Reinforcement Learning Decoders for Fault-Tolerant Quantum Computation

Ryan Sweke,[1] Markus S. Kesselring,[1] Evert P. L. van Nieuwenburg,[2] and Jens Eisert[1, 3]

[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*
[3]*Department of Mathematics and Computer Science, Freie Universität Berlin, 14195 Berlin*
(Dated: October 16, 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.

I. INTRODUCTION

In order to implement large scale quantum computations 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. The known 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 is 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]. In particular, 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 fact, current road maps towards the realization of robust quantum computing have identified surface code based approaches as the most feasible methodology for achieving this goal [18].

However, the known realistic topological quantum error correcting codes, including the surface code, are not self-correcting, and are therefore not robust to natural thermal noise. For this reason one has to actively diagnose and correct for errors, and as such, 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 have occurred on the physical qubits), and provide as output a suggestion of corrections for any errors that may have occurred during the computation. In practice, these decoding algorithms have to be extremely fast - in particular, one has to be able to decode faster than the rate at which errors occur. As such, the development of decoding algorithms constitutes a serious bottleneck in the realization of fault-tolerant quantum computers and are key to gaining an understanding of quantum computing in realistic regimes.

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. For this reason, while the setting of perfect syndrome measurements provides a paradigmatic test-bed for the development of decoding algorithms, any decoding algorithm which aims to be experimentally useful must necessarily 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.

As of yet, it is precisely the development of decoders applicable to the fault-tolerant setting that constitutes a particular challenge. However, due to the importance of decoding algorithms for fault-tolerant quantum computation, several 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 [19], although alternative approaches based on techniques such as the renormalization group [20] and locally operating cellular automata [21–23] have also been put forward. These algorithms solve the problem in principle, but may well be too slow in realistic settings.

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 [24–26], the identification of phase transitions [27–31], and the autonomous design of novel experimental set-ups [32, 33] - and in an attempt to tackle the issue of fast decoding various neural-network based decoders have also been proposed [34–41]. In particular, previously proposed neural network decoders promise extremely fast decoding times [37], flexibility with respect to the underlying code and noise model [37–40] and the potential to scale to large code distances [40, 41]. However, despite this diversity of proposed decoding algorithms, and the clear potential of machine learning based approaches, there is as of yet no algorithm or technique which clearly satisfies all the required experimental criteria listed above. As such, there remains room for improvement and new techniques, particularly within the fault-tolerant setting.

Simultaneously, the last few years have also seen impressive advances in the development of deep reinforcement learning al-

gorithms, which have allowed for the training of neural network based agents capable of obtaining super-human performance in challenging domains such as Atari [42–45], Chess [46] and Go [47, 48]. 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. The decoding problem within the context of fault-tolerant quantum computation is precisely such a problem. Given the significant challenges involved in the development of fast decoders for the fully fault-tolerant setting and the clear parallels between decoding and the domains in which reinforcement learning techniques have excelled, it is natural to ask both the extent to which these techniques can be applied to the decoding problem and the advantages that such an approach would offer over alternative methods.

In this work we introduce a novel reinforcement learning based framework for obtaining a new class of decoding algorithms applicable to the setting of fully fault-tolerant quantum computation. Importantly, we discuss the advantages of such an approach over the previously proposed neural network decoders, and argue that the framework presented here both lays the foundations and provides a toolbox for obtaining decoders satisfying all the required criteria. In particular, within the newly established framework to be presented here, 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 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 re-framing 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
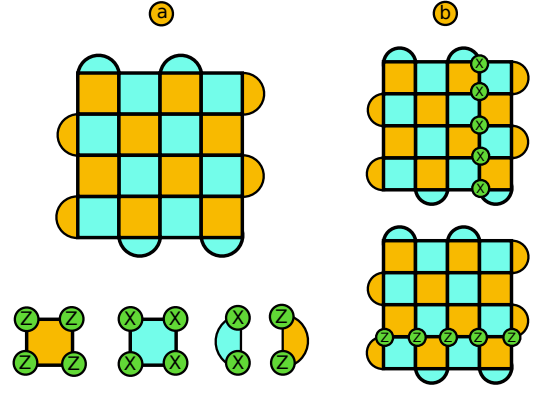

FIG. 1. An overview of the 5×5 surface code. (a) We consider square lattices, with a physical data qubit on each vertex. The colored plaquettes indicate stabilizer operators as defined in Eq. (1). (b) Logical X_L and Z_L operators for the surface code are given by continuous strings of single qubit X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

details [1, 17, 49].

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 colored 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 fictitious Hamiltonian $H = -\sum_p S_p$ from which the surface code $\mathcal{H}_{\text{sc}} \subset \mathcal{H}$ is 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 us examine the consequences of a single qubit Pauli flip on a physical data qubit. If we assume that the initial state vector $|\psi\rangle \in \mathcal{H}_{\text{sc}}$ is an element of the code space, then the subsequent state vector $|\psi'\rangle \notin \mathcal{H}_{\text{sc}}$ 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 analyzing 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.

