DeepQ Decoding for Fault Tolerant Quantum Computation

Ryan Sweke,[1] Markus S. Kesselring,[1] Evert P.L. van Nieuwenburg,[2] and Jens Eisert[1]

[1]*Dahlem Center for Complex Quantum Systems, Freie Universität Berlin, 14195 Berlin, Germany*
[2]*Institute for Quantum Information and Matter, Caltech, Pasadena, CA 91125, USA*
(Dated: October 7, 2018)

Topological error correcting codes, and particularly the surface code, currently provide the most feasible roadmap towards large-scale fault tolerant quantum computation. As such, obtaining fast and flexible decoding algorithms for these codes, within the experimentally relevant context of faulty syndrome measurements, is of critical importance. In this work we show that the problem of decoding such codes, in the full fault tolerant setting, can be naturally reformulated as a process of repeated interactions between a decoding agent and a code environment, to which the machinery of reinforcement learning can be applied to obtain decoding agents. As a demonstration, by using deepQ learning, we obtain fast decoding agents for the surface code, for a variety of noise-models, within the fully fault tolerant setting.

I. INTRODUCTION

In order to implement large scale quantum algorithms it is necessary to be able to store and manipulate quantum information in a manner that is robust to the unavoidable errors introduced through interaction of the physical qubits with a noisy environment. A useful strategy for achieving such robustness is to encode a single logical qubit into the state of many physical qubits, via a quantum error correcting code, from which it may be possible to actively diagnose and correct errors that may occur [1, 2]. While many quantum error correcting codes exist, topological quantum codes [1–8], in which only local operations are required to diagnose and correct errors, are of particular interest as a result of their experimental feasibility [9–15]. Recently the surface code has emerged as an especially promising candidate for large-scale fault-tolerant quantum computation, due to the combination of its comparatively low overhead and locality requirements, coupled with the availability of convenient strategies for the implementation of all required logical gates [16, 17]

In any code-based strategy for fault tolerant quantum computation, decoding algorithms play a critical role. At a high level, these algorithms take as input the outcomes of syndrome measurements (which provide a diagnosis of errors that occurred on the physical qubits), and provide as output a suggestion of corrections for any errors that may have occurred during the computation.

It is particularly important to note that in any physically realistic setting the required syndrome measurements are obtained via small quantum circuits, and are therefore also generically faulty As such, while the setting of perfect syndrome measurements provides a good test-bed for the development of decoding algorithms, any decoding algorithm which aims to be experimentally useful should also be capable of dealing with such faulty syndrome measurements. Additionally, such algorithms should also be capable of dealing with experimentally relevant noise models, as well as be fast enough to not present a bottleneck to the execution of computations, even as the size of the system (i.e. the code distance) grows.

Due to the importance of decoding algorithms for fault tolerant quantum computation, many different approaches have been developed, each of which tries to satisfy as many of the experimentally required criteria as possible. Perhaps most prominent are algorithms based on minimum-weight perfect matching subroutines [18], although alternative approaches based on techniques such as the renormalization group [19] and locally operating cellular automata [20–22] have also been put forward.

Recently, techniques from machine learning have begun to find application in diverse areas of quantum physics - such as in the efficient representation of many-body quantum states [23–25], the identification of phase transitions [26–28], and the autonomous design of novel experimental set-ups [29] - and various neural-network based decoders have also been proposed [30–37]. Despite this diversity of proposed decoding algorithms, there is as of yet no algorithm which clearly satisfies all the required experimental criteria listed above. While previously proposed neural network decoders promise extremely fast decoding times [37], flexibility with respect to the underlying code and noise model [33, 37] and the potential to scale to large code distances [36], most neural network decoders are so far restricted to the setting of perfect syndrome measurements. Neural network decoders taking faulty syndrome measurements into account [34, 35] currently restrict themselves to the setting of preserving a logical qubit for as long as possible, without the requirement of performing a subsequent logical gate. As such, while neural network based approaches to decoding seem promising, there remains room for improvement and new techniques.

Simultaneously, the last few years have also seen impressive advances in the development of deep reinforcement learning algorithms, which have allowed for the training of neural network based agents capable of obtaining superhuman performance in domains such as Atari [38–41], Chess [42] and Go [43, 44]. These techniques are particularly powerful in situations where it is necessary to learn strategies for complex sequential decision making, involving consideration of the future effects of ones actions. In this work we demonstrate that decoding in the context of fault-tolerant quantum computing provides a natural

setting for the application of reinforcement learning methods. We will discuss the advantages of such an approach over the previously proposed neural network decoders.

In particular, we interpret the decoder as an agent performing discrete, sequential actions on an environment which is defined by a quantum error correction code. This conceptual framework allows for the application of various deep reinforcement learning algorithms to obtain neural network based decoding agents. As a demonstration, we then utilize to deepQ learning to obtain fast surface code decoders, for a variety of noise models, in the fully fault-tolerant setting. These results provide a foundation for extension via both more sophisticated reinforcement learning techniques and neural network models.

In this work we begin by providing an introductory overview of the surface code in Section II, before presenting a description of the decoding problem for fault-tolerant quantum computation in Section III. After a brief introduction to the formalism of reinforcement learning and Q-functions in Section IV we are then able to provide the conceptual reframing of decoding as a reinforcement learning problem in Section V, representing one of the primary results of this work. In Section VII we then present deepQ surface code decoders for a variety of noise models, before finally in Section VIII we discuss both the advantages and disadvantages of the approach presented here, along with various potential strategies for building upon the results presented in this work.

II. THE SURFACE CODE

We begin by providing a brief description of the surface code. The framework and methods presented in this work are not restricted to the surface code however, and may be applied to any stabilizer code. The restriction to the surface code is made both for simplicity of presentation and experimental relevance. We will focus on presenting the essential elements of the surface code and refer to more complete treatments for details [1, 17, 45].

We will consider $d \times d$ lattices with a *physical* data qubit on each vertex v, as illustrated in Fig. 1 for $d = 5$. The collective state of all qubits on the lattice is an element of the Hilbert space $\mathcal{H} = \mathbb{C}^{2^{(d \times d)}}$. We associate stabilizer operators with each coloured plaquette of the lattice. Stabilizers on blue (orange) plaquettes are operators which apply Pauli X (Z) flips to all qubits on the vertices of the plaquette. Specifically, denoting the set of all blue (orange) plaquettes as B_p (O_p) we define the stabilizer S_p on plaquette p as,

$$S_p = \bigotimes_{v \in p} \sigma_v \quad \text{where} \begin{cases} \sigma_v = X_v & \text{if } p \in B_p, \\ \sigma_v = Z_v & \text{if } p \in O_p. \end{cases} \quad (1)$$

All stabilizers are mutually commuting and have eigenvalues ± 1. This allows for the introduction of a Hamiltonian $H = -\sum_p S_p$ from which the surface code $\mathcal{H}_{\text{sc}} \subset \mathcal{H}$ is
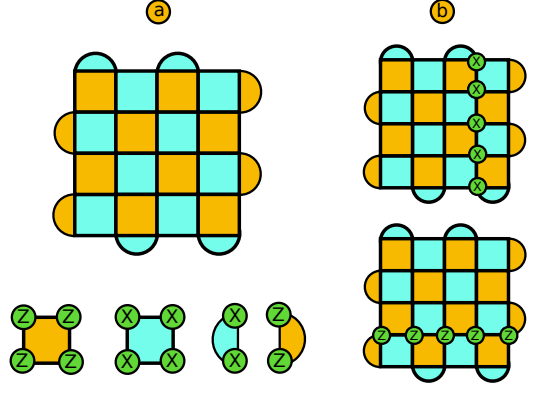

FIG. 1. An overview of the 5×5 surface code. (a) We consider square lattices, with a physical data qubit on each vertex. The coloured plaquettes indicate stabilizer operators as defined in Eq. (1). (b) Logical X_L and Z_L operators for the surface code are given by continuous strings of single qubit X or Z operators connecting the top and bottom or left and right boundaries of the code respectively.

defined to be the ground state space of H. Alternatively, this space consists of all simultaneous $+1$ eigenstates of all stabilizers. This subspace is two dimensional, i.e. $\mathcal{H}_{\text{sc}} \simeq \mathbb{C}^2$, and hence can encode a single *logical* qubit. Logical operators are operators which preserve the code space, and can therefore be used to manipulate the state of the logical qubit. Fig. 1 shows logical X (Z) operators, denoted X_L (Z_L), which are continuous strings of single vertex X (Z) operators connecting the top and bottom (left and right) boundaries of the lattice.

