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 3, 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 with respect to the unavoidable errors introduced through the interaction of physical qubits with a noisy environment. A typical 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 might occur [? ?]. While many quantum error correcting codes exist, topological quantum codes [? ? ? ? ? ? ? ? ?] in which only local operations are required to diagnose and correct errors, are of particular interest as a result of their experimental feasibility [? ? ? ? ? ? ?]. Recently the surface code has emerged as an especially promising code 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.

Within any such code based strategy for fault tolerant quantum computation, decoding algorithms play a critical role. At a high level, throughout the course of a computation these algorithms take as input the outcome of diagnostic syndrome measurements and should provide as output suggested corrections for any errors which might have occurred, which can then be tracked through the computation and later used to apply corrections to any obtained results. It is particularly important to note that in any physically realistic setting the required syndrome measurements are themselves 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 feasible 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 code scales to larger code distances.

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 criterion as possible. Perhaps most prominent are algorithms based on minimum-weight perfect matching subroutines [?], however alternative approaches based on techniques such as the renormalization group [?] and locally operating cellular automata [?] have also been put forward. Recently, techniques from classical machine learning have begun to find application in diverse areas of quantum physics - such as in the efficient representation of many-body quantum states, the identification of phase transitions, and the autonomous design of novel experimental set-ups - and various neural-network based decoders have also been proposed [? ? ? ? ? ? ?]. However, despite the diversity of decoding algorithms now available, there is not as of yet an algorithm which clearly satisfies all the required criteria, or a clear consensus as to which technique would be the most experimentally feasible in any given experimental context. MK: I think this is a bit to harsh towards MWPM/blossom, which I would say still is the best candidate and probably will one day be used for really large codes. I think we should rather say: for near term realisations of quantum error correction codes, until MWPM gets faster, a flexible and fast decoder is needed. We put forward RL decoding as one candidate. (This also does not put us in too tight a spot on delivering ideas how to tackle larger code distances.) In particular, while the so-far proposed neural network decoders promise extremely fast decoding times, flexibility with respect to the underlying noise model and the potential to scale to large code distances, all such decoders are so far restricted either to the setting of perfect syndrome measurements, or to the setting in which one is trying to store a logical qubit as long as possible, without the requirement of performing a subsequent logical gate. As such, while this approach seems promising, there remains room for improvement and generalization.

Simultaneously, the last few years have also seen extremely impressive advances in the development of deep reinforcement learning algorithms, which have allowed for the training of neural network based agents capable of obtaining super-human performance in domains such as Atari [?], Chess [?] and Go [?]. 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 problem of decoding within the context of fault-tolerant quantum computation seems like exactly such a setting and as such it natural to ask to what extent reinforcement learning techniques could be used to obtain decoding agents, and what advantages such agents might have over existing approaches. In this work we set out to provide answers to these questions.

In particular, we reformulate the problem of decoding within the setting of fault-tolerant quantum computation as a process of sequential competitive interaction between a decoding agent and a code environment. This reframing provides a conceptual framework which allows for the application of various deep reinforcement learning algorithms to obtain neural network based decoding agents. As a proof-of-principle, we then utilize to deepQ learning to obtain fast surface code decoders, for a variety of noise models, within the fully fault-tolerant setting. These results then provide a foundation for extension via both more sophisticated reinforcement learning techniques and more sophisticated neural network models.

In this work we begin by providing an introductory overview of the surface code in Section **??**, before presenting a description of the decoding problem for fault-tolerant quantum computation in Section **??**. After a brief introduction to the formalism of reinforcement learning and Q-functions in Section **??** we are then able to provide the conceptual reframing of decoding as a reinforcement learning problem in Section **??**, which is one of the primary results of this work. In Section **??** we then present deepQ surface code decoders for a variety of noise models, before finally in Section **??** discussing 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, and could be applied to any stabilizer code, but we choose to restrict ourselves to this code here for both simplicity of presentation and experimental relevance. We will focus on presenting the elements of the surface code necessary for understanding the decoding problem, in the process omitting many details, for a more detailed treatment refer to refs *a,b, and c.*