In the terminology of stabilizer codes and quantum error correction, such a surface code on a $d \times d$ lattice is a $[[d^2, 1, d]]$ code [49]. This means that d^2 physical qubits data qubits are required to encode a single logical qubit, with code distance d. Specifically, the *distance of a stabilizer code* is the weight of the minimal-weight non-trivial logical operator - i.e. the minimal weight Pauli operator that both preserves the code subspace and acts non-trivially within this code subspace. In particular, this quantity characterizes the error correcting capabilities of a given code: if a logical qubit is affected by the action of a Pauli operator, the weight of which is less than half the code distance, then one can successfully recover from the error by applying the minimal weight Pauli operator which returns the altered state to the code subspace.

III. THE DECODING PROBLEM

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

Problem 1 (Decoding problem) *Assume that at $t = 0$ one is given a state vector $|\psi\rangle = \alpha|0_L\rangle + \beta|1_L\rangle \in \mathcal{H}_{\mathrm{sc}}$. 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 us 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
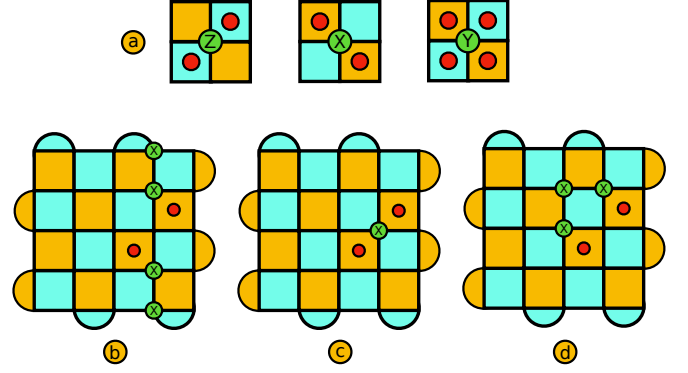

FIG. 2. (a) Single qubit Pauli flips violate surrounding stabilizers. (b-d) Strings of Pauli flips only violate stabilizers at the endpoint of the string. Multiple error configurations can give rise to the same syndrome. They can differ by stabilizers, as for example in (c) and (d), or by logical operators,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

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. The mathematical structure underlying the decoding problem is that of homology, which provides a concise representation of the relationship between errors and syndromes [50]. In this language, a decoder determines the approximate relative likelihood of different homology classes – equivalence classes of error patterns – of given error configurations that are captured by the syndrome.

In addition to the inherent difficulty resulting from syndrome ambiguity, in experimental settings the process of extracting the syndrome is its itself subject to noise. That is to say, one must reasonably assume that the syndrome itself may be faulty [51, 52]. In practice, each stabilizer may be 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.

The essential aspect of the additional difficulty from faulty syndrome measurements can be phenomenologically modeled by imagining each time step as consisting of two distinct error processes [52], 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 realistic scenario, lacking only the propagation of errors
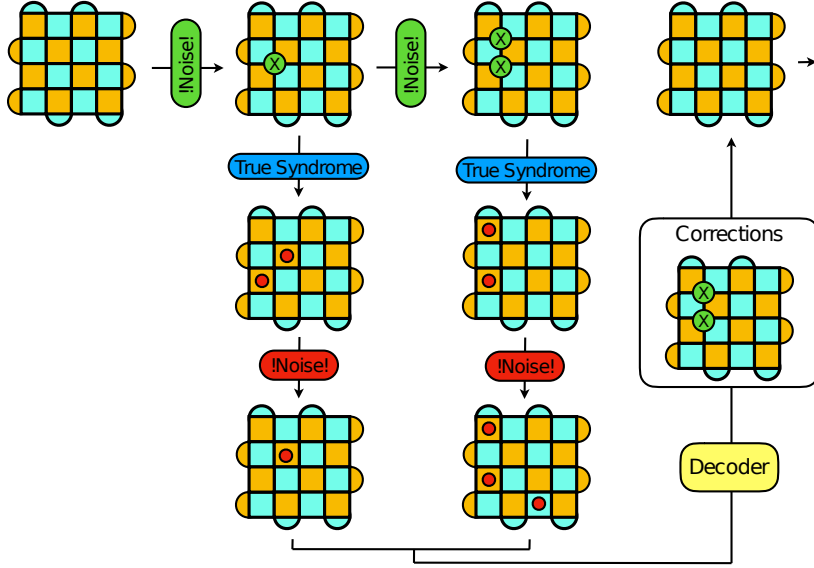
FIG. 3. A typical decoding cycle is illustrated for the simplified faulty measurements scenario in which one imagines each time step consisting of an initial physical error process generating errors on the data qubits, followed by a second measurement error process which corrupts the true syndrome. The decoding algorithm then has access to a sequence of potentially faulty syndromes.

during syndrome extraction, single syndrome measurements are hence no longer reliable. Decoding in the fault-tolerant setting therefore typically requires providing a set of sequential syndrome measurements, and it should be clear at this point that by including the requirement of fault tolerance, the nature of the decoding problem changes substantially.

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 [53, 54]. In contemporary proposals for surface code based quantum computing [16, 17], protocols are known for implementing Clifford gates either by tracking, via code deformation [55] or via lattice surgery [56, 57]. 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 [54]. 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.

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 prob-

lem in Section V. Again, we will keep the discussion brief and refer to Ref. [58] for a more 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 $\mathcal{S}$ 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 T_{t+1}, 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 classical *finite Markov decision process (FMDP)* governed by the transition probabilities

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

FIG. 4. An illustration of the signals passed between an agent and the environment through the duration of a sequential turn based episode.

the *value* of a state s under policy π as,

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

$\forall S_t \in \mathcal{S}$. 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.