To illustrate the motivation behind such an encoding, let's examine the consequences of a single qubit Pauli flip on a physical data qubit. If we assume that the initial state $|\psi\rangle \in \mathcal{H}_{\text{sc}}$ is an element of the code space, then the subsequent state $|\psi'\rangle \notin \mathcal{H}_{\text{sc}}$ will no longer be an element of the code space. In particular, $|\psi'\rangle$ will be an eigenstate with eigenvalue -1 of at least one stabilizer. We say that $|\psi'\rangle$ violates these stabilizers, as illustrated by red circles in Fig. 2 (a). The *syndrome* of a state is a list of the outcomes of a simultaneous measurement of all the stabilizers, each of which takes the value ± 1. Given that a single Pauli flip occurred on a single physical data qubit, by analysing the syndrome we may be able to identify and correct this error, in the process conserving the logical qubit state. This process of *decoding* is discussed in the next section.

III. THE DECODING PROBLEM

With the foundations from the previous section, we can now formulate a simple preliminary version of the decoding problem:

Assume that at $t = 0$ one is given a state $|\psi\rangle = \alpha|0_L\rangle + \beta|1_L\rangle \in \mathcal{H}_{\text{sc}}$. At some time $t_1 > 0$

a syndrome measurement is performed which indicates that one or more stabilizers are violated - i.e. some errors have occurred on physical data qubits. From the given syndrome, determine a set of corrections which should be applied to the code lattice such that the subsequent state $|\psi'\rangle$ is equal to the initial state $|\psi\rangle$.

Before proceeding to discuss more subtle and technical versions of the decoding problem, let's examine why even the above problem is indeed potentially difficult. The most important observation is that the map from error configurations to syndromes is many-to-one, i.e. many different sets of errors can lead to the same syndrome. As an example, consider the error configurations illustrated in Fig. 2 (b-d), all of which lead to the same syndrome. If the probability of an error on a single physical data qubit is low, given such a syndrome one might reasonably assume that the error shown in (c) occurred, as one error on a single qubit is a more likely event than errors on multiple qubits. Given this reasoning, one might then suggest to correct by applying an X flip on the physical data qubit in the third row and fourth column. If indeed the error shown in (c) occurred, the post-correction state would be error-free, thus preserving the initial state. However, if the error pattern shown in (d) occurred, this set of errors combined with the applied X flip would implement a stabilizer. Since the original state was a simultaneous $+1$ eigenstate of all the stabilizers, and stabilizers act trivially on logical states, the proposed correction indeed preserves the initial logical state. Finally, if the error in (b) occurred, then the combination of the original error with the correction would implement the logical X_L operator. As a result, even though the post-correction state is back in the code space, it will be in a different logical state. Thus, the information we were trying to preserve would have been corrupted. From this simple example one can see that most often solving the decoding problem as stated above involves deciding, given an inherently ambiguous syndrome and (possibly imperfect) knowledge of the underlying error model, which error configuration most likely occurred.

In addition to the inherent difficulty resulting from syndrome ambiguity, in experimental settings the process of extracting the syndrome is also subject to noise, and hence the syndrome itself may be faulty [46, 47]. In practice, each stabilizer is associated with a physical ancilla qubit. The syndrome value for that particular stabilizer is obtained by first executing a small quantum circuit which entangles the ancilla with each of the physical data qubits on which the corresponding stabilizer is supported. The syndrome value is then extracted via a measurement of the ancilla qubit. In order to fully account for errors during the process of syndrome extraction one should therefore model this entire circuit, in which errors can occur on both the data qubits and ancilla qubits at each time step. Moreover, errors on the ancilla qubits can propagate onto the data qubits via the required entangling gates.
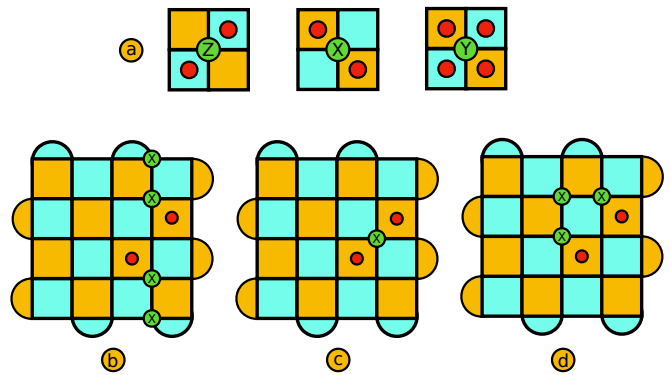

FIG. 2. (a) Single qubit Pauli flips violate surrounding stabilizers. (b-d) Strings of Pauli flips only violate stabilizers at the endpoint of the string. Multiple error configurations can give rise to the same syndrome. They can differ by stabilizers, as for example in (c) and (d), or by logical operators, see (b) and (c).

The essential aspect of the additional difficulty from faulty syndrome measurements can be phenomenologically modelled by imagining each time step as consisting of two distinct error processes [47], as illustrated in Fig. 3. In the first error process, an error occurs on each data qubit with some probability. One then imagines extracting the perfect syndrome before a second error process occurs, in which with a given probability an error occurs on each stabilizer measurement outcome. In this scenario, single syndrome measurements are hence no longer reliable. Decoding in the fault tolerant setting therefore typically requires providing a set of sequential syndrome measurements.

Finally, in the context of surface code based fault tolerant quantum computing, all logical gates are implemented either via protocols which also involve an inherent decoding procedure or do not spread errors. To be specific, it is sufficient for universal quantum computing to be able to implement both Clifford and T gates [48, 49]. In contemporary proposals for surface code based quantum computing [16, 17] protocols are known for implementing Clifford gates either by tracking, via code deformation [50] or via lattice surgery [51, 52]. Code deformation and lattice surgery requires several decoding cycles. Non-Clifford gates, such as the T gate, can be performed fault tolerantly via gate teleportation using magic states. High quality magic states can be obtained via magic state distillation, which requires only Clifford gates and faulty magic states [49]. As such the goal of decoding idling logical qubits within a quantum computation should be to suppress errors to the extent that any of the above procedures can succeed with high probability. Therefore, we can relax the requirement that the decoding process should return the post-error state to the initial state in the code space. In Section V we discuss a proxy criterion for decoding success within this framework.

FIG. 3. A typical decoding cycle is illustrated for the simplified faulty measurements scenario in which one imagines each time step consisting of an initial physical error process generating errors on the data qubits, followed by a second measurement error process which corrupts the true syndrome. The decoding algorithm then has access to a sequence of potentially faulty syndromes.

IV. REINFORCEMENT LEARNING AND Q-FUNCTIONS

In this section we shift focus and introduce some of the fundamental concepts of reinforcement learning and q-functions, which will be essential to our rephrasing of the decoding problem in Section V. Again, we will constrain ourselves to the basics and refer to Ref. [53] for a complete treatment. A generic reinforcement learning problem considers an agent interacting with an environment, as is illustrated in Fig. 4. The agent can act on and observe (parts) of the environment, and is tasked with achieving a problem-specific goal by performing a sequence of actions. We typically consider discrete problems, in which at each step time step t the environment can be described by a state $S_t \in \mathcal{S}$, where S is referred to as the state space. Given a state of the environment, the agent can then choose to perform an action $A_t \in \mathcal{A}$, where $\mathcal{A}$ is referred to as the action space. As a result of the agents chosen action, the environment then updates accordingly, entering a new state S_{t+1} and providing feedback to the agent on its choice of action in the form of a scalar reward R_{t+1}. We will restrict ourselves here to episodic environments, for which there exist a set of terminal states $\mathcal{S}_{\text{terminal}} \subset \mathcal{S}$. In such episodic settings, in addition to a scalar reward, the agent also receives a boolean signal indicating whether $S_{t+1} \in \mathcal{S}_{\text{terminal}}$ - i.e. whether or not it is "game over".