We will consider $d \times d$ lattices with a *physical data*

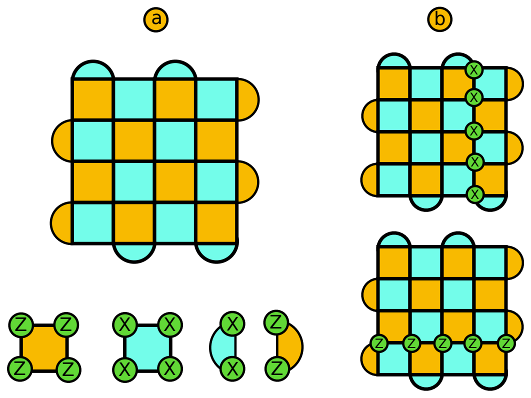

FIG. 1. An overview of the 5×5 surface code. (a) We consider square lattices, with a data qubit on each vertex of the lattice. The coloured plaquettes indicate stabilizer operators as defined in Eq. (**??**). (b) Logical X_L and Z_L operators for the surface code are given by continuous strings of single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

qubit on each vertex v, as illustrated in Fig. **??** where $d = 5$. 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,

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

All stabilizers are mutually commuting and have eigenvalues ±1. If we define the Hamiltonian $H = -\sum_p S_p$, then the surface code $\mathcal{H}_{\text{sc}} \subset \mathcal{H}$ is the ground state space of this Hamiltonian, which is the space consisting only of simultaneous $+1$ eigenstates of all the stabilizer operators. This subspace is two dimensional, i.e. $\mathcal{H}_{\text{sc}} \simeq \mathbb{C}^2$, and as a result we can encode a single *logical qubit* into this code space. Logical operators are operators which preserve the code space and manipulate the state of the logical qubit. As shown in Fig. **??**, logical X (Z) operators, denoted X_L (Z_L), are continuous strings of single vertex X (Z) operators which connect the top and bottom (left and right) boundary 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 $|\psi'\rangle \notin \mathcal{H}_{\text{sc}}b$ 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. **??** (a). The *syndrome* of a state is the string

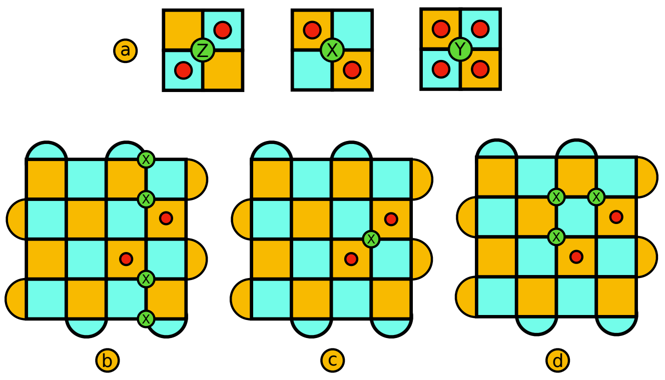

FIG. 2. (a) Single qubit Pauli flips violate surrounding stabilizers. (b-d) Strings of Pauli flips only violate stabilizers at the endpoint of the string. Multiple error configurations can give rise to the same syndrome. They can differ by stabilizers, as for example in (c) and (d), or by logical operators,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

encoding the outcomes of a simultaneous measurement of all the stabilizers, each of which takes the value ± 1. As such, unlike a truly single qubit encoding of a quantum state, if a Pauli flip error occurs on a single physical data qubit, we may be able to identify and correct this error, in the process conserving the logical qubit state, by examining the syndrome of the post-error state. This process of *decoding* is discussed in the next section.

III. THE DECODING PROBLEM

Given the foundations from the previous section, we are able to succinctly state a simple preliminary version of the decoding problem:

