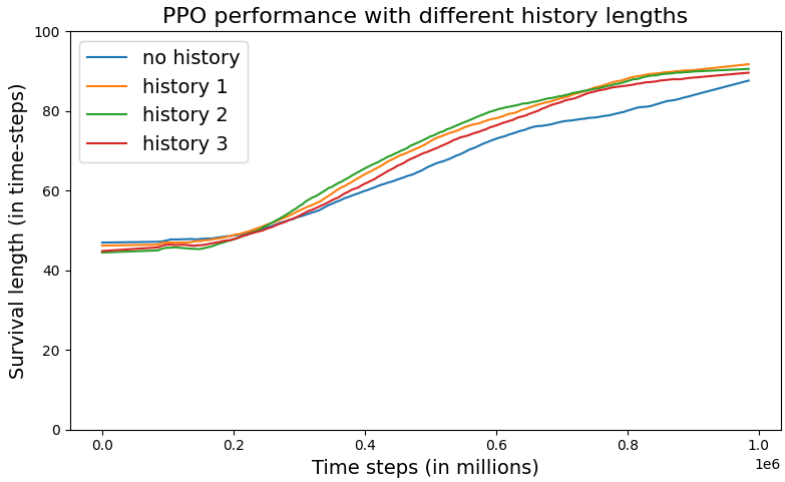


Figure 7: PPO algorithm performance on a 3x3 lattice with a bit-flip error rate of 0.01 with 0.01 drift, for history depths $k \in \{0, 1, 2, 3\}$. We see that, still, the best performing agent is the one with only a single previous observation (history-1) taken into account.

The final learning curves of the four agents are shown in Figure 7. We see that all three history agents outperform the no history agent (87.65), confirming that history is beneficial. The benefit does not, however, grow with depth, as the history-1 agent achieves the highest survival length (91.76), and adding further history slightly reduces it (90.58 for history-2 and 89.64 for history-3).

We want to understand the cause of this result, to know how much an RL agent stands to gain in terms of QEC logical error prevention through the inclusion of previous observations. However, the results in Figure 7 could be due to two different underlying issues. Given our observations, we may conclude that a single step of history is able to capture all useful temporal information in this setting, and deeper histories give diminishing, or slightly negative returns, likely because the additional observations grow the observation space (too much, even for the PPO agent) without adding exploitable information. This is a likely scenario considering that the most recent observation is indeed the most important

one, and any previous observations are simply used to try to better understand the current observation.

Contrastingly, it is also possible that deeper history would improve the performance further, but that it would only be the case for other experiment configurations. This may include an increased redundancy that would occur with a larger surface code, unlike ours with $d = 3$. Another possible explanation is that the deeper history agents have not yet converged after 1 million training time-steps, and would perhaps outperform the history-1 agent with more time.

That is, we cannot yet determine with certainty that increasing the number of previous observations does not continue to benefit the agent beyond a single history observation. Looking at the curves in Figure 7, they are still increasing towards the end of the plot. And indeed, the agent can reach a maximum of 100, so that it is still possible for the deeper history agents to overtake the single time-step history agent. For these reasons, it is only possible to know which of these cases is the right one by running the experiment for longer (or on a larger physical lattice).

6.3 Final Comparison

Finally, let us observe the overall comparison of all methods and their performance after an equal number of training time-steps.