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

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$
$$= \mathbb{E}_\pi\Big[\sum_{k=0}^\infty \gamma^k R_{t+k+1}\Big| S_t = s, A_t = a\Big]. \quad (4)$$

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

$$q_*(s,a) = \mathbb{E}\big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')\big| S_t = s, A_t = a\big]. \quad (5)$$

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

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. (5), for which $q_*(s,a)$ is a stationary solution [58].

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 [42–44]. In particular, in deepQ learning we parameterize q by a neural network, and use Eq. (5) 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 form

$$[S_t, A_t, R_{t+1}, S_{t+1}, T_{t+1}]. \quad (6)$$

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

$$C = y_\text{pred} - y_\text{true} \quad (7)$$
$$= q(S_t, A_t) - \big[R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a')\big], \quad (8)$$

which, by comparison with Eq. (5), will be minimized by the optimal policy q_*. 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 [42–45], or to the associated code repository [59], 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. For clarity, we will utilize the surface code to present all the required elements, however

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 a Pauli flip which is not already in the action history (b), then the procedure illustrated in box (c) occurs, if the agent requests a new syndrome or repeats an action (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.

the framework described here could be applied to any stabilizer quantum error correcting code.

In order to present such a reformulation it is necessary to define the state space $\mathcal{S}$, action space $\mathcal{A}$ and terminal state subset $\mathcal{S}_{\text{terminal}} \subset \mathcal{S}$, as well as the stochastic process via which the environment generates the tuple $[S_{t+1}, R_{t+1}, T_{t+1}]$, when acted upon with action A_t. As discussed in Section III, within the context of fault-tolerant quantum computation, the goal of the decoding problem is to continuously suppress errors on a logical qubit, to the extent that future logical operations involving this logical qubit can succeed with high probability. As such, the fundamental idea is to define all the required elements in such a way that allows for decoding agents to learn to continuously correct errors by performing single qubit Pauli flips on the underlying physical qubits of a given code, remaining "alive" as long as future logical operations can succeed with high probability, and being rewarded whenever all errors have been successfully corrected. In particular, given some initial logical state $|\psi_0\rangle \in \mathcal{H}_{\text{sc}}$, the goal of the agent is to suppress errors for as long as possible, such that future logical operations can succeed with high probability.

To provide a framework for achieving this, we consider environments consisting of the following three elements: A *hidden state*, an *error model* and a *referee decoder*. At the beginning of any time step t, the *hidden state* of the environment $S_{\text{hidden},t}$, will be a list of all the single qubit Pauli flips which have been applied to physical data qubits through the course of the episode, either via errors or as corrections. Given both the hidden state $S_{\text{hidden},t}$ and the initial logical state $|\psi_0\rangle$, the current underlying state $|\psi_t\rangle$ of all the physical data qubits

could be obtained by applying all the Pauli operations listed in $S_{\text{hidden},t}$ to $|\psi_0\rangle$. Note that because we are particularly interested in only the difference between the current state $|\psi_t\rangle$ and the initial logical state $|\psi_0\rangle$, and because we consider only Pauli noise and corrections, we are able to utilise the simple and efficient hidden state description provided here.

In addition to the hidden state, the *error model* of the environment defines the physical process via which *faulty syndrome volumes* - i.e. a list of violated stabilizers from multiple sequential syndrome measurements - are generated from the hidden state of the environment. In Section VII we utilize the two-stage error model involving separated physical and measurement errors, as described in Fig. III, however we wish to emphasise that in principle a circuit model for syndrome extraction could also be simulated.

Finally, a *referee decoder* is included to act as a proxy for future logical operations, as such providing a mechanism for the determination of terminal states. 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, and which may fail by inadvertently suggesting corrections which perform a logical operation. In particular, at any stage, the ability of the referee decoder to decode a *perfect* syndrome generated from the current hidden state of the environment will be used as a proxy for the success of future logical operations, and an indicator of whether or not the current state is a terminal state.

Given these fundamental elements of the environment we define the action space $\mathcal{A}$ to consist of all Pauli X and Z flips on single physical data qubits, along with a special *Request New*

Syndrome action. Note that Pauli Y flips can be implemented via X and Z flips, and that in practice we imagine all single qubit corrections suggested between successive syndrome measurements would be accumulated, either to be tracked through the computation, or applied simultaneously, as illustrated in Fig. 6. In addition, we define the state space $\mathcal{S}$ to consist of all possible states of the form $S_t = \{S_{\mathrm{sv},t}, h_t\}$, where $S_{\mathrm{sv},t}$ is a faulty syndrome volume and h_t is the *action history*, a list of all the actions performed by the agent since the syndrome volume $S_{\mathrm{sv},t}$ was generated.

Finally, we have all the ingredients necessary to describe the rules, illustrated in Fig. 5, via which the environment generates the tuple $[S_{t+1} = \{S_{\mathrm{sv},t+1}, h_{t+1}\}, R_{t+1}, T_{t+1}]$ when acted upon with action A_t. In particular, given an environment with hidden state $S_{\mathrm{hidden},t}$, depending on the action A_t one of two procedures, illustrated in Figs. 5 (c) and (e) respectively, will be utilized to generate the tuple $[S_{t+1}, R_{t+1}, T_{t+1}]$.