Assume that at $t = 0$ one is given a state $|\psi\rangle = \alpha|0_L\rangle + \beta|1_L\rangle \in \mathcal{H}_{\mathrm{sc}}$. 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'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 lattices (b), (c) and (d) of Fig. **??**, 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 in lattice (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 in lattice (c) occurred then this would be the correct operation. However, if the error in lattice (d) occurred, then the combination of the original error with the correction would implement a stabilizer. As the original state was a simultaneous $+1$ eigenstate of all the stabilizers, stabilizer operators act trivially on logical states, and hence the proposed correction preserves the initial logical state. Finally, if the error in lattice (b) occurred, then the combination of the original error with the correction would implement the logical X_L gate. As a result, even though the post-correction state is back in the code space, it will be in a different state to the original state and 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 was most likely to have 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 as such the syndrome one receives may also contain errors. In practice, each stabilizer is associated with a physical ancilla qubit, and 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, before extracting the syndrome value 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, and importantly, errors on the ancilla qubits can propagate onto the data qubits via the required entangling gates. However, 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, as illustrated in Fig. **??**. 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. As single syndromes are no longer reliable, decoding within the setting of faulty syndromes typically requires providing the decoding algorithm with multiple sequential syndromes.

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. In contemporary proposals for surface code based quantum computing

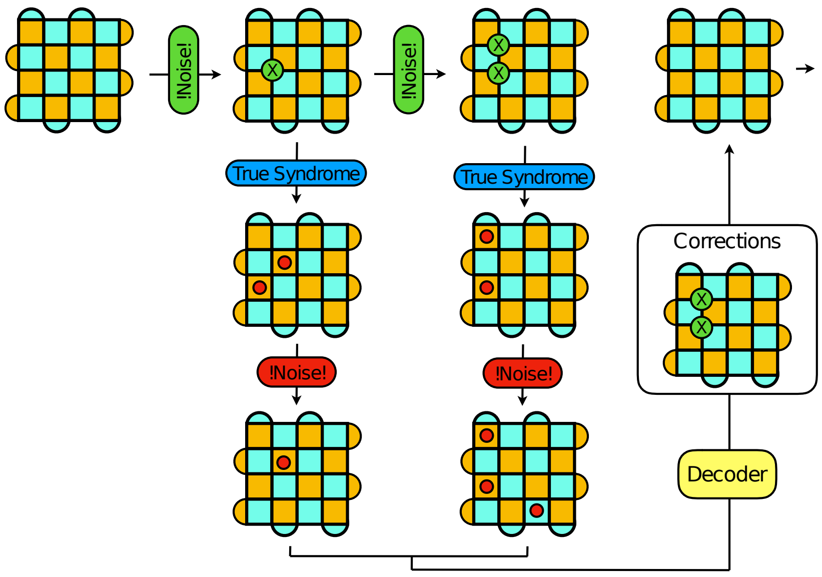

FIG. 3. A typical decoding cycle is illustrated for the simplified faulty measurements scenario in which one imagines each time step consisting of an initial physical error process generating errors on the data qubits, followed by a second measurement error process which corrupts the true syndrome. The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

protocols are known for implementing Clifford gates either transversally, via code deformation or via lattice surgery. While transversal gates do not spread errors, code deformation and lattice surgery require 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 initially faulty magic states. 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 **??** 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 crucial to our conceptual reframing of the decoding problem in Section **??**. Again, we will omit many interesting details, which can be found in the standard textbook. As illustrated in Fig. **??**, we imagine an agent interacting with some environment in discrete time steps. In each time step t the agent is in some state S_t, which is generically a combination of its knowledge of the environment $S_{E,t}$, its memory of its own previous actions, and possibly the state of additional internal variables. From this state the agent must then choose some valid action A_t, which it applies to the environment. In response to this action, via some process which is typically not directly known by the agent, the environment updates and provides the agent with a potentially reduced description of its new state $S_{E,t+1}$ (i.e. a description of the part of the environment which is *visible* to the agent), along with a a scalar reward R_{t+1} and a boolean signal which indicates whether the game is over - i.e. whether the agent has failed, died or lost. In principal it is also possible to imagine settings in which the sequence of interactions can carry on indefinitely, but we will consider here only episodic actions, which end once the environment is placed into a terminal state (which may not be unique) by the actions of the agent. Generically, we consider to goal of the agent to be to avoid dying, while accumulating as much reward as possible.