In general, the agent's choice of action, the resulting state of the environment and the returned reward can all be stochastic. In the case of finite state and action spaces, the environment can then be formalized via a finite Markov decision process (FMDP);

$$p(s', r|s, a) \equiv \text{pr}(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a). \quad (2)$$

To formalize the decision making process of the agent, we define an agent's *policy* π, in essence the agent's strategy, as a mapping from states to probabilities of specific actions - i.e. $\pi(a|s)$ is the probability that agent chooses $A_t = a$, given that the environment is in state $S_t = s$. For FMDP's we then define the *value* of a state s under policy π as,

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\Big|S_t = s\right] \quad (3)$$

$\forall S_t \in \mathcal{S}$. The term G_t is the discounted return (discounted cumulative reward), with discount factor $0 \leq \gamma \leq 1$, and is the quantity that the agent is tasked with optimizing. In episodic settings the infinite sum terminates whenever state S_{t+k+1} is a terminal state - i.e. $S_{t+k+1} \in \mathcal{S}_{\text{terminal}}$. We call v the *state-value function*, providing the expected discounted cumulative reward the agent would obtain when following policy π from state s. It is an important conceptual point to note that by using the metric of the discounted cumulative reward the value of any given state depends not only on the immediate reward obtained by following a specific policy from that state, but includes future expected rewards. Hence strategies which involve some element of successful future planning may lead to higher state values. As such, we see that the value of a state with respect to a given policy reflects accurately the ability of an agent to achieve its long-term goals when following that policy from that state.
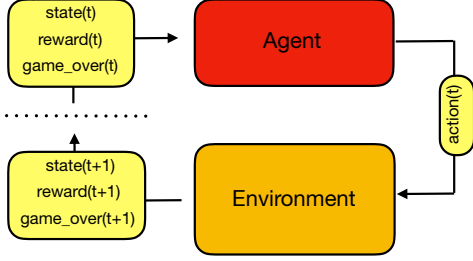
FIG. 4. An illustration of the signals passed between an agent and the environment through the duration of a sequential turn based episode.

Similarly to the *state-value function*, we can define the *action-value function* (referred to as the q-function) for policy π via:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{4}$$

$$= \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\Big|S_t = s, A_t = a\Big] \tag{5}$$

Clearly, the q-function with respect to a given policy is conceptually similar to the state-value function, differing only in that it provides the value for state-action pairs. Importantly, value functions allow us to place an order over policies, i.e. $\pi > \pi' \iff v_\pi(s) > v_{\pi'}(s) \quad \forall s \in \mathcal{S}$. This in turn allows us to define an optimal policy π^*, for which:

$$q_*(s,a) = \mathbb{E}\big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')\big|S_t = s, A_t = a\big]. \tag{6}$$

Note that given the optimal q-function it is easy to obtain the optimal strategy; In a given state s simply choose the action $a = \mathrm{argmax}_{a'}[q_*(s, a')]$.

Given this framework, there are many different approaches and methodologies that can be used to learn optimal policies. We will focus here on q-learning, the goal of which is to find or approximate $q_*(s, a)$. This is generically done via iterative q-learning, in which the agent starts with an arbitrary q-function and then iteratively refines it on the basis of experience gained from interaction with the environment. In particular, in order to generate such experience, the agent uses a policy derived from its q-function (possibly in addition to other explorative policies) to choose actions. This q-function is then periodically updated using Eq. (6), for which $q_*(s, a)$ is a stationary solution [53].

The above describes concisely the elements of q-learning, however it does not address the general impracticality of storing this q-function. In most real world applications, the number of valid state-action pairs can be impractically large (e.g. consider the number of possible chess configurations). It is precisely to address this problem, and in effect to render q-learning applicable in practice,

that *deepQ* learning was introduced [38–40]. In particular, in deepQ learning we parametrize q by a neural network, and use Eq. (6) to construct the cost function from which the network weights can be updated via a stochastic gradient descent method. The learning of the q-function is hence done by training a neural network in an online supervised manner, the training instances for which are generated by having the agent explore the environment. Specifically, we let the agent interact with the environment via an ϵ-greedy exploration policy, in the process generating experience-tuples of the following form:

$$[S_t, A_t, R_{t+1}, S_{t+1}] \tag{7}$$

The set of these tuples then provides an experience memory, from which we can periodically sample a training-batch. Given such a batch of training instances, the q-network is then updated via the cost function:

$$C = y_{\mathrm{pred}} - y_{\mathrm{true}} \tag{8}$$

$$= q(S_t, A_t) - \big[R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a')\big] \tag{9}$$

which, by comparison with Eq. (6), will be minimized only for the optimal policy. Unfortunately, despite the simplicity of this idea, in practice a variety of tricks - such as separate active and target q-networks, double-q learning and dueling networks - are required to achieve stable q-learning. We refer to the relevant references [38–41], or to the associated code repository [54], for details.

V. DECODING AS A REINFORCEMENT LEARNING PROBLEM

We now turn to describing the main topic of this work. Namely, we formulate the problem of decoding within the context of *fault-tolerant* quantum computation as a reinforcement learning problem. The advantage of this formulation is that it allows for all the methods and techniques of reinforcement learning to be brought to bear on this problem, thereby providing a new toolbox for obtaining diverse *decoding agents*, a novel class of decoding algorithms. To present this formulation, we will describe a single episode, as illustrated in Fig. 5.

At a high level, the environment is constructed from the surface code and the action space of the agent consists of all single qubit Pauli X or Z flips, along with an identity action. A single episode starts with the extraction of a sequence of faulty syndromes and ends whenever errors have accumulated to the extent that a subsequent logical operation is likely to fail, in a sense that will be made precise below. Specifically, as shown in Fig. 5 (a), the initial step ($t = 0$) of any episode, in which the initial state S_1, reward R_1 and terminal state indicator T_1 are generated, proceeds according to the following steps;

FIG. 5. An illustration of the various steps occurring within a single episode. (a) In an initial step, a faulty syndrome volume is extracted from a state initially in the code space. This faulty syndrome volume is combined with an initially empty action history and passed to the agent as the initial state. Given an input state, the agent must decide on an action, which it passes to the environment. If the action is neither an identity or a repetition of an action already in the action history (b), then the procedure illustrated in box (c) occurs, if a trivial or already performed action is chosen (d), then the procedure illustrated in box (e) occurs. (c) The chosen action is applied to the underlying quantum state. From this state the reward for the applied action is determined, while simultaneously a referee decoder decides whether or not the episode is now over - i.e. whether the underlying quantum state is now in a terminal state. Additionally, the applied action is appended to the action history, which is concatenated with the non-updated syndrome volume and provided to the agent, in conjunction with the reward and terminal state indicator. (e) Once again, the underlying quantum state is first updated, and from this updated state both a reward and terminal state indicator are determined (not shown). However, the trivial or repeated action of the agent triggers a new round of faulty syndrome measurements. After this new round of syndrome extraction, the new syndrome volume is combined with a reset action history and provided to the agent, from which it can once again choose another move.

1. First, the state of the system (consisting of physical data qubits), which we will refer to as the *hidden state* of the environment, is initialized to some logical state $|\psi_0\rangle \in \mathcal{H}_{\rm sc}$. The agent's task will be to preserve this state. Since this state is in the code space by construction, it can be simply represented as an empty list representing a code lattice on which no Pauli flips have been applied to any physical data qubits.

2. Then an initial sequence of faulty syndromes is generated. We utilize the two-stage error process involving separated physical and measurement errors, as described in Fig. III. We wish to emphasise however that in principle a circuit model for syndrome extraction could also be simulated. The errors on physical data qubits which were generated during the syndrome extraction protocol are then added to the hidden state of the environment, since we will need them later to evaluate the agent's choice of actions. The agent has no knowledge of, or direct access to, this hidden state.