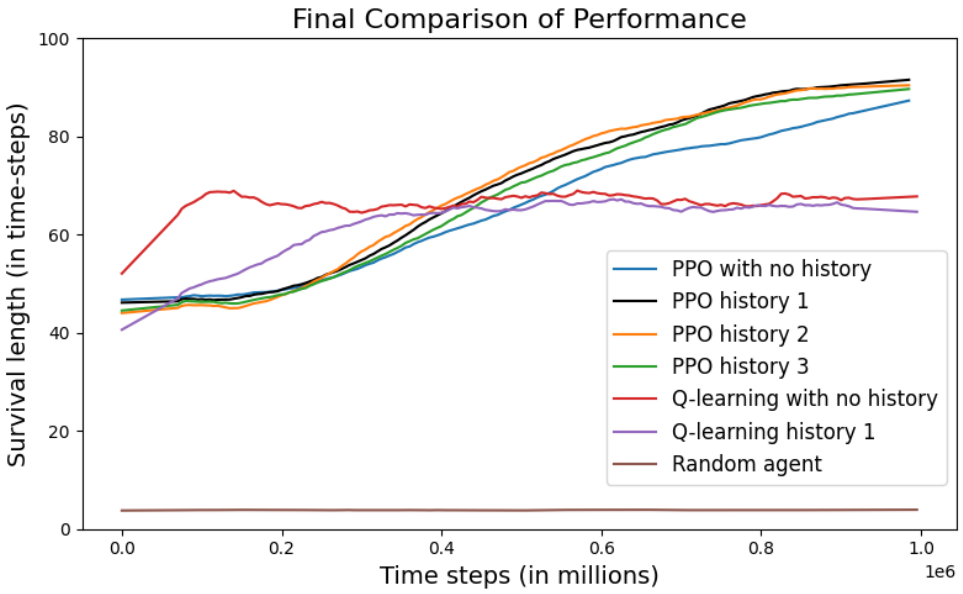


Figure 8: PPO with history depths $k \in \{0, 1, 2, 3\}$, and Q-learning algorithm with history depths $k \in \{0, 1\}$ performance on a 3x3 lattice with a bit-flip error rate of 0.01 with 0.01 drift. Indeed, the best performing agent is PPO history-1, marked in black, which achieves a final average survival length of 91.8 time-steps, as opposed to the worst-performing agent (except the random agent), the Q-learning history-1 agent reaching an averaged 64 time-steps, as can be seen in the purple line.

As before, the PPO agent with a single history observation is (somewhat marginally) the best performing of all agents, reaching a survival length of around 91 time-steps on average, closely followed by the history-2 and history-3 PPO agents. So, the agent which is best able to decode the rotated surface code is one which receives exactly one additional observation to the current one, and is able to generalize across similar states.

Moreover, as mentioned in the motivation for switching to the PPO agent from the Q-learning agent, a tabular RL agent is unable to generalize its knowledge. This is also reflected in the fact that, as can be seen in Figure 8, the Q-learning curves are much noisier than those of the PPO agent. We attribute this difference to the way each method represents and updates its policy. The tabular agent maintains a separate, independent value estimate for every state-action pair, and its evaluation policy is the arg max over these estimates. Because each estimate is updated from single transitions with a constant learning rate $\alpha = 0.1$, the values do not settle to a fixed point but continue to fluctuate

around it, as every update includes a high-variance, bootstrapped target. Whenever two competing actions in a given state have nearby values, such a fluctuation can flip the arg max for that state, which changes the evaluated behaviour. This effect is amplified by the low-error rate. As established in our analysis of the visitation tables, the agent spends almost all of its time in a small number of low-error states, so the evaluated survival length is dominated by the action selected in those few states. A single flip of the greedy action in, for example, the no-error state therefore produces a larger jump in survival length, which manifests as the noise seen in Figure 8. The PPO agent, by contrast, represents its policy with a neural network whose weights are shared across all states. A gradient update thereby adjusts the policy smoothly and globally rather than rewriting individual state-action values, and the resulting action distribution is a continuous function of the network parameters that shifts gradually over training. All of this produces the smoother learning curves observed for PPO.

In Table 6, we see the final survival length of all six agents.

History depth	**Final survival length**
Q-learning no history ($k = 0$)	66.60
Q-learning history 1 ($k = 1$)	64.01
PPO no history ($k = 0$)	87.65
PPO history-1 ($k = 1$)	91.76
PPO history-2 ($k = 2$)	90.58
PPO history-3 ($k = 3$)	89.64

Table 6: Final survival length of both agents at each studied history depth, after 1,000,001 training time-steps, averaged over 15 runs (seeds).

As previously stated, we see a clear improvement between Q-learning and PPO, and a further, smaller improvement when introducing a previous observation to the PPO agent, with slightly negative results when adding more previous observations. We see that the best agent is able to make it quite close to the maximum number of time-steps before it incurs the logical error. The PPO curves seem to continue improving also in later time-steps. This means it is certainly a possible next step to consider if the agent can eventually reach the entire 100 evaluation time-steps without a logical error occurring when given

more training time-steps. While the results are not compared to classical methods such as MWPM, it must be considered that RL agents are better able to adapt to the noise drift, and to accumulating physical errors, such that a direct comparison of survival length may not be sufficient. That being said, a comparison of our RL method with the classical methods would still make for interesting future research.