A. Non-Repeated Pauli Flips

If, as shown in Fig. 5 (c), the action A_t chosen by the agent is any Pauli flip that is not already an element of the action history list h_t, then the environment responds as follows:

1. The hidden state of the environment $S_{\mathrm{hidden},t+1}$ is obtained by appending A_t to $S_{\mathrm{hidden},t}$. In essence, the agent's chosen correction is applied to the underlying state $|\psi_t\rangle$, yielding state $|\psi_{t+1}\rangle$.

2. We set $S_{\mathrm{sv},t+1} = S_{\mathrm{sv},t}$, therefore providing the agent an opportunity to provide more corrections in response to the current syndrome volume. In addition, the updated action history h_{t+1} is obtained by appending to A_t to h_t. The output state is then $S_{t+1} = \{S_{\mathrm{sv},t+1}, h_{t+1}\}$.

3. If $|\psi_{t+1}\rangle$ is equal to the initial logical state $|\psi_0\rangle$ - i.e. if all errors have been corrected at this stage without implementing a logical operation - then $R_{t+1} = 1$, otherwise $R_{t+1} = 0$. Technically, this can be determined by checking both that no stabilizers are violated by $|\psi_{t+1}\rangle$, and that $|\psi_{t+1}\rangle$ belongs to the same homology class as $|\psi_0\rangle$.

4. Finally, the referee decoder is given a perfect syndrome generated from $|\psi_{t+1}\rangle$. If the referee decoder can successfully decode this syndrome, then S_{t+1} is not a terminal state and we set $T_{t+1} = 0$. If the referee decoder incorrectly decodes the given syndrome, then $T_{t+1} = 0$ and the episode is over.

B. Request New Syndrome or Repeated Pauli Flip

If on the other hand the agent requests a new syndrome, or chooses an action $A_t \in h_t$ - i.e. an action it has already chosen since first seeing the syndrome volume $S_{\mathrm{sv},t}$ - then the environment responds via the following procedure, as illustrated in Fig. 5 (e).

1. If the action A_t is a Pauli flip already in h_t, i.e. if for some reason the agent is choosing to repeat an action, then this chosen correction is applied to the underlying state by appending A_t to $S_{\mathrm{hidden},t}$. We denote this new hidden state as $S'_{\mathrm{hidden},t+1}$. In this case, the reward R_{t+1} and terminal state indicator T_{t+1} are determined from $S'_{\mathrm{hidden},t+1}$ by the process described in Section V A for non-repeated Pauli flips.

2. If action A_t is the *request new syndrome* action, then the hidden state is not updated, i.e. $S'_{\mathrm{hidden},t+1} = S_{\mathrm{hidden},t}$, and as such $R_{t+1} = R_t$ and $T_{t+1} = T_t$.

3. Now, a new syndrome volume $S_{\mathrm{sv},t+1}$ is generated, via the given error model, from the underlying hidden state $S'_{\mathrm{hidden},t+1}$. In the process errors may occur on physical data qubits, and the final hidden state $S_{\mathrm{hidden},t+1}$ is obtained by applying these errors to the intermediate hidden state $S'_{\mathrm{hidden},t+1}$.

4. Finally, as a new syndrome volume has just been generated, the action history h_{t+1} is reset to an empty list and the total state $S_{t+1} = \{S_{\mathrm{sv},t+1}, h_{t+1}\}$ is returned.

Given all the above elements, it is now possible to summarize a complete episode. In particular, every episode starts with a resetting of the environment. Specifically, as shown in Fig. 5 (a), given an initial logical state $|\psi_0\rangle \in \mathcal{H}_{\mathrm{sc}}$, represented by a hidden state $S_{\mathrm{hidden},0}$ which is just an empty list, the *request new syndrome* action is applied to the environment to obtain the initial tuple $[S_1 = \{S_{\mathrm{sv},1}, h_1\}, R_1, T_1]$. Now, in any step t, given $[S_t, R_t, T_1]$ the agent needs to decide on an action A_t, which is applied to the environment. Using the rules described above the environment then generates the tuple $[S_{t+1}, R_t, T_1]$, and the process continues until $T_{t+1} = 1$, i.e. until $S_{t+1} \in \mathcal{S}_{\mathrm{terminal}}$. From the construction above one can see that in order to maximise long term discounted cumulative reward, a decoding agent needs to learn how to choose actions that suppress errors, preserving as closely as possible the initial logical state $|\psi_0\rangle$. If however, the agent chooses incorrect actions, then errors will accumulate and the referee decoder will no longer be able to correctly decode, indicating the probable failure of subsequent logical gates involving the logical qubit.

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, as mentioned above, the agent will only accumulate reward provided it can learn to consistently correct any errors that might

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 requests a new syndrome. (f) At this stage the corrections could be applied to the lattice state, although in practice they would be tracked through the entire computation.

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, in the rare event of multiple measurement errors occurring within the extraction of a syndrome volume, creating a highly ambiguous and difficult to decode input state, 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 various alternative design choices are possible, and that 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, the structure of the reward mechanism presented here is only one of many possible choices. In particular, while this natural reward structure facilitated stable training and the obtaining of decoding agents whose performance is shown in Section VII, it is an interesting and open question to explore the effectiveness of alternative reward mechanisms.