3. The visible aspect of the environment, which the agent can observe, is then constructed from the sequence of faulty syndromes, illustrated by the green slices in Fig. 5. We will refer to this sequence of faulty syndromes as the *syndrome volume*, denoted $S_{E,1}$. In addition to the syndrome volume, the total state received by the agent at any time step also contains a list of the actions taken by the agent since the last new syndrome volume was received, which we illustrate in Fig. 5 with the yellow slices and refer to as the action history h. Since for this first step a new syndrome volume was just extracted, the action history of the agent is reset - i.e. h_1 is an empty list - and this reset action history is combined with the syndrome volume and provided to the agent as the total output state of the environment - i.e. $S_1 = (S_{E,1}, h_1)$.

4. The agent has not yet taken any actions, so the reward for this first step is $R_1 = 0$. Additionally, in order to allow the episode to continue the terminal state indicator to false - i.e. $T_1 = 0$.

After this initial step, we now need to perform step $t = 1$. In general, as illustrated in Fig. 4, given the state S_t, the agent needs to choose an action A_t to apply to the environment, after which the environment needs to respond with the tuple $(S_{t+1}, R_{t+1}, T_{t+1})$, where $S_{t+1} = (S_{E,t+1}, h_{t+1})$. Depending on the action chosen by the agent, the environment can respond in one of two ways, illustrated in Fig. 5 (c) and (e) respectively.

A. Non-identity and non-repeated actions

If, as shown in Fig. 5 (c), the agent chooses any non-identity action that is not already an element of the action

history list h_t, then this step of the episode continues via the following process;

1. Firstly, given the chosen non-trivial action A_t, the first step is to apply this action to the code lattice - i.e. the agents chosen action is added to the hidden state of the environment, which is now a combination of all actions and errors on physical data qubits.

2. The environment now needs to determine the reward R_{t+1} the agent should receive for its action, and whether resulting hidden state is a terminal state. As was discussed in Section 3, within the context of surface code based fault tolerant quantum computation the goal of the decoding process for idling logical qubits should be to suppress errors to the extent that future logical operations (which either involve decoding cycles of their own or do not spread errors) can succeed with high probability. As such, if after application of the agent's chosen action the current state of the system $|\psi\rangle$ is equal to the original state $|\psi_0\rangle$ - or equivalently, if all errors have been corrected without implementing a logical operation - we give the agent a reward of +1, i.e. $R_{t+1} = +1$. In all other cases we set $R_{t+1} = 0$, giving the agent no reward.

3. In addition to the above, we introduce a separate *referee* decoder to act as a proxy for future logical operations. This referee decoder should be a "single-shot" decoder which, given a single perfect syndrome, suggests corrections which always move the current state back into the code space. However, this referee decoder may fail by inadvertently performing a logical operation. After the hidden state of the environment is updated to include the agents chosen action, this referee decoder is then given a perfect syndrome extracted from the state $|\psi\rangle$ corresponding to the current hidden state. If the referee decoder is able to correctly decode this perfect syndrome then the hidden state is not a terminal state, and $T_{t+1} = 0$. If on the other hand the referee decoder is not able to correctly decode this perfect syndrome, then we set $T_{t+1} = 1$, which indicates the end of the episode.

4. Finally, the new total state of the environment, $S_{t+1} = (S_{E,t+1}, h_{t+1})$, is constructed as follows; Firstly, the action history h_{t+1} is obtained by appending A_t to h_t. Then, as no new syndrome volume was generated in this step, we set $S_{E,t+1} = S_{E,t}$.

B. Identity or repeated actions

If on the other hand the agent chooses either the identity action or an action it has already chosen, then the environment responds via the following procedure, as illustrated in Fig. 5 (e).

1. If the action A_t is a non-identity action which is already in the action history - i.e. $A_t \in h_t$, then the first step is to apply the action to the code lattice by appending the action to the hidden state. In this case, the reward R_{t+1} and terminal state indicator T_{t+1} are then determined by the procedure already described above, as in steps in which a non-identity action was taken.

2. If the chosen action was the identity however, then this hidden state update is not necessary, and given that the previous hidden state could not have been a terminal state we set $T_{t+1} = 0$, before setting $R_{t+1} = R_t$, as the hidden state has not changed.

3. Now, we interpret the performing of an identity (either explicitly via the identity action or implicitly by repeating a correction twice) by the agent as a request for a new syndrome, and hence a new syndrome volume is generated. Similarly to the initial step of the episode, the hidden state is then updated to include the errors on physical data qubits introduced via this syndrome extraction. Additionally, the action history h_{t+1} is once again reset to an empty list, while the visible state $S_{E,t+1}$ is updated to reflect the newly extracted syndrome volume. Finally, the total output state $S_{t+1} = (S_{E,t+1}, h_{t+1})$ is returned to the agent.

At this stage we have all the ingredients required to obtain decoding agents through the application of a variety of reinforcement learning algorithms. In Section VI we will present a detailed construction for a deepQ agent, that allows us to apply deepQ learning and obtain decoding agents whose performance is shown in Section VII. Before proceeding however it is worthwhile to emphasize a few points;

First, it is important to note that the above agent-interaction framework is both code and error model agnostic, provided one has access to a referee decoder, capable of single-shot decoding on perfect syndromes. From this perspective, one might view the framework presented here as a tool for leveraging single-shot, perfect-syndrome decoding algorithms, into decoding algorithms for the fully fault-tolerant setting. Additionally, it should be clear from the rules via which rewards are assigned, that the agent will only accumulate reward provided it can learn to consistently correct any errors that might occur on physical data qubits, without implementing logical operations. However, the agent is *not* constrained to return the state back into the code space between successive syndrome measurements. In particular, because typical reinforcement learning algorithms take into account not only immediate rewards, but rather discounted cumulative rewards, the agent may learn strategies involving actions whose benefit may not be immediate. For example, if a rare event occurs and multiple measurement errors occur within the extraction of a syndrome volume, creating a highly ambiguous and difficult to decode input state,

FIG. 6. The procedure for decoding with a trained agent. (a) Given a faulty syndrome volume generated by some experiment, this volume is concatenated with an empty action history to produce a suitable input state for the decoding agent. (b) The decoding agent takes in the combined faulty syndrome volume and action history and chooses an action. This action is simultaneously (c) added to a list of accumulated corrections and (d) used to update only the action history component of the input state to the agent. This updated input state is then given to the agent and procedures (b-d) continue until the agent (e) repeats an action or chooses the identity. (f) At this stage the corrections could be applied to the lattice state, although in practice they would be tracked through the entire computation.

the agent may choose to only partially decode before requesting a new syndrome volume, in which hopefully less measurement errors will occur and the underlying error configuration may be less ambiguous. As such, the decoding agents obtained via this framework may have access to truly novel decoding strategies, that are not currently available to alternative decoding algorithms.

It is also useful to note that in practice it is possible to speed-up learning by utilizing various improvements. First of all, when choosing exploratory actions, we can restrict the action space to a reduced set of potentially valid corrections. In particular, the agent need only consider actions on vertices either involved in a violated stabilizer or adjacent to vertices which have already been acted on. This restriction effectively increases the probability of the agent discovering useful actions, and therefore the effectiveness of any exploration phase. Second, when generating new syndrome volumes, we discard those cases in which the syndrome is trivial. In these cases, the agent does not need to act and hence no useful experience tuple would be generated. This allows any experience memory required by the learning algorithm to consist of only useful memories.

