Optimizing Quantum Error Correction Codes with Reinforcement Learning

Hendrik Poulsen Nautrup,[1, *] Nicolas Delfosse,[2] Vedran Dunjko,[3] Hans J. Briegel,[1, 4] and Nicolai Friis[5, 1]

[1]*Institute for Theoretical Physics, University of Innsbruck, Technikerstr. 21a, A-6020 Innsbruck, Austria*
[2]*Station Q Quantum Architectures and Computation Group, Microsoft Research, Redmond, WA 98052, USA*
[3]*LIACS, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands*
[4]*Department of Philosophy, University of Konstanz, Konstanz 78457, Germany*
[5]*Institute for Quantum Optics and Quantum Information, Austrian Academy of Sciences, Boltzmanngasse 3, 1090 Vienna, Austria*

(Dated: December 21, 2018)

Quantum error correction is widely thought to be the key to fault-tolerant quantum computation. However, determining the most suited encoding for unknown error channels or specific laboratory setups is highly challenging. Here, we present a reinforcement learning framework for optimizing and fault-tolerantly adapting quantum error correction codes. We consider a reinforcement learning agent tasked with modifying a quantum memory until a desired logical error rate is reached. Using efficient simulations of a surface code quantum memory with about 70 physical qubits, we demonstrate that such a reinforcement learning agent can determine near-optimal solutions, in terms of the number of physical qubits, for various error models of interest. Moreover, we show that agents trained on one task are able to transfer their experience to similar tasks. This ability for transfer learning showcases the inherent strengths of reinforcement learning and the applicability of our approach for optimization both in off-line simulations and on-line under laboratory conditions.

I. INTRODUCTION

Quantum computers hold the promise to provide advantages over their classical counterparts for certain classes of problems [1–3]. Yet, such advantages may be fragile and can quickly disappear in the presence of noise, losses, and decoherence. Provided the noise is below a certain threshold, these difficulties can in principle be overcome by means of fault-tolerant quantum computation [1, 4]. There, the approach to protect quantum information from detrimental effects is to encode each logical qubit into a number of physical qubits. This is done in such a way that physical-level errors can be detected, and corrected, without affecting the logical level, provided they are sufficiently infrequent [5]. Quantum error correction (QEC) codes thus allow for devices —usually referred to as quantum memories [6] —that can potentially store quantum information for arbitrarily long times if sufficiently many physical qubits are available. However, physical qubits will be a scarce resource in near-term quantum devices. It is hence desirable to make use of QEC codes that are resource-efficient given a targeted logical error rate. Yet, while some types of errors can be straightforwardly identified and corrected, determining the most suitable QEC strategy for arbitrary noise is a complicated optimization problem.

Here, we consider a scenario where certain QEC codes can be implemented on a quantum memory that is subject to arbitrary noise. Given the capacity to estimate the logical error rate, our objective is to provide an automated scheme that determines the most economical code that achieves a rate below a desired threshold. A key contributing factor to the complexity of this task is the diversity of the encountered environmental noise and the corresponding error models. That is, noise may not be independent and identically distributed, may be highly correlated or even utterly unknown in specific realistic settings [7, 8]. Besides possible correlated errors, the error model might change over time, or some qubits in the architecture might be more prone to errors than others. Moreover, even for a given noise model, the optimal choice of QEC code still depends on many parameters such as the minimum distance, the targeted block error rate, or the computational cost of the decoder [1]. Determining these parameters requires considerable computational resources, e.g., the computation of the minimum distance of a linear code is NP-hard [9]. At the same time, nascent quantum computing devices are extremely sensitive to noise while having only very few qubits available to correct errors.

For the problem of finding optimized QEC strategies, adaptive machine learning [10] approaches may succeed where brute force searches fail. For example, in Ref. [11], neural networks were used to determine sequences of quantum gates and measurements to correct errors. In this setting, the algorithm is required to search the whole space of encoding quantum circuits for an optimal QEC strategy. In fact, the space of all possible QEC strategies that the algorithm explores is so vast, that scaling inevitably becomes an issue. While this work demonstrates a successful strategy on up to 4 physical qubits subject to uncorrelated bit-flip errors, significant advances would be needed to generalize this to larger, potentially varying numbers of physical qubits and more realistic noise. In order to deal with such a general scenario, we follow a different approach. We first reduce the complexity of the problem of optimizing QEC strategies by separating two optimization tasks: code selection and error decoding. In this way, the QEC code can be optimized using a

* hendrik.poulsen-nautrup@uibk.ac.at

generic, unbiased, and fast decoder independently of the optimization of the latter. Indeed, neural networks have already been employed as error decoders for fixed code structures [12–20] and can be adapted to various noise models. For the problem of optimizing QEC strategies, efficient decoding goes hand-in-hand with the optimization of QEC codes.