Typically the way that the agent chooses its action, the way that the environment is effected by the action, and the reward that is generated can all be stochastic, and in the context of finite state and action spaces, denoted $\mathcal{S}$ and $\mathcal{A}$ respectively, this process can be formalized within the framework of finite Markov decision processes (FMDP):

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

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 $A_t = a$ if $S_t = s$. For FMDP's we then define the *value* of a state s under policy π as,

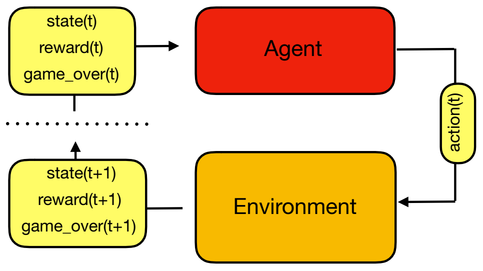

FIG. 4. An illustration of the signals passed between an agent and the environment through the duration of a sequential turn based episode.

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

$\forall S_t \in \mathcal{S}$, where G_t is the discounted return (discounted cumulative reward), with discount factor $0 \leq \gamma \leq 1$, and in practice the infinite sum terminates whenever state S_{t+k+1} is a terminal state. We call v the *state-value function*, which provides 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 an agent would obtain from following a specific policy, and 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, we can define the *action-value function* (referred to as the q-function) for policy π via:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] & (4)\\ &= \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\Big|S_t = s, A_t = a\Big] & (5)\end{aligned}$$

Clearly, the q-function with respect to a given policy is conceptually similar to the value function, except that it provides the value (expected discounted cumulative reward) associated with following a given policy after taking a certain action from a certain state. Importantly, value functions allow us to place an order over policies, i.e. $\pi > \pi' \iff v_\pi(s) > v_{\pi'}(s) \quad \forall s \in \mathcal{S}$. Given such an ordering we can then define an optimal policy π^*, with respect to which the action-value function will be the unique solution of the following system of equations:

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

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

It is now possible to introduce the idea of iterative q-learning as a tool for obtaining optimal q-functions via experience gained from interaction with a given environment. Starting from an arbitrary initial q-function, the idea is to use Eq. (**??**) as the basis for an update method, via which the q-function is updated, while an agent uses this q-function (possibly in conjunction with additional exploratory strategies) to interact with the environment. The goal is that this q function will eventually converge to q_*, which is a stationary point of Eq. (**??**). Under certain conditions, it can in fact be proven that this will be the case.

More specifically, for *deepQ* learning we parametrize q with a neural network, and use Eq. (**??**) to construct the cost function from which the network weights can be updated via some gradient descent method. In particular, we let the agent interact with the environment via an ϵ-greedy exploration policy, in the process generating experience-tuples of the following form:

$$[S_t, A_t, R_{t+1}, S_{t+1}] \quad (7)$$

From these tuples we can construct a loss on which to train the q-network, by using the cost function:

$$\begin{aligned} C &= y_{\text{pred}} - y_{\text{true}} & (8)\\ &= q(S_t, A_t) - \big[R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a')\big] & (9)\end{aligned}$$

which, by comparison with Eq. (**??**), will be 0 only for the optimal policy. Unfortunately, despite the simplicity of the idea, in practice a variety of tricks - such as batched experience replay, 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, or to the associated code repository, for details.

V. DECODING AS A REINFORCEMENT LEARNING PROBLEM

At this stage we are able to present the primary conceptual contribution of this work, which is a reframing of the decoding problem for idling logical qubits, within the context of a fault tolerant quantum computation, as a sequence of interactions between a decoding agent and a code environment. In particular, as we will see in Section **??**, the advantage of this reframing is that it allows for the methods and techniques of reinforcement learning to be applied to this problem, as a tool for obtaining decoding agents for this setting. In order to present all the required elements, as discussed in Section **??**, we will describe in detail a single episode, as illustrated in Fig. **??**.