7 Discussion

7.1 Reward formulation

Possibly one of the most important components of the environment is the formulation of the reward. Initially, we used a pure survival reward during training, namely, the agent received a reward of +1 for every time-step in which it had not yet incurred a logical error, and a large penalty for incurring a logical error. However, this reward proved to be insufficiently informative for the agent to learn from. Because a logical error occurs only rarely at an error rate of 0.01, the agent receives an almost constant reward of +1 regardless of which action it selects, such that there is very little signal to distinguish a good correction from a harmful one. Furthermore, the large penalty is received only at the very end of an episode, so it is both rare and far removed from the individual action which caused the logical error. This means the agent is unable to attribute the penalty to the specific action responsible for it. For this reason, we introduced the shaped training reward defined in Equation (3), which explicitly rewards the agent for removing physical errors and punishes it for introducing them, by means of the net change in physical bit-flip errors Δe. This shaped reward is only possible during training, where the agent receives implicit knowledge of the underlying true error configuration rather than the syndrome alone. The same restriction would also go for real quantum computing devices, on which the agent would also not have access to the underlying configuration.

A second possible drawback of this reward shaping is that it is indeed focused on physical errors, rather than logical errors, unlike the evaluations of the agents. While normally fewer physical errors reduce the risk of a logical error, this is not guaranteed to be the case. This may be avoided by introducing heuristic reward shaping, which considers closeness to a logical error instead, and is a method that can be integrated with Actor-Critic DRL algorithms [WXX+25].

While this reward was sufficient for the agent to learn, it was hand-designed and we did not tune the specific coefficients (the 0.4 and 0.2 weightings), but rather decided on them, by a short comparison to other weightings. A natural next step would be to investigate the effect of the reward shaping more carefully, in order to determine whether a different formulation allows the agent to learn more quickly or to reach a higher survival length.

7.2 Evaluation over episodes versus time-steps

A second consideration concerns the manner in which we measure the performance of the agent over the course of training. Originally, we evaluated the agents after a fixed number of episodes, rather than after a fixed number of time-steps. This proved to lead to misleading results. Because the history agent initially performs very poorly, its episodes terminate quickly, such that it experiences far fewer time-steps per episode than the no history agent. Therefore, when evaluating per episode, the history agent receives considerably fewer learning updates than the no history agent within the same number of episodes, and as a result it improves much more slowly along the episode axis. This gives the false impression that the no history agent is substantially better, when in fact the difference is largely an artifact of the history agent having had less experience to learn from. The initial performance of the agent was preventing it from improving sufficiently. While the performance of the history agent would, in time, match the performance of the no history agent, the learning would have to be run for an incredibly large number of episodes for these results, and the resulting plot would lead to misleading conclusions. For the history agent to overtake the no history agent would very likely require an infeasible number of learning episodes (considering the run-time of the no history agent, which would have relatively long episodes from the beginning). For this reason, we instead evaluate all agents as a function of the total number of environment time-steps, which ensures that both agents are compared on equal grounds in terms of the amount of experience they have gathered.

A related consideration is whether to measure the agent's training performance, which therefore includes exploration, or its testing performance, which is greedy. In this work, all reported learning curves, in both Q-learning and PPO, are obtained from independent evaluation episodes in which exploration is switched off and the agent acts greedily from a clean start, so that the curves reflect test performance rather than the noisier behaviour of

the agent during training. This was done differently from the CleanRL implementation of PPO, in order to have a similar comparison between the two methods. Reporting the training return with exploration active would give a different, and generally lower performance.