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 requests a new syndrome volume.

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 [42]. 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 [42–45] or associated code repository [59]. 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 $S_t = \{S_{\text{sv},t}, h_t\}$, where $S_{\text{sv},t}$ is a faulty syndrome volume, given as a list of violated stabilizers from successive syndrome measurements, and h_t is the action history list. In deepQ learning, and in many other deep reinforcement learning algorithms,
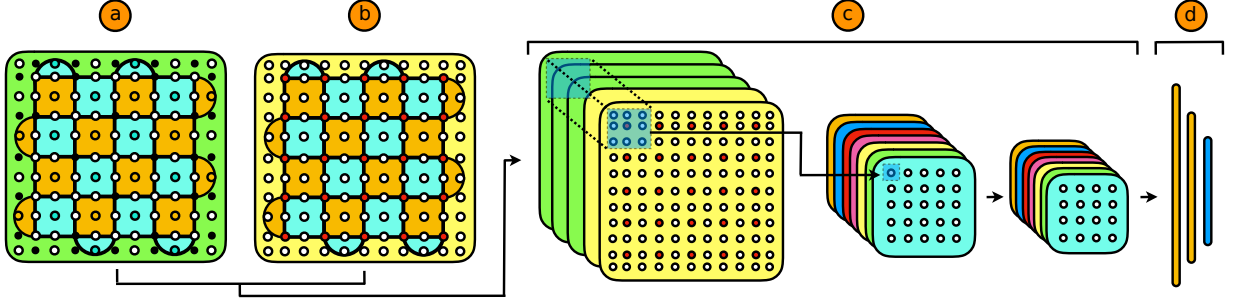
FIG. 7. Details of a deepQ decoding agent for a $d \times d$ code lattice. (a) A single faulty syndrome is embedded into a $(2d+1) \times (2d+1)$ binary matrix, where by default all entries are initially set to 0. Entries indicate by orange (blue) circles are then used to indicate violated Z (X) stabilizers 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.

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 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 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 from some local patch of the lattice - in particular, the violated stabilizers and their type, as well as the previously applied actions. 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 have 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 used to obtain these agents is supplied in the corresponding *DeepQ-Decoding* code repository [59], and as such we will provide only an overview of the details here. In particular, for a single fixed set of hyper-parameters, we have exploited the original deepQ algorithm [42] with annealed ϵ-greedy exploration, implemented via the keras-rl library [60] and incorporating doubleQ updates [44] and a dueling network architecture [45]. In addition, we have made use of 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 base structure,

$$[[64, 3, 2], \ [32, 2, 1], \ [32, 2, 1], \ [512, 0.2], \ [|\mathcal{A}|, 0]], \quad (9)$$

where $|\mathcal{A}|$ is the size of the action space. Additional minor modifications as required for dueling methodologies were also included [45]. All other additional hyper-parameters used to obtain each decoding agent, along with histories of each training procedure, are provided in Appendix B.

We considered both bit-flip and depolarizing noise models. For both of these 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
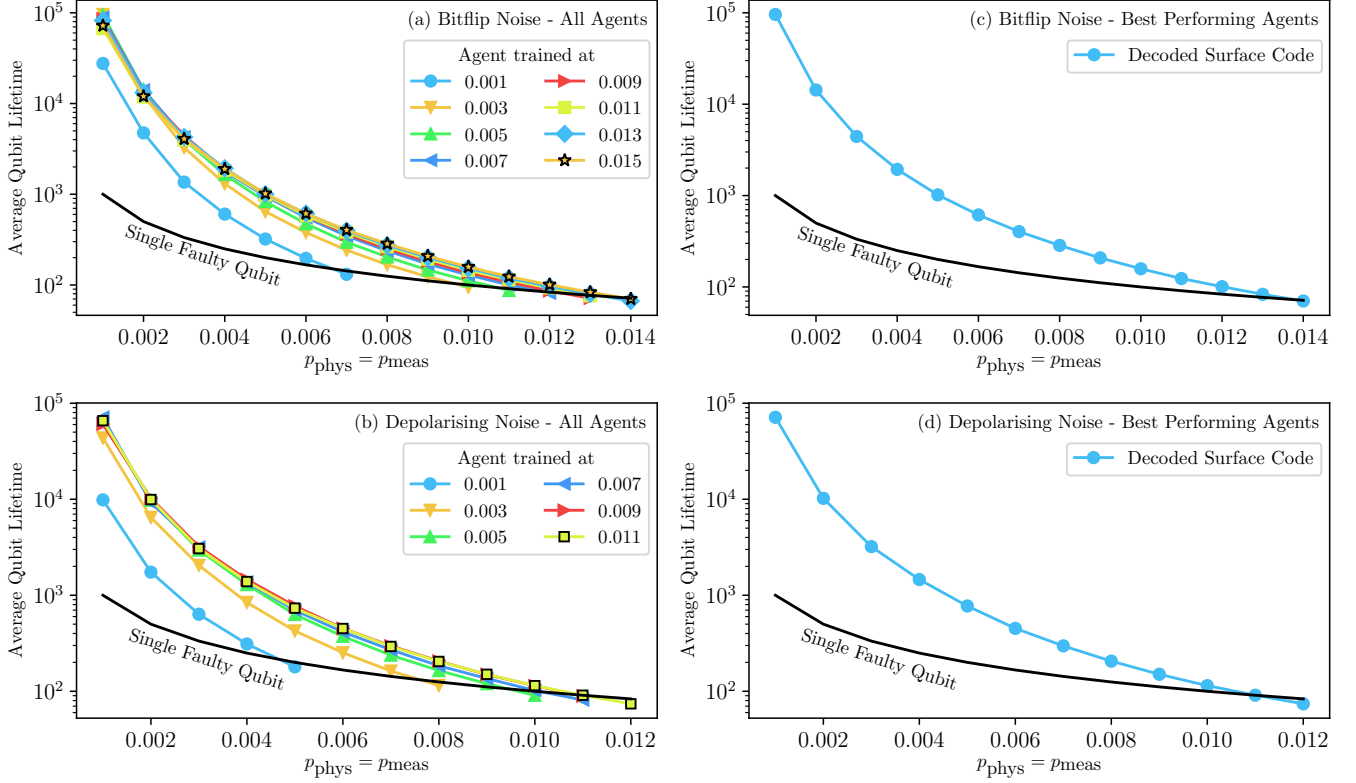
FIG. 8. (a,b) Performance of all agents obtained during the iterative training procedure. Each agent was evaluated at increasing error rates, until the average lifetime of the logical qubit actively decoded by the agent became less than the average lifetime of a single faulty qubit. (c,d) Results obtained by using the best performing decoding agent for each error rate.