In particular, as shown in Fig. **??** (a), the first step ($t = 0$) of any episode consists of the following sub-steps,

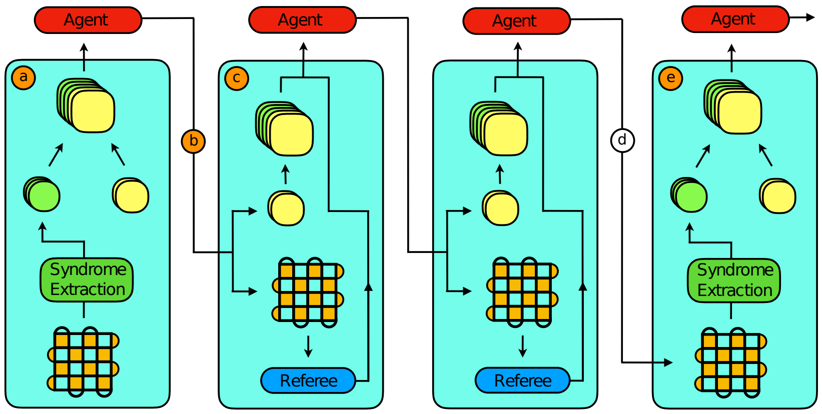

FIG. 5. An illustration of the various steps occurring within a single episode. (a) In an initial step, a faulty syndrome volume is extracted from a state initially in the code space. This faulty syndrome volume is combined with an initially empty action history and passed to the agent as the initial 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.

which generate the initial state S_1, scalar reward R_1 and boolean terminal state (or "game over") indicator T_1; Firstly, the state of the system, which we will refer to as the *hidden state* of the environment, is initialized to some logical state $|\psi_0\rangle \in \mathcal{H}_{\rm sc}$, which is the state we are trying to preserve. As this state is in the code space 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. Then, a sequence of faulty syndromes is extracted from this state. We will utilize the two-stage error process involving separated physical and measurement errors, as described in Fig. **??**, however 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. Note that at any stage during an episode this hidden state is a list containing all errors and corrections applied to the physical data qubits throughout the course of the episode. We can think of this hidden state list as representing the underlying error-plus-corrections configuration of the underlying code lattice, however at any point the current state of the system $|\psi\rangle$ could be obtained by applying all operations listed in the hidden state of the environment to the initial state $|\psi_0\rangle$. In later steps of the episode this hidden state will be used to determine rewards and terminal states, however the agent will never have direct access to this state. The visible state of the environment $S_{E,1}$ is then constructed from the extracted sequence of faulty syndromes, illustrated by the green slices in Fig. **??**, which we refer to as the *syndrome volume*. 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. **??** with the yellow slices and refer to as the action history h. As a new syndrome volume was just extracted, the action history of the agent is reset - i.e. h_1 is some representation of 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. $S_1 = (S_{E,1}, h_1)$. Furthermore, as the agent has not yet taken any actions we set the reward $R_1 = 0$, and 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. **??**, 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 $S_{t+1} = (S_{E,t+1}, h_{t+1})$. In any time step we allow the agent to perform a single qubit Pauli X or Z flip on any one physical data qubit, or to perform an identity action, which is used as a signal to request a new syndrome volume. Within the context of a reinforcement learning algorithm the agent typically chooses this action via an annealed exploratory policy, such as an ϵ-greedy strategy. Depending on the action chosen by the agent, the environment can respond in one of two ways, illustrated in Fig. **??** (c) and (e) respectively.

In particular, as shown in Fig. **??** (c), if the agent

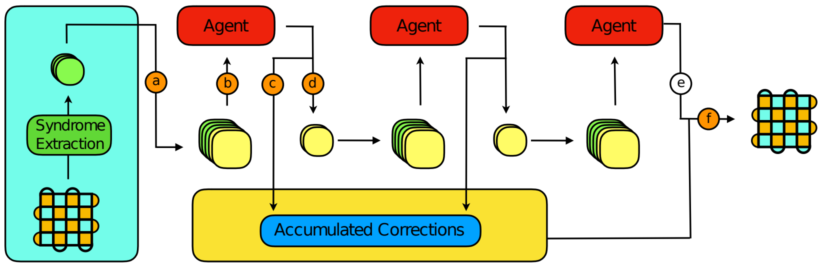

FIG. 6. The procedure for decoding with a trained agent. (a) Given a faulty syndrome volume generated by some experiment, this volume is concatenated with an empty action history to produce a suitable input state for the decoding agent. (b) The decoding agent takes in the combined faulty syndrome volume 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.