7.3 Future work

Several directions remain open for future work. The most natural first step would be a hyperparameter study to optimize the learning of the PPO agent. In particular, the learning rate and the entropy coefficient, which we expect to have a considerable effect on both the speed of learning and the final survival length of the PPO agent. The entropy coefficient is important in our setting, since it controls how strongly the agent is encouraged to continue to explore rather than committing early to a single action. Naturally, in an environment where the agent spends almost all of its time in the low-error region, an overly low entropy coefficient may cause the agent to prematurely settle on the no-operation action, whereas an overly high one may prevent it from choosing the already-known and correct best action. The learning rate, similarly, controls the trade-off between the speed of convergence and the stability of the policy updates, such that a specific tuning of both parameters would allow us to determine the configuration best suited to the QEC setting, rather than relying on the values inherited from the CleanRL implementation. An approach similar to that used for the ε value in Q-learning can also be considered, where instead of static hyperparameter values, they adapt over time.

A further direction would be to introduce supervised learning into the setting, both on its own and in combination with reinforcement learning by re-using the same network for both objectives (supervised and reinforcement learning), in order to compare a purely supervised learning agent against one which is additionally improved through RL. In the supervised setting, the network would be trained to predict the correction directly from the syndrome, using the underlying true error configuration as the target label, which is available during training in the same way that it is available to our shaped reward. That is, rather than learning only through trial and error, the agent would first be given explicit examples of which correction to apply, and the RL could then refine this initial policy on the cases in which the supervised learning agent achieves a lower survival rate. This combination could be helpful because it would address the low initial performance of the agent, by providing

a better starting policy before the RL begins.

It would also be interesting to investigate the scalability of the QEC approach on larger code distances. Naturally, the tabular setting would not be able to cope with a larger lattice, even without history, due to the exponential growth of the state space, yet such an experiment could give further insight into how helpful history is in the PPO setting specifically. As shown in Table 1, the number of states already grows rapidly with the history depth at $d = 3$, and increasing the code distance compounds this further, since the number of Z-stabilizers itself grows as $\frac{d^2-1}{2}$. Therefore, larger distances are likely to be the setting in which the generalization ability of PPO would become valuable, and in which the gap between the tabular and function-approximation approaches is expected to be most noticeable. Furthermore, a larger lattice is the more realistic setting, since a greater code distance allows more physical errors to be corrected before a logical error occurs, which would also bring the environment closer to a real-life application.

As discussed in our analysis of the exploring starts, a completely random initial state is not well matched to the states that the agent encounters during evaluation. Therefore, a more successful variant would likely be a semi-exploring start, in which each episode begins from a state in which each qubit carries an error with some small probability p_x, so that the starting states remain closer to the low-error region the agent reaches in evaluation. This would maintain the benefit of exploring starts, which is the increased coverage of the state space, while avoiding the problem we identified, namely that the agent spends a large fraction of its updates on high-error states which it will rarely reach when following its policy at the true error rate. By setting this probability equal to the error rate of the environment, the distribution of starting states would be closer the distribution the agent encounters during evaluation. Then, the additional coverage would be concentrated on the states which are relevant to the agent's performance. This would also be an important consideration because we did, in fact, see that the history agent outperforms the no history agent when exploring starts are implemented.

Given that history is the main consideration of this work, it would also be a natural next step to consider network architectures which are designed to process sequential input, such as recurrent neural networks, or transformers, rather than the feed-forward network used here. These alternatives address this temporal dependency differently, where a recurrent network maintains a hidden state that summarizes an arbitrarily long history without a

fixed window [Tea18], and a transformer uses attention to model dependencies between any two positions in the sequence directly [ANT+23]. One more specific example of an RNN that could be successful here would be Mamba, which reaches state-of-the-art performance in many different long-sequence processing tasks, and which scales linearly with sequence length [GD24]. This may be especially helpful for the inclusion of more history time-steps. In the current implementation, the history is provided to the feed-forward network as a flat concatenation of the most recent syndromes, such that the temporal structure of the sequence is not clear to the network architecture. A recurrent network or a transformer, by contrast, is designed to model dependencies across a sequence, and could therefore make better use of the order in which the syndromes occurred. This is especially interesting given our earlier observation that the agent benefits from recognizing transition patterns regardless of their position in the history window. Furthermore, the syndrome measurements are not an unordered vector but have fixed positions on the two-dimensional lattice, a structure that the current implementation of a flat-vector input ignores. Making use of this spatial structure, for example by using a convolutional neural network ,could allow the agent to make use of the fact that neighbouring stabilizers correspond to physically adjacent qubits and that bit-flip errors act locally.