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 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 $p := p_{\text{phys}} = p_{\text{meas}}$, and used a syndrome volume depth of 5. 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 *request new syndrome* action 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 $|\mathcal{A}| = 2d^2 + 1$ rather than $|\mathcal{A}| = 3d^2 + 1$.

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 repeated actions or the request of a new syndrome triggering new syndrome volumes. As referee decoders, we utilized fast feed-forward neural network based homology class predictors, trained in a supervised manner, as per Refs. [34, 35]. All utilized referee decoders are included in the DeepQ repository [59]. In particular, 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 should be compared to 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 best performing decoding agent for each error rate. For bit-flip (depolarising) noise we find that for approximately $p < 1.3 \times 10^{-2}$ ($p < 1.1 \times 10^{-2}$) there is a decoding agent for which the average lifetime of the actively decoded $d = 5$ logical qubit is longer than the average lifetime of a single faulty qubit. While these results should not be interpreted as rigorous thresholds, they can be seen as proof-of-principle demonstrations of the feasibility of using decoding agents, and in particular deepQ agents, as decoders for the fully fault-tolerant setting. In particular, in the context of near term fault-tolerant demonstrations of quantum computation, we insist that what is most important is the life-time of the logical qubit at the specific error-rate given by the experimental device. In this

respect, our proof-of-principle decoders perform competitively at code distances and error-rates expected in near-term devices.

Furthermore, there are various points worth emphasising. First, both the neural network architecture and reinforcement learning algorithm used here are comparatively simple with respect to the current state-of-the-art [46–48, 61], 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. Factoring in the ability to straightforwardly apply the techniques here to circuit-level noise renders decoding agents 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. It is important to stress that this framework is both code and error model agnostic, and hence can be directly used to train decoding agents, a novel class of flexible decoding algorithms, for a wide variety of other 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 [46–48], strongly indicate that the initial results presented here could be successfully improved and extended upon (given appropriate computational resources). With the development of dedicated special purpose hardware for the fast implementation of neural networks, decoding agents could indeed provide a practical solution for decoding in near-term fully fault-tolerant quantum computation.

There is a plethora of natural and interesting ways of extending these initial results. 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. Crucially, a key feature of this approach is the ability to straight-forwardly tackle general error-models, even including models exhibiting intricate correlations. Equally important is the consideration of techniques allowing for the scaling of neural network decoders to larger code distances. In this work we have focused on the decoding of idling logical qubits during quantum computation, encoded

via surface code patches of fixed size. For such a setting recent literature provides neural network architecture suggestions for scaling to larger code distances [41], although an alternative method for future research could be found in the use of multiple communicating decoding agents, simultaneously decoding on overlapping sub-lattices of the total code.

However, current surface code based approaches to large scale fault-tolerant quantum computing require methods for decoding of irregularly shaped and constantly changing surface code patches, formed via lattice surgery and code deformation [16, 17]. Hence, to provide a truly complete toolbox for fault-tolerant quantum computing it is necessary to provide decoding algorithms which do not explicitly depend on the underlying code lattice. For the case of decoding agents, this would require the development of methods circumventing the requirement to retrain agents for different shape surface code patches. The aforementioned approach of multiple communicating agents acting on fixed size sub-lattices could however provide a promising approach to this problem.

Furthermore, the framework presented here relies on the availability of a referee decoder, with access to the state of the underlying physical data qubits. As a result, the framework given here can not be used to train decoding agents only on experimental data, and in practice one would be required to first estimate an error model for the experimental setup. In order to remove this restriction it would be of interest to investigate modifications to the framework given here, which require only experimentally generated syndromes. One such modification would be to consider "single-shot" episodes, which terminate as soon as the agent has returned the current state into the code space.