chooses any non-trivial action that it is not already in the action history list then this step of the episode continues via the following process; 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. The environment now needs to determine the reward R_{t+1} the agent should receive for its action, and whether the agent is "dead" or "alive" - i.e. whether or not the new hidden state of the environment is a terminal state. As was discussed in Section **??**, 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 agents chosen action the current state of the system $|\psi\rangle$ (obtained by applying all the errors plus actions listed in the hidden state) 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. Furthermore, in order to act as a proxy for future logical operations we introduce a *referee decoder*. 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, but 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 we say that the agent is still alive and we set $T_{t+1} = 0$, and if the agent is not able to correctly decode this perfect syndrome, then we say that the agent is dead and we set $T_{t+1} = 1$, which indicates the end of the episode. In order to construct the output state S_{t+1} we first update the action history h_{t+1} by appending A_t to h_t. Then, as the agent has not requested a new syndrome volume via the identity action, we do not extract a new syndrome volume, and rather set $S_{E,t+1} = S_{E,t}$. Finally, we concatenate the non-updated syndrome volume with the updated action history to obtain the total output state $S_{t+1} = (S_{E,t+1}, h_{t+1}) = (S_{E,t}, h_{t+1})$.

However, if the agent chooses either the identity action or an action it has already chosen (which has the effect of undoing this previously chosen correction), then the environment responds via the following procedure, as illustrated in Fig. **??** (e). If the action is not the identity action, but an action that has already been performed, 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 determined by the procedure already described above, as in steps in which a non-trivial action was taken. If the chosen action was the identity, 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 determining R_{t+1} via the criterion already discussed, which will necessarily be the same as the reward R_t received from the environment in the previous step. 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 as such a syndrome extraction procedure is performed. 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 is $S_{E,t+1}$ 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 order to provide a proof-of-principle, in Section **??** we will present a de-

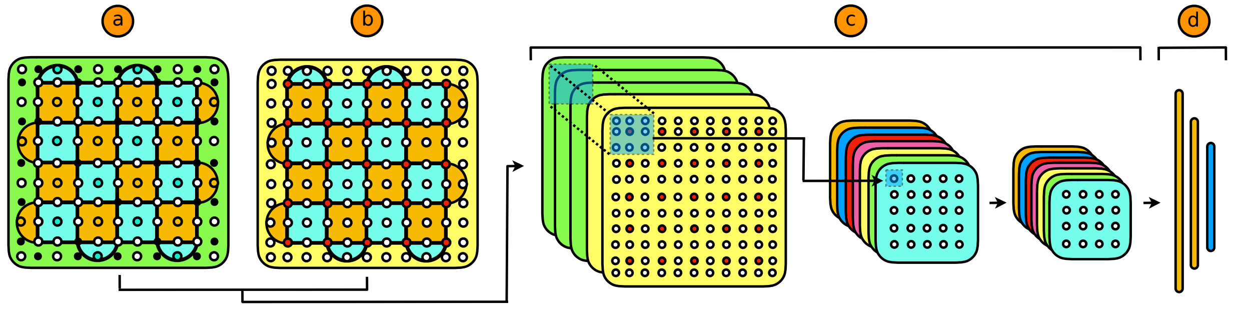

FIG. 7. Details of a deepQ decoding agent for a $d \times d$ code lattice. (a) A single faulty syndrome is embedded into a $(2d+1) \times (2d+1)$ binary matrix, where by default all entries are initially set to 0. Entries indicate by orange (blue) circles are then used to indicate violated Z (X) stabilizers 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.

tailed construction for a deepQ agent, that allows us to apply deepQ learning and obtain decoding agents whose performance is shown in Section **??**. However, before proceeding it is worthwhile to emphasize a few points; Firstly, it is important to note that the above agent-interaction framework is both code and error model agnostic, provided you have access to a referee decoder, capable of single-shot decoding with perfect syndromes. From this perspective, one might view the framework presented here as a tool for leveraging single-shot decoding algorithms, capable of dealing only with perfect syndromes, 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, 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, not currently available to alternative decoding algorithms.