Finally, in this work we do not directly compare the performance of the RL agents to classical decoders of QEC. To know the real value of these algorithms, one should evaluate other decoders on the same lattice configuration. Examples include MWPM, but also an agent which does nothing, to see when the environment would reach a logical error on its own.

8 Conclusion

In this thesis, we investigated the use of reinforcement learning as a decoder for quantum error correction on a rotated surface code. We formulated QEC as a partially observable Markov decision process, in which an agent observes the syndromes and applies bit-flip corrections to prevent a logical error from occurring for as long as possible, under a realistic physical error rate that drifts over time. Within this setting, the central question we set out to answer was how much an RL agent stands to gain from being given past observations. We addressed this question using a tabular method, Q-learning, and a function-approximation

method, PPO.

Our results show, firstly, that the function-approximation agent substantially outperforms the tabular agent, with PPO reaching a survival length of around 88 to 92 time-steps, as opposed to around 64-67 with Q-learning. Secondly, to answer the question, we note that the two methods exploit observation history to different extents. For the tabular agent, adding a single step of history did not improve performance, and slightly decreased the final survival length (64.01 compared to 66.60 without history). This is because the history-including state space grows exponentially, and the agent cannot visit enough state-action pairs to form reliable value estimates within a fixed training budget. We further found that exploring starts reduces the tabular agent's survival length. Yet, it allows the history agent to outperform the no history agent, which can be an interesting direction for future research. For the PPO agent, by contrast, a single step of history did improve performance (91.76 against 87.65). This confirms that the history observation is, at least somewhat, informative, and that the tabular agent's failure to benefit from it stemmed from insufficient state coverage. Adding further history beyond a single step, however, had a slightly negative effect, returning 90.58 for two steps and 89.64 for three. This suggests that at code distance $d = 3$ and 1 million training time-steps, a single previous observation already captures most of the information that is useful for this task. We also observed that the PPO learning curves are smoother than those of Q-learning, despite equal smoothing windows, which we attribute to PPO's shared-weight generalization, and policy parametrization, in contrast to the tabular agent's state-action-specific noisier value estimates.
Taken together, these findings indicate that, given a rotated surface code with drifting error, observation history is beneficial for RL-based QEC, but to see this benefit requires a method capable of generalizing across observations.

Our study also has several limitations which can be improved on in future work. First, the shaped training reward relies on true physical errors, which are available only during training. Furthermore, this reward targets physical rather than logical errors, using manually chosen weights rather than optimized parameters. Second, the experimental results could likely be improved through hyperparameter optimization and the exploration of alternative neural network architectures, as the current configuration was largely inherited from an existing implementation. Finally, all experiments were carried out at a single code distance, $d = 3$,

though the generalization advantage of PPO may become more pronounced at larger, more realistic code distances.

References

[AFL21] Mikaela Algren, Wendy Fisher, and Amy E. Landis. Chapter 8 - machine learning in life cycle assessment. In Jennifer Dunn and Prasanna Balaprakash, editors, *Data Science Applied to Sustainability Analysis*, pages 167–190. Elsevier, 2021.

[AJLG19] Philip Andreasson, Joel Johansson, Simon Liljestrand, and Mats Granath. Quantum error correction for the toric code using deep reinforcement learning. *Quantum*, 3:183, 2019.

[ANT+23] Sabeen Ahmed, Ian E. Nielsen, Aakash Tripathi, Shamoon Siddiqui, Ravi P. Ramachandran, and Ghulam Rasool. Transformers in time-series analysis: A tutorial. *Circuits, Systems, and Signal Processing*, 42(12):7433–7466, 2023.

[BBB+26] Gilles Buchs, Thomas L. Beck, Ryan S. Bennink, Daniel Claudino, Andrea Delgado, Nur Aiman Fadel, Peter Groszkowski, Kathleen E. Hamilton, Travis S. Humble, Neeraj Kumar, Ang Li, Phillip C. Lotshaw, Olli Mukkula, Ryousei Takano, Amit Saxena, In-Saeng Suh, Miwako Tsuji, Roel Van Beeumen, Ugo Varetto, Yan Wang, Kazuya Yamazaki, and Mikael P. Johansson. The role of quantum computing in advancing scientific high-performance computing: A perspective from the adac institute. *Future Generation Computer Systems*, 182:108487, 2026.