Finally, although we have now thoroughly addressed a framework to which reinforcement learning algorithms could be applied to *train* an agent, we have not yet explicitly discussed the implementation of our *q*-function based agent and how it may be used to decode within an experimental setting. As illustrated in Fig. 6, the latter problem is straightforwardly addressed. Given a faulty syndrome volume from an experiment, we initialize an empty action history list and combine this with the faulty syndrome volume as an initial input to the decoding agent. The trained agent can then be asked directly for an initial correction, which is added to a list of accumulated corrections. Simultaneously *only* the action history element of the previous input state is updated. This updated input state, containing the original syndrome volume and the subsequently performed corrections, is then given again

to the agent as input. This process is iterated until the agent either repeats a correction or chooses the identity, which should be taken as a signal to perform another round of syndrome measurements.

VI. A DEEPQ DECODING AGENT

As discussed in the previous section, many different reinforcement learning algorithms could now be applied within the framework presented here. However, in order to provide a concrete example and proof-of-principle, we will specialize to deepQ learning, which has been previously utilized to obtain agents capable of human-level control in domains such as Atari. As mentioned briefly in Section IV, a variety of now standard tricks are required to get deepQ learning to work in practice, and we will not present these details here, referring the reader to the relevant references [38–41] or associated code repository [54]. However, there are various details concerning the construction of the deepQ agent which may be useful for applying alternative deep reinforcement learning algorithms within this framework, and as such we will present these details in this section.

In particular, in Section V we described how at the beginning of any time step t the agent is supplied with the state $S_t = (S_{E,t}, h_t)$, where $S_{E,t}$ needs to be constructed from a faulty syndrome volume, and h_t is a representation of the agents action history. In deepQ learning, and in many other deep reinforcement learning algorithms, this state S_t needs to be provided as the input to a deep neural network. For example, in the case of deepQ learning, the state S_t is the input to the deepQ network parametrizing the q-function from which the agent is partially deriving its policy. As such, utilizing an appropriate encoding of S_t, which allows for the use of appropriate neural networks, is important.

In Fig. 7 we have illustrated the encoding of S_t which was used to facilitate the use of deep convolutional neural
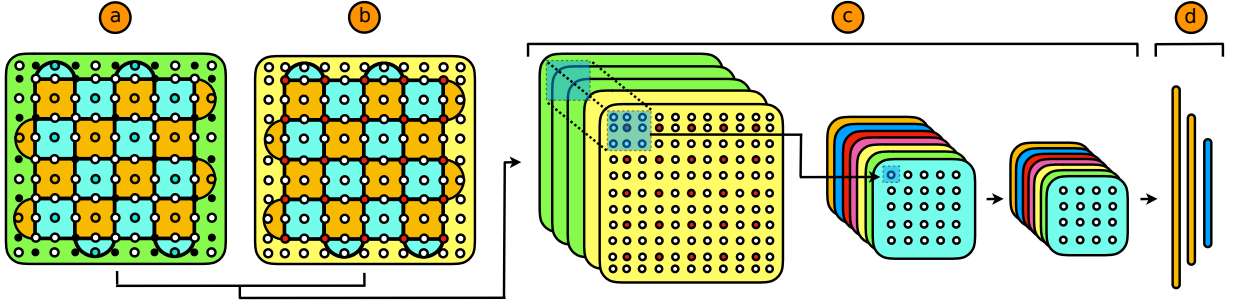
FIG. 7. Details of a deepQ decoding agent for a $d \times d$ code lattice. (a) A single faulty syndrome is embedded into a $(2d + 1) \times (2d + 1)$ binary matrix, where by default all entries are initially set to 0. Entries indicate by orange (blue) circles are then used to indicate violated Z (X) stabilizers on the corresponding plaquette. Entries indicated by black circles are set to $+1$ as a way of differentiating different types of stabilizers. (b) Using the same embedding, the action history for Z (X) flips can be encoded by indicating a previous flip on a specific vertex qubit via the entry corresponding to that vertex. (c) By stacking the syndrome and action history slices, the total state S_t can be constructed in a form suitable for input to a convolutional neural network. (d) To complete the deepQ network, we stack a feed forward neural network on top of the convolutional layers. In particular, the final layer of this network has $|\mathcal{A}|$ activations, each of which encodes $q(S_t, a)$ for an action a.

networks to parametrize the q-function. In particular, as shown in Fig. 7 (a) and (b), we can embed a $d \times d$ code lattice into a $(2d + 1) \times (2d + 1)$ binary matrix, where each entry corresponds to either a plaquette, a vertex, or an edge of the lattice. As shown in Fig. 7 (a), we can then use a single such binary matrix to encode each of the faulty syndromes, by using the entries corresponding to plaquettes to indicate violated stabilizers, and the remaining entries to differentiate both blue and orange, and bulk and boundary plaquettes. Similarly, as illustrated in Fig. 7 (b) we can use two such binary matrices to encode the action history, by using one matrix to indicate the physical data qubits on which X flips have already been applied, and the other binary matrix to indicate the data qubits on which Z flips have already been applied. As can be seen in Fig. 7 (c), the total state S_t can then be obtained by stacking the action history slices on top of the faulty syndrome slices, effectively creating a multi-channel image suitable as input for a convolutional neural network. In particular, the strength of this encoding is that any convolutional filter, such as the one indicated in Fig. 7 (c), then isolates all relevant information, both violated stabilizers and applied actions, from some local patch of the lattice. Finally, the deepQ network we utilize is completed by stacking a feed forward neural network on top of multiple convolutional layers. In particular, the final layer of this network has $|\mathcal{A}|$ activations, each of which encodes $q(S_t, a)$ for an action a.

VII. RESULTS

As a demonstration, we utilized deepQ learning within the framework presented in Section V, to obtain deepQ decoding agents for both bit-flip and depolarizing noise, with faulty syndromes, for a $d = 5$ surface code lattice. All the code utilized to obtain these agents is supplied in the corresponding *DeepQ-Decoding* code repository

[54], and as such we will provide only an overview of the details here. In particular, for a single fixed set of hyper-parameters, we used the original deepQ algorithm [38] with annealed ϵ-greedy exploration, implemented via the keras-RL library [55] and incorporating doubleQ updates [40] and dueling networks [41]. In addition we used a custom training procedure, described in Appendix A, for sequentially iterating through increasing error rates, while simultaneously performing a hyper-parameter grid search at each error-rate iteration. For both error models we utilized a convolutional deepQ network, as illustrated in Fig. 7, consisting of three convolutional layers, followed by a single feed-forward layer before the final output layer. Specifically, if we describe a single convolutional layer with a three-tuple $[n, w, s]$, where n is the number of filters, w is the filter width and s is the stride, and a single feed forward layer via a two-tuple $[n, d]$, where n is the number of neurons and d is the output drop-out rate, then from input to output our deepQ networks had the structure,

$$[[64, 3, 2], [32, 2, 1], [32, 2, 1], [512, 0.2], [|\mathcal{A}|, 0]], \quad (10)$$

where $|\mathcal{A}|$ is the size of the action space. All other additional hyper-parameters used to obtain each decoding agent, along with histories of each training procedure, are provided in Appendix B.