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. It may therefore be insightful to construct test-cases (i.e. specific error volumes) to infer the learned strategy, and use this information in the design of model-tailored decoding algorithms. It is the hope that the present work constitutes a significant step forward in the understanding of the applicability of notions of reinforcement learning – with its ability to predict sophisticated situations that require an understanding of the impact of present actions into the future – in quantum information science and in the quantum technologies.

like to thank Jörg Behrmann for incredible technical support, without which this work would not have been possible. RS acknowledges the financial support of the Alexander von Humboldt foundation. MSK is supported by the DFG (CRC183, project B02). EvN is supported by the Swiss National Science Foundation through grant P2EZP2-172185. JE is supported by DFG (CRC 183, EI 519/14-1, and EI 519/7-1), the ERC (TAQ), the Templeton Foundation, and the BMBF (Q.com). This work has also received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 817482 (PASQUANS).

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 [59]. 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 Hyper-Parameters and Learning Curves

We implemented the distributed iterative training procedure described in Appendix A, for both bit-flip and depolarising
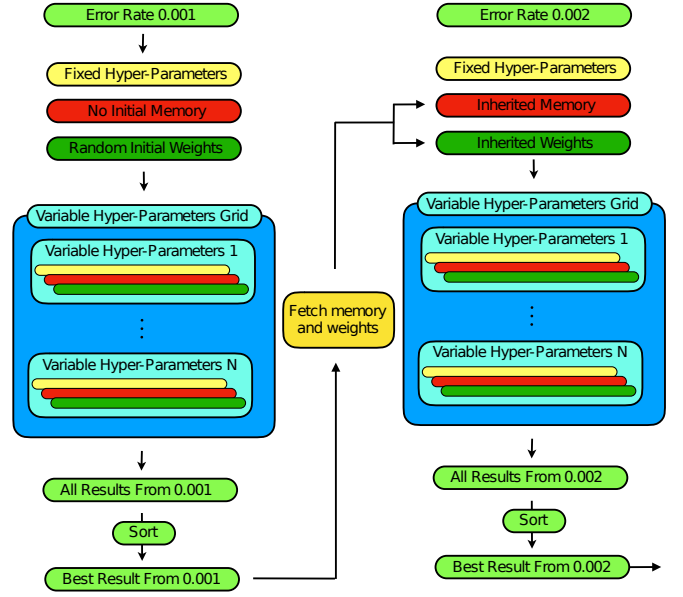

FIG. 9.   Iterative training procedure, via a hyper-parameter grid search at each error rate. After training multiple decoding agents at a given error rate, one for each point in the specified hyper-parameter grid, the results are sorted. The weights and experience memory from the optimal decoding agent 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.

noise (all utilized code can be found in the *DeepQ-Decoding* repository [59]). In particular, the initial error rate was set to $p_1 = p_{\mathrm{phys}} = p_{\mathrm{meas}} = 1 \times 10^{-3}$, and incremented by 2×10^{-3} in each iteration. As described in Section VII, from input to output the neural network architecture was as follows:

$$[[64, 3, 2], [32, 2, 1], [32, 2, 1], [512, 0.2], [|\mathcal{A}|, 0]], \quad \text{(B1)}$$

with additional modifications required for dueling methodologies [45] implemented automatically by keras-rl [60]. Single convolutional layers have been described 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 single feed forward layers via a two-tuple $[n, d]$, where n is the number of neurons and d is the output drop-out rate. All agents were trained via the original deepQ algorithm [42], implemented via the keras-rl library [60], with annealed ϵ-greedy exploration, doubleQ updates [44] and dueling networks [44]. Both the fixed hyper-parameters and variable hyper-parameter grids utilized during the training procedure are specified in Table. I. Figs. 10 and 11 then show the training history of the optimal decoding agent at each error rate, while the associated values for the variable hyper-parameters are given in Table II.

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	$\{2500, 5000\}$

TABLE I. Training Hyper-Parameters.
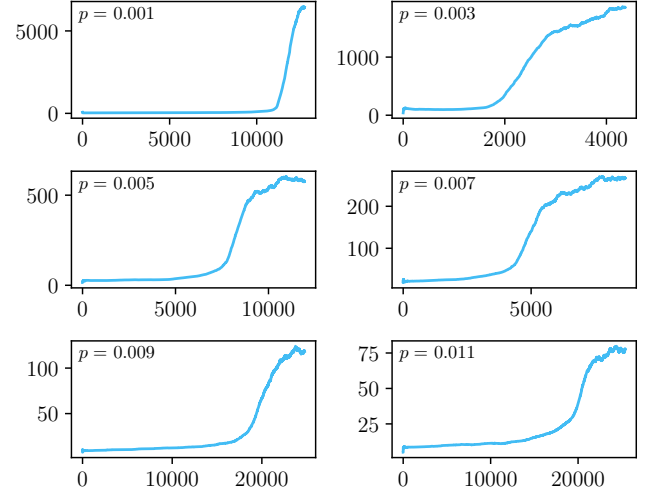
FIG. 10. Training histories for optimal depolarising noise agents. The x-axis shows the number of episodes, and the y-axis shows the rolling average of the decoded qubit lifetime. Hyper-parameters for each agent are given in Table II.