[Ber] Dave Bergmann. What is deep learning? — ibm.

[BSH+24] Johannes Bausch, Andrew Senior, Francisco Heras, Thomas Edlich, Alex Davies, Michael Newman, Cody Jones, Kevin Satzinger, Yuezhen Niu, Sam Blackwell, George Holland, Dvir Kafri, Juan Atalaya, Craig Gidney, Demis Hassabis, Sergio Boixo, Hartmut Neven, and Pushmeet Kohli. Learning high-accuracy error decoding for quantum processors. *Nature*, 635:834–840, 11 2024.

[CM23] Mohamed-Amine Chadi and Hajar Mousannif. Understanding reinforcement learning algorithms: The progress from basic q-learning to proximal policy optimization, 2023.

[CMR22] Kittisak Chumpong, Khamron Mekchay, and Sanae Rujivan. A simple closed-form formula for the conditional moments of the ornstein-uhlenbeck process. *Songklanakarin Journal of Science and Technology*, 42:836–843, 03 2022.

[CPG23] Avimita Chatterjee, Koustubh Phalak, and Swaroop Ghosh. Quantum error correction for dummies. *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, page 70–81, Apr 2023.

[DMN13] Simon J Devitt, William J Munro, and Kae Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, Jun 2013.

[FLHI+18] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3–4):219–354, 2018.

[GD24] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024.

[Got97] Daniel Gottesman. Stabilizer codes and quantum error correction, 1997.

[Ion17] Radu Ionicioiu. Schrödinger's cat: Where does the entanglement come from? *Quanta*, 6:57–60, Sep 2017.

[MBM+16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

[ME22] Eduardo F. Morales and Hugo Jair Escalante. Chapter 6 - a brief introduction to supervised, unsupervised, and reinforcement learning. In Alejandro A. Torres-García, Carlos A. Reyes-García, Luis Villaseñor-Pineda, and Omar Mendoza-Montoya, editors, *Biosignal Processing and Classification Using Computational Learning and Intelligence*, pages 111–129. Academic Press, 2022.

[Mic24] Microsoft Quantum. Quantum error correction. https://quantum.microsoft.com/en-us/insights/education/concepts/quantum-error-correction, 2024. Accessed: June 27, 2026.

[PZN26] Yotam Peled, David Zenati, and Eliya Nachmani. Neural minimum weight perfect matching for quantum error codes, 2026.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

[SBB+24] Marnix Suilen, Thom Badings, Eline M. Bovy, David Parker, and Nils Jansen. Robust markov decision processes: A place where ai and formal methods meet, 2024.

[SKvNE20] Ryan Sweke, Markus S Kesselring, Evert P L van Nieuwenburg, and Jens Eisert. Reinforcement learning decoders for fault-tolerant quantum computation. *Machine Learning: Science and Technology*, 2(2):025005, December 2020.

[SML+15] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, page 5, 2015.

[Spa12] Matthijs Spaan. Partially observable markov decision processes. *Adaptation, Learning, and Optimization*, 01 2012.

[Spo24] Lindsay Spoor. Quantum error correction on the toric code using two distinct reinforcement learning game frameworks, Mar 2024.

[SS24] Josh Schneider and Ian Smalley. What is a qubit? — ibm, Feb 2024.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Tea18] Ahmed Tealab. Time series forecasting using artificial neural networks methodologies: A systematic review. *Future Computing and Informatics Journal*, 3(2):334–340, 2018.

[TFG+25] Cewen Tian, Zaixu Fan, Xiaoxuan Guo, Xinying Song, and Yanbing Tian. Transformer-based quantum error decoding enhanced by qgans: towards scal-

able surface code correction algorithms. *EPJ Quantum Technology*, 12, 06 2025.

[WXX+25] Yizhi Wang, Yongfang Xie, Degang Xu, Jiahui Shi, Shiyu Fang, and Weihua Gui. Heuristic dense reward shaping for learning-based map-free navigation of industrial automatic mobile robots. *ISA Transactions*, 156:579–596, 2025.

[WYSR22] Che Wang, Shuhan Yuan, Kai Shao, and Keith Ross. On the convergence of the monte carlo exploring starts algorithm for reinforcement learning, 2022.