We then considered both bit-flip and depolarizing noise. For both error models we considered the measurement of a single syndrome to consist of two separate error channels, as illustrated in Fig. 3. For bit-flip noise, in the first error channel - the physical error channel - a Pauli X flip was applied to each physical data qubit with probability p_{phys}. For depolarising noise, the physical error channel acted by applying to each physical data qubit, with probability p_{phys}, either a Pauli X, Y or Z flip, with equal probability. For both noise models, after the physical error channel, the true syndrome was
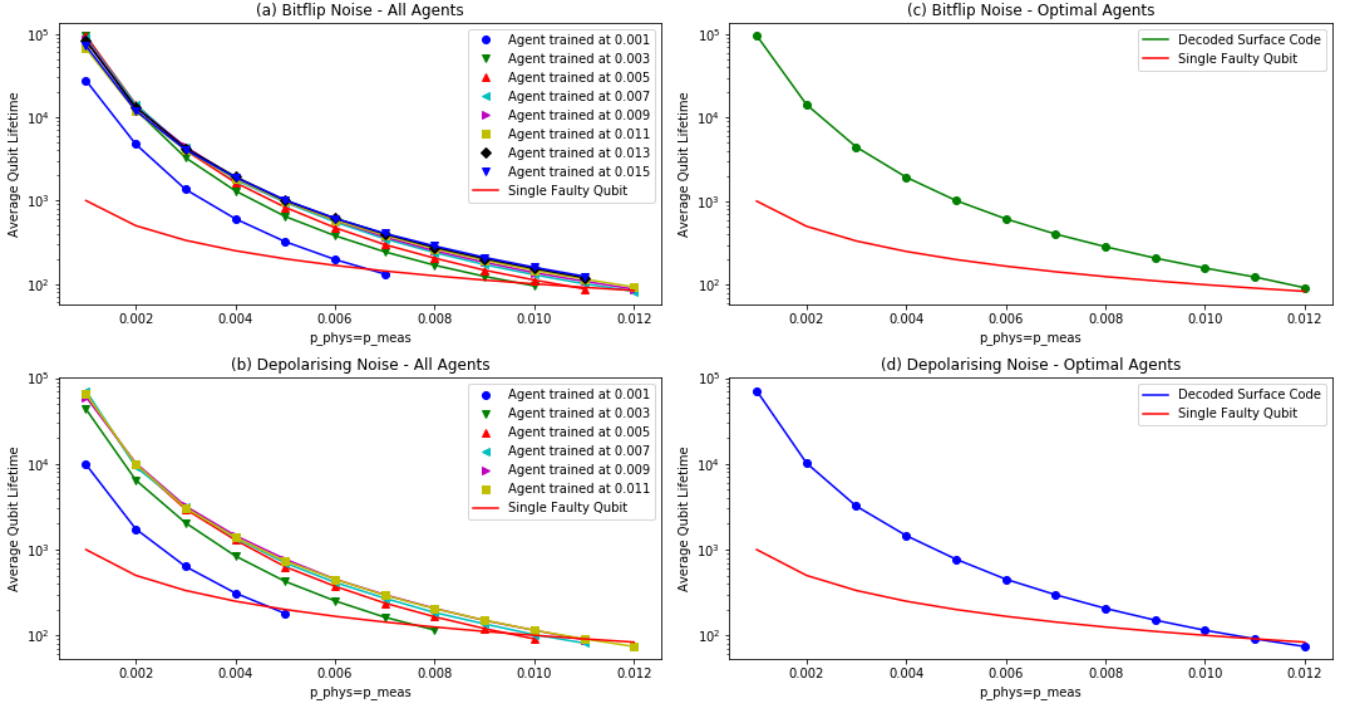
FIG. 8. (a,b) Performance of all agents obtained during the iterative training procedure. Each agent was evaluated at increasing error rates, until the average lifetime of the logical qubit actively decoded by the agent became less than the average lifetime of a single faulty qubit. (c,d) Results obtained by using the optimal decoding agent for each error rate.

calculated, after which the second measurement error channel was applied, in which the value of each stabilizer measurement was flipped with probability p_{meas}. For all simulations we set $p \equiv p_{\text{phys}} = p_{\text{meas}}$. Also, note that for bit-flip noise, as only X corrections were necessary, we had $|\mathcal{A}| = d^2 + 1$ - i.e. the agent could perform either the identity or a single qubit X flip on any individual physical data qubit. For depolarising noise, we restricted the agent to only X and Z flips (as Y errors can be corrected via both an X flip and a Z flip), such that $|\mathcal{A}| = 2d^2 + 1$.

To evaluate the performance of our trained decoders we used the procedure described in Fig. 6, where the agent selected actions via the final q-function in a purely greedy manner, with identity or repeated actions triggering new syndrome volumes. The referee decoder was used to check after every action of the agent whether or not a terminal state had been reached, and the length of a single episode was reported as the number of individual syndromes seen by the agent (i.e. the number of times the two-fold error channel was applied) before a terminal state was reached. For each error rate, the average lifetime of the actively decoded logical qubit was determined by the average episode length, over a number of episodes that guaranteed at least 10^6 syndromes were seen by the decoding agent. This average logical qubit lifetime was then compared with the average lifetime of a single faulty qubit.

The final results are then shown in Fig. 8. In particular, Fig. 8 (a, b) shows the performance of all decoding

agents obtained during the iterative training procedure (as described in Appendix A), while Fig. 8 (c, d) shows the results obtained by the using the optimal decoding agent for each error rate. For bit-flip (depolarising) noise we find that for approximately $p < 1.4 \times 10^{-2}$ $(p < 1.1 \times 10^{-2})$ there is a decoding agent for which the average lifetime of the actively decoded logical qubit is longer than the average lifetime of a single faulty qubit. These results are however by no means rigorous thresholds, and are meant as a demonstration and proof-of-principle. In this regard there are various points worth emphasising.

Firstly, both the neural network architecture and reinforcement learning algorithm used here are comparatively simple with respect to the current state-of-the-art [42–44, 56], and were chosen to allow for the execution of the required training procedure with the available computational resources. As such, it is expected that utilization of either more sophisticated neural network architectures or learning algorithms, coupled with the computational resources required for implementing the required training procedures, could allow one to obtain significantly better results. Furthermore, as discussed in ref. [37], using dedicated hardware it is expected that the forward pass time of various applicable neural network architectures (including the architecture used here) can be brought below the time-scales necessary for near-term experiments. This, coupled with the ability to straightforwardly apply the techniques here to circuit-level noise, renders decoding agents such as the ones shown here promising candidates

for near-term fault-tolerant experiments.

VIII. CONCLUSION

We have shown that the problem of decoding within the setting of fault-tolerant quantum computation can be naturally reformulated as a reinforcement learning problem. In particular, we have provided an agent-environment framework which allows for the application of diverse reinforcement learning algorithms, and provided a proof-of-principle demonstration by training deepQ decoding agents for the $d = 5$ surface code, for both bit-flip and depolarizing noise, with faulty syndrome measurements. However, this framework is both code and error model agnostic, and as such could be used to obtain decoding agents, a novel class of flexible decoding algorithms, for a wide variety of experimentally relevant settings. Additionally, the recent use of more sophisticated reinforcement learning techniques and neural network architectures to demonstrate super-human performance in complex domains such as Chess and Go [42–44], strongly indicate that the initial results presented here, via deepQ learning with small neural networks, could be successfully improved and extended, given the required computational resources for training. This, coupled with the development of dedicated special purpose hardware for the fast implementation of neural networks, suggest that decoding agents could indeed provide a practical solution for decoding in near-term fully fault-tolerant quantum computation.

As a result, there are various natural avenues for extension of this work. First and foremost it would be of interest to explore the performance of alternative decoding agents, obtained via different reinforcement learning algorithms, for a wider class of codes and error models. Simultaneously, in the perfect-measurements setting, recent effort has focused on developing techniques to allow for the scaling of neural network decoders to larger code distances [36] As such, it would be natural to explore the extent to which these developments could be utilized within the framework presented here, to facilitate obtaining decoding agents for significantly larger code distances. Finally, due to the inherent design of the procedure via which these decoding agents are obtained, in which discounted future performance is valued over immediate rewards, these agents have the potential to learn decoding strategies not available to alternative decoding algorithms. Hence, it may be possible that analysis of the strategies learned by decoding agents could aid the development of other new decoding algorithms.

ACKNOWLEDGMENTS

Appendix A: Distributed Iterative Training with Simultaneous Hyper-Parameter Optimization

In order to obtain optimal decoders for multiple error rates we implemented a custom iterated training procedure, involving a hyper-parameter grid search at each error rate, as illustrated in Fig. 9. All code for implementing this procedure on an HPC cluster can be found in the associated *DeepQ-Decoding* repository [54]. In particular, this procedure involves the following steps:

1. Choose an initial error rate p_1, and fix the values for any hyper-parameters which should remain constant throughout the entire training procedure.

2. For all hyper-parameters not yet fixed, specify a list of values to be used for training - i.e. specify a hyper-parameter grid over which one would like to search for an optimal hyper-parameter configuration.