Bitflip Noise	
0.001	$[2 \times 10^5, 1, 0.02, 1 \times 10^{-5}, 5000]$
0.003	$[2 \times 10^5, 1, 0.02, 1 \times 10^{-5}, 5000]$
0.005	$[1 \times 10^5, 0.25, 0.001, 1 \times 10^{-5}, 2500]$
0.007	$[1 \times 10^5, 1, 0.02, 1 \times 10^{-5}, 5000]$
0.009	$[1 \times 10^5, 0.5, 0.04, 1 \times 10^{-5}, 5000]$
0.011	$[2 \times 10^5, 0.5, 0.04, 1 \times 10^{-5}, 2500]$
0.013	$[1 \times 10^5, 0.5, 0.001, 1 \times 10^{-5}, 5000]$
0.015	$[2 \times 10^5, 0.25, 0.04, 1 \times 10^{-5}, 2500]$
Depolarising Noise	
0.001	$[2 \times 10^5, 1, 0.001, 5 \times 10^{-5}, 5000]$
0.003	$[1 \times 10^5, 0.25, 0.02, 1 \times 10^{-5}, 5000]$
0.005	$[2 \times 10^5, 0.5, 0.001, 1 \times 10^{-5}, 5000]$
0.007	$[1 \times 10^5, 1, 0.02, 1 \times 10^{-5}, 5000]$
0.009	$[2 \times 10^5, 1, 0.02, 1 \times 10^{-5}, 2500]$
0.011	$[2 \times 10^5, 1, 0.02, 5 \times 10^{-6}, 2500]$

TABLE II. Hyper-parameters for optimal agents at each error rate, in the form [*number of exploration steps, initial ϵ , final ϵ, learning rate, target network update frequency*]
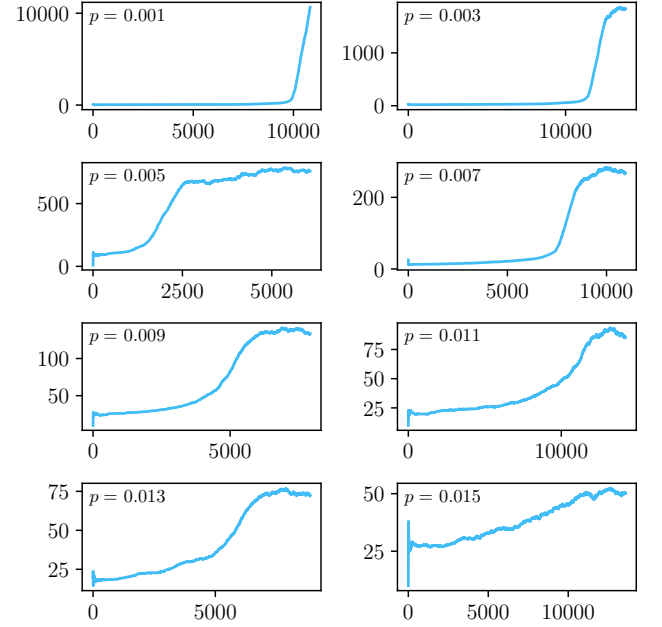
FIG. 11. Training histories for optimal bit-flip noise agents. The x-axis shows the number of episodes, and the y-axis shows the rolling average of the decoded qubit lifetime. Hyper-parameters for each agent are given in Table II.

[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. Comm. **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. Acin, I. Bloch, H. Buhrman, T. Calarco, C. Eichler, J. Eisert, D. Esteve, N. Gisin, S. J. Glaser, F. Jelezko, S. Kuhr, M. Lewenstein, M. F. Riedel, P. O. Schmidt, R. Thew, A. Wallraff, I. Walmsley, and F. K. Wilhelm, *The European quantum technologies roadmap,* New J. Phys. **20**, 080201 (2018).

[19] 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).

[20] G. Duclos-Cianci and D. Poulin, *Fast decoders for topological quantum codes,* Phys. Rev. Lett. **104**, 050504 (2010).

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

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

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

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

[25] 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).

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

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

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

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

[30] P. Broecker, F. F. Assaad, and S. Trebst, *Quantum phase recognition via unsupervised machine learning,* arXiv preprint arXiv:1707.00663 (2017).

[31] F. Schindler, N. Regnault, and T. Neupert, *Probing many-body localization with neural networks,* Phys. Rev. B **95**, 245134 (2017).

[32] 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,* Proc. Natl. Ac. Sc. , 201714936 (2018).

[33] T. Fösel, P. Tighineanu, T. Weiss, and F. Marquardt, *Reinforcement learning with neural networks for quantum feedback,* Phys. Rev. X (2018), 10.1103/PhysRevX.8.031084.

[34] G. Torlai and R. G. Melko, *Neural decoder for topological codes,* Phys. Rev. Lett. **119**, 030501 (2017).

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

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

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

[38] 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).

[39] 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).

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

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

[42] 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).

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

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

[45] 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).

[46] 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).

[47] 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.

[48] 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).

[49] D. Gottesman, *Stabilizer codes and quantum error correction*, Ph.D. thesis, California Institute of Technology (1997).

[50] H. Anwar, B. J. Brown, E. T. Campbell, and D. E. Browne, *Fast decoders for qudit topological codes,* New J. Phys. **16** (2014), 10.1088/1367-2630/16/6/063038.

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

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

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

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

[55] 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).

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

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

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

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

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

[61] 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.