Furthermore, it is useful to note that in practice it is possible to speed-up learning by utilizing various tricks. Firstly, when choosing exploratory actions, we can restrict the action space to a reduced set of potentially valid corrections. In particular, 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. Secondly, in order to only generate useful experience tuples, if a newly extracted syndrome is trivial, in the sense that it contains no violated stabilizers, then rather than passing this to the agent via the visible state we can repeat the syndrome extraction until a non-trivial syndrome volume is obtained. Once again, 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 how one could use a trained decoding agent to decode within an experimental setting. As illustrated in Fig. **??**, given a trained agent this can be done quite simply. In particular, 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. Given this input, the agent then uses its learned decoding strategy to choose an initial correction. This chosen correction is then added to a list of accumulated corrections, while simultaneously used to update *only* the action history element of the previous input state. This updated input state, containing the original syndrome volume and the subsequently performed corrections is then given to the agent, from which it can choose another correction, and the process is repeated 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 **??**, 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 or associated code repository. 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 **??** 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. **??** 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. **??** (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. **??** (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. **??** (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. **??** (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. **??** (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.

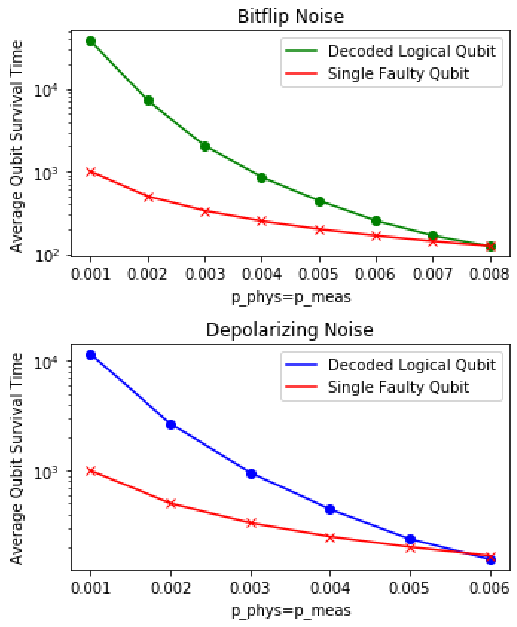

FIG. 8. These are the results.

VII. RESULTS

As a proof-of-principal, we utilized deepQ learning within the framework presented in Section **??**, to obtain deepQ decoding agents for both bit-flip and depolarizing noise, for a $d = 5$ surface code lattice. All the code utilized to obtain these agents is supplied in the corresponding *DeepQ-Decoding* code repository, 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 with annealed ϵ-greedy exploration, implemented via the keras-RL library and incorporating doubleQ updates and dueling networks. In addition we used a custom training procedure, described in Appendix **??**, 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. **??**, 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,

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

All other additional hyper-parameters used to obtain each decoding agent, along with histories of each training procedure, are provided in Appendix **??**.

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. **??**. 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 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 $p \equiv p_{\mathrm{phys}} = p_{\mathrm{meas}}$. 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. **??**, where the agent selected actions via the final q-function in a purely greedy manner, and with identities or repeated actions triggering new syndrome volumes. Importantly, we used the referee decoder to check after every action of the agent whether or not a terminal state had been reached. The qubit survival time of a single episode was then 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. Using the optimally trained decoder for each error rate, the results in Fig. **??** were then obtained by averaging over 10^3 decoding episodes (each of which contained at least 100 decoding cycles on average). As can be seen in Fig. **??**, the average lifetime of the logical qubit was then compared with the average lifetime of a single faulty qubit. For bit-flip (depolarising) noise we find that for approximately $p < 7 \times 10^{-3}$ ($p < 5 \times 10^{-3}$) the decoded logical qubit outlasts a single faulty qubit on average.

RS: TO DO: There needs to be a more informed rigorous discussion of the "threshold" - i.e why we don't bother to calculate it precisely, how it compares to known results, and why we believe the results can be made significantly better. i.e. we need to really motivate that this is only a proof of principal, and that we believe it can be scaled in many directions. There also needs to an informed discussion of why we think these agents are "fast" - i.e. on dedicated hardware (asics or TPU's) how fast can you actually implement such a small neural network?

VIII. CONCLUSION

ACKNOWLEDGMENTS

The authors gratefully acknowledge helpful and insightful discussions with Daniel Litinski, Nicolas Delfosse, Paul Baireuther, and Hendrik Poulsen Nautrup. Additionally, the authors would 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 ... 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).

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

Appendix B: Agent Hyperparameters and Learning Curves