3. Train multiple decoding agents at the initial error rate, one for each hyper-parameter configuration in the specified grid. Each agent is initialized with an empty experience memory and random initial neural network weights. The training of all agents can be done simultaneously in a distributed manner.

4. Given the results from all agents trained at p_1, sort the results and store the neural network weights and experience memory from the optimal decoding agent.

5. Increase the error rate to p_2. Once again, train multiple decoding agents, one for each hyper-parameter configuration in the specified grid. However, this time all agents are initialized with the experience memory and neural network weights from the optimal agent at p_1.

6. Iterate this procedure until a specified final error rate p_n, or until the performance of the optimal agent is worse than that of a single faulty qubit.

Appendix B: Agent Hyperparameters and Learning Curves

We implemented the distributed iterative training procedure described in Appendix A, for both bit-flip and
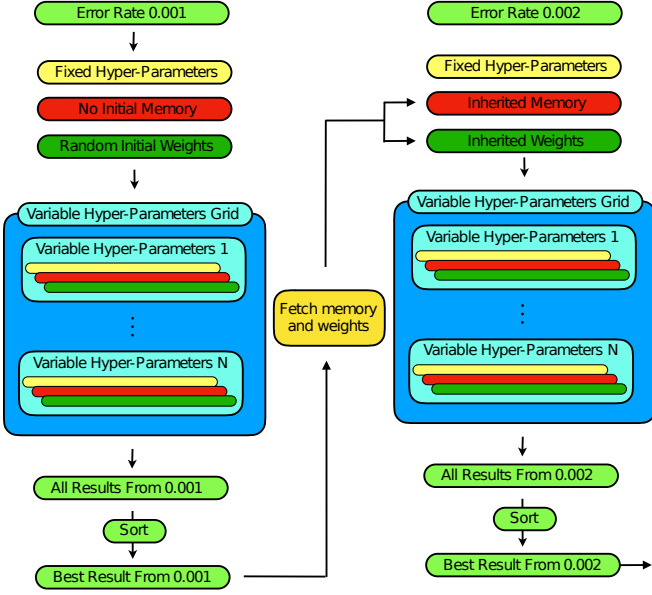
FIG. 9. Iterative training procedure, via a hyper-parameter grid search at each error rate. After training multiple decoding agents at a given error rate, one for each point in the specified hyper-parameter grid, the results are sorted. The weights and experience memory from the optimal decoding agent are then used as the initial weights and experience memory for a new round of training procedures, again one for each point in the hyper-parameter grid, but at an increased error rate.

depolarising noise (all utilized code can be found in the *DeepQ-Decoding* repository). In particular, the initial error rate was set to $p_1 = p_{\text{phys}} = p_{\text{meas}} = 1 \times 10^{-3}$, and incremented by 2×10^{-3} in each iteration. The neural network architecture was as described in Section VII. All agents were trained via the original deepQ algorithm [38], implemented via the keras-RL library [55], with annealed ϵ-greedy exploration, doubleQ updates and dueling networks [40]. Both fixed and variable hyper-parameter grids are specified in Table. I. Figs. 10 and 11 show the training history of the optimal decoding agent at each error rate, as well as the associated variable hyper-parameters.

TABLE I. Hyper-Parameters

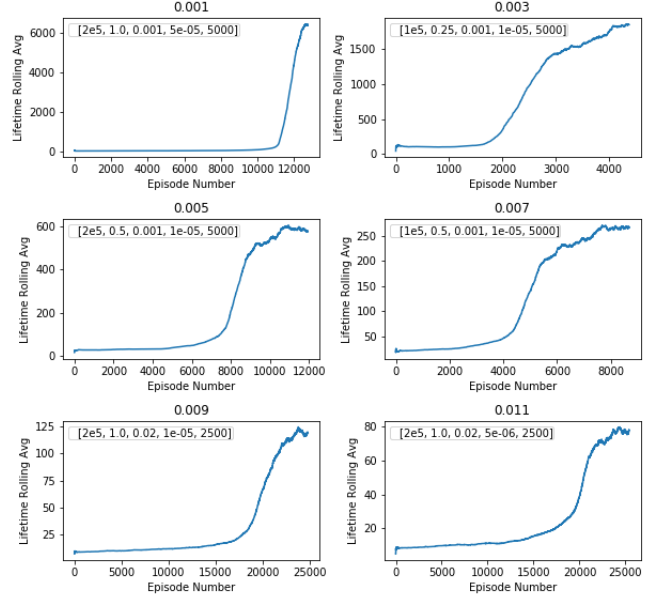| Fixed Hyper-Parameters	
Batch size	32
Rolling average length	1×10^3
Stopping patience (in episodes)	1×10^3
Maximum training steps	1×10^6
Memory buffer size	5×10^4
Syndrome volume depth	5
Discount factor γ	0.99
Variable Hyper-Parameters	
Initial ϵ	$\{1, 0.5, 0.25\}$
Final ϵ	$\{0.04, 0.02, 0.001\}$
Number of exploration steps	$\{1, 2\} \times 10^5$
Learning rate (LR)	$\{10, 5, 1, 0.5\} \times 10^{-5}$
Target network update frequency	$\{2500, 5000\}$

FIG. 10. Training histories and variable hyper-parameter values for optimal depolarising noise agents. Hyper-parameters are given as a list [Number of exploration steps, initial ϵ, final ϵ, learning rate, target network update frequency].

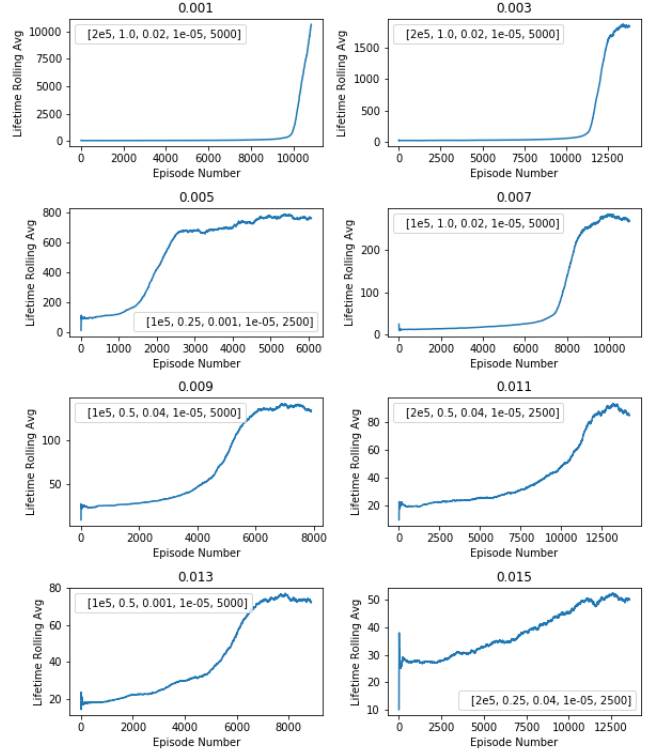
FIG. 11. Training histories and variable hyper-parameter values for optimal bitflip noise agents. Hyper-parameters are listed as per Fig. 10.

[1] B. M. Terhal, *Quantum error correcton for quantum memories,* Rev. Mod. Phys. **87**, 307 (2015).

[2] E. T. Campbell, B. M. Terhal, and C. Vuillot, *Roads towards fault-tolerant universal quantum computation,* Nature **549**, 172 (2017).

[3] A. Y. Kitaev, *Fault-tolerant quantum computation by anyons,* Ann. Phys. **303**, 2 (2003).

[4] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, *Topological quantum memory,* J. Math. Phys. **43**, 4452 (2002).

[5] J. Preskill, Topological quantum computation, (2017), (Chapter 9 of Lecture Notes on Quantum Computation).

[6] C. Nayak, S. H. Simon, A. Stern, M. Freedman, and S. D. Sarma, *Non-Abelian anyons and topological quantum computation,* Rev. Mod. Phys. **80**, 1083 (2008).

[7] J. K. Pachos, *Introduction to topological quantum computation* (Cambridge University Press, 2012).

[8] B. J. Brown, D. Loss, J. K. Pachos, C. N. Self, and J. R. Wootton, *Quantum memories at finite temperature,* Rev. Mod. Phys. **88**, 045005 (2016).

[9] M. D. Reed, L. DiCarlo, S. E. Nigg, L. Sun, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf, *Realization of three-qubit quantum error correction with superconducting circuits,* Nature **482**, 382 (2012).

[10] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffry, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland, and J. M. Martinis, *Superconducting quantum circuits at the surface code threshold for fault tolerance,* Nature **508**, 500 (2014).

[11] D. Nigg, M. Müller, E. A. Martinez, P. Schindler, M. Hennrich, T. Monz, M. A. Martin-Delgado, and R. Blatt, *Quantum computations on a topologically encoded qubit,* Science **345**, 302 (2014).

[12] A. D. Córcoles, E. Magesan, S. J. Srinivasan, A. W. Cross, M. Steffen, J. M. Gambetta, and J. M. Chow, *Demonstration of a quantum error detection code using a square lattice of four superconducing qubits,* Nat. Comms. **6**, 6979 (2015).

[13] S. M. Albrecht, A. P. Higginbotham, M. Madsen, F. Kuemmeth, T. S. Jespersen, J. Nygård, P. Krogstrup, and C. M. Marcus, *Exponential protection of zero modes in Majorana islands,* Nature **531**, 206 (2016).

[14] M. Takita, A. D. Córcoles, E. Magesan, B. Abdo, M. Brink, A. W. Cross, J. M. Chow, and J. M. Gambetta, *Demonstration of weight-four parity measurements in the surface code architecture,* Phys. Rev. Lett. **117**, 210505 (2016).

[15] N. M. Linke, M. Gutierrez, K. A. Landsman, C. Figgatt, S. Debnath, K. R. Brown, and C. Monroe, *Fault-tolerant quantum error detection,* Sci. Adv. **3**, e1701074 (2017).

[16] A. G. Fowler and C. Gidney, *Low overhead quantum computation using lattice surgery,* arXiv preprint arXiv:1808.06709 (2018).

[17] D. Litinski, *A game of surface codes: Large-scale quantum computing with lattice surgery,* arXiv preprint arXiv:1808.02892 (2018).

[18] A. G. Fowler, *Minimum weight perfect matching of fault-tolerant topological quantum error correction in average o(1) parallel time,* arXiv preprint arXiv:1307.1740 (2013).

[19] G. Duclos-Cianci and D. Poulin, *Fast decoders for topological quantum codes,* Physical review letters **104**, 050504 (2010).

[20] M. Herold, E. T. Campbell, J. Eisert, and M. J. Kastoryano, *Cellular-automaton decoders for topological quantum memories,* npj Quantum Information **1**, 15010 (2015).

[21] M. Herold, M. J. Kastoryano, E. T. Campbell, and J. Eisert, *Cellular automaton decoders of topological quantum memories in the fault tolerant setting,* New Journal of Physics **19**, 063012 (2017).

[22] A. Kubica and J. Preskill, *Cellular-automaton decoders with provable thresholds for topological codes,* arXiv preprint arXiv:1809.10145 (2018).

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

[24] G. Carleo, Y. Nomura, and M. Imada, *Constructing exact representations of quantum many-body systems with deep neural networks,* arXiv preprint arXiv:1802.09558 (2018).

[25] X. Gao and L.-M. Duan, *Efficient representation of quantum many-body states with deep neural networks,* Nature communications **8**, 662 (2017).

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

[27] P. Huembeli, A. Dauphin, and P. Wittek, *Identifying quantum phase transitions with adversarial neural networks,* Phys. Rev. B **97**, 134109 (2018).

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

[29] A. A. Melnikov, H. P. Nautrup, M. Krenn, V. Dunjko, M. Tiersch, A. Zeilinger, and H. J. Briegel, *Active learning machine learns to create new quantum experiments,* Proceedings of the National Academy of Sciences , 201714936 (2018).

[30] G. Torlai and R. G. Melko, *Neural decoder for topological codes,* Physical review letters **119**, 030501 (2017).

[31] S. Varsamopoulos, B. Criger, and K. Bertels, *Decoding small surface codes with feedforward neural networks,* Quantum Science and Technology **3**, 015004 (2017).

[32] S. Krastanov and L. Jiang, *Deep neural network probabilistic decoder for stabilizer codes,* Scientific reports **7**, 11003 (2017).

[33] N. P. Breuckmann and X. Ni, *Scalable neural network decoders for higher dimensional quantum codes,* Quantum **2**, 68 (2018).

[34] P. Baireuther, T. E. O'Brien, B. Tarasinski, and C. W. Beenakker, *Machine-learning-assisted correction of correlated qubit errors in a topological code,* Quantum **2**, 48 (2018).

[35] P. Baireuther, M. Caio, B. Criger, C. Beenakker, and T. O'Brien, *Neural network decoder for topological color codes with circuit level noise,* arXiv preprint arXiv:1804.02926 (2018).

[36] X. Ni, *Neural network decoders for large-distance 2d toric codes,* arXiv preprint arXiv:1809.06640 (2018).

[37] C. Chamberland and P. Ronagh, *Deep neural decoders for near term fault-tolerant experiments,* arXiv preprint arXiv:1802.06441 (2018).

[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K.

Fidjeland, G. Ostrovski, *et al.*, *Human-level control through deep reinforcement learning,* Nature **518**, 529 (2015).

[39] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, *Prioritized experience replay,* arXiv preprint arXiv:1511.05952 (2015).

[40] H. Van Hasselt, A. Guez, and D. Silver, in *AAAI*, Vol. 2 (Phoenix, AZ, 2016) p. 5.

[41] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, *Dueling network architectures for deep reinforcement learning,* arXiv preprint arXiv:1511.06581 (2015).

[42] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, *Mastering chess and shogi by self-play with a general reinforcement learning algorithm,* arXiv preprint arXiv:1712.01815 (2017).

[43] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, *et al.*, *Mastering the game of go with deep neural networks and tree search,* Nature **529**, 484 EP (2016), article.

[44] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, *Mastering the game of go without human knowledge,* Nature **550**, 354 (2017).

[45] D. Gottesman, *Stabilizer Codes and Quantum Error Correction*, Ph.D. thesis, California Institute of Technology (1997).

[46] Y. Tomita and K. M. Svore, *Low-distance surface codes under realistic quantum noise,* Physical Review A **90**, 062320 (2014).

[47] A. M. Stephens, *Fault-tolerant thresholds for quantum error correction with the surface code,* Physical Review A **89**, 022321 (2014).

[48] G. Nebe, E. M. Rains, and N. J. Sloane, *The invariants of the clifford groups,* Designs, Codes and Cryptography **24**, 99 (2001).

[49] S. Bravyi and A. Kitaev, *Universal quantum computation with ideal clifford gates and noisy ancillas,* Physical Review A **71**, 022316 (2005).

[50] B. J. Brown, K. Laubscher, M. S. Kesselring, and J. R. Wootton, *Poking holes and cutting corners to achieve Clifford gates with the surface code,* Phys. Rev. X **7**, 021029 (2017).

[51] C. Horsman, A. G. Fowler, S. Devitt, and R. V. Meter, *Surface code quantum computing by lattice surgery,* New J. Phys. **14**, 123011 (2012).

[52] D. Litinski and F. von Oppen, *Lattice surgery with a twist: Simplifying clifford gates of surface codes,* Quantum **2**, 62 (2018).

[53] R. Sutton, A. Barto, and F. Bach, *Reinforcement Learning: An Introduction*, Adaptive Computation and Machine Learning (MIT Press, 2018).

[54] R. Sweke, M. Kesselring, E. P. Van Nieuwenburg, and J. Eisert, Deepq-decoding, `https://github.com/R-Sweke/DeepQ-Decoding` (2018).

[55] M. Plappert, keras-rl, `https://github.com/keras-rl/keras-rl` (2016).

[56] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, in *International conference on machine learning* (2016) pp. 1928–1937.