In this paper, we present a reinforcement learning (RL) [21] framework for adapting and optimizing QEC codes. That is, our approach focuses on an adaptive code selection procedure that exploits available, sophisticated QEC techniques. In particular, we consider stabilizer codes [5], thereby significantly reducing the search space. The proposed scheme can be employed both for off-line simulations with specified noise models and for on-line optimization for arbitrary, unknown noise. In particular, we discuss how a learning algorithm trained on a *simulated* environment can *transfer* its experience to physical setups for further optimization under real conditions, capitalizing on the strength of RL. Maintaining the option to switch from one scenario to an other, we can train a learning algorithm on fast simulations with modeled environments before optimizing the QEC code under real conditions. We show that our scheme can handle simulations with up to 68 qubits and is hence sufficiently efficient for application to setups expected to be available in the near-future [22–24]. In addition, these methods are parallelizable and hence expected to perform well also for larger system sizes. Our results thus suggest that RL is an effective tool for adaptively optimizing QEC codes.

II. FRAMEWORK & OVERVIEW

We consider a situation where a learning algorithm —referred to as an 'agent' —interacts with an 'environment' by modifying a given QEC code based on feedback from the environment. As illustrated in Fig. 1, this environment consists of a topological quantum memory [6] (subject to noise) and its classical control system guiding the QEC. In each round of interaction, the agent receives perceptual input (*'percepts'*) from the environment, that is, some information about the current code structure is provided. The agent is tasked with modifying the code to achieve a logical error rate below a desired threshold. To this end, the agent is able to perform certain *actions* in the form of fault-tolerant local deformations [25, 26] of the code. The agent is *rewarded* if the logical qubits have been successfully protected, i.e., if the logical error rate drops below the specified threshold. The problem of adapting QEC codes thus naturally fits within the structure of RL. Here it is important to note that the agent optimizing the quantum memory is oblivious to the details of the environment. In particular, the agent cannot discern whether the environment is an actual experiment or a simulation.

The quantum memory we consider is based on the surface code [26, 27], one of the most promising candidates for practical QEC [28]. In particular, this versatile code is easily adaptable to compensate for a wide range of different types of errors [29]. We assume that a logical qubit is initially encoded in an 18-qubit surface code that can be extended by the agent by adding up to 50 additional physical qubits via fault-tolerant local deformations [26]. At the heart of the classical control system is the SQUAB algorithm [30, 31] simulating an optimal decoder in linear-time. SQUAB returns an estimation of the logical error rate in the (simulated) QEC procedure. As a specific learning model we employ Projective Simulation (PS) [32], which has been shown to perform well in standard RL problems [33–35], in advanced robotics applications [36], and recently PS has been used to design new quantum experiments [37].

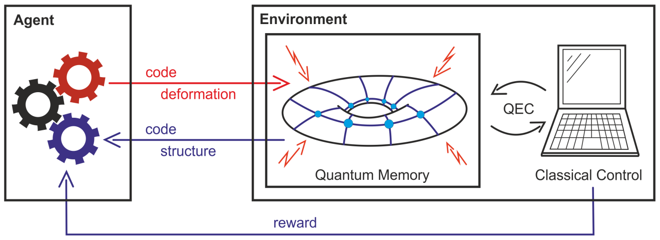


FIG. 1. **Adapting quantum memories via reinforcement learning:** In the RL paradigm, an agent interacts with an environment through actions (red) and percepts (blue). The environment consists of a surface code quantum memory subject to noise, and a classical control system that guides the quantum error correction (QEC) on the memory and estimates logical error rates. The topological quantum memory embedded on a torus can be adapted through code deformations as instructed by the agent. In turn, the agent receives perceptual input in form of the current code structure and a reward which is issued by the classical control if the logical error rate is estimated to be below a desired threshold.

Within this framework, we demonstrate that agents can learn to adapt the QEC code based on limited resources to achieve logical error rates below a desired value for a variety of different error models. Moreover, we show that agents trained in such a way perform well also for modified requirements (e.g., lower thresholds) or changing noise models, and are thus able to transfer their experience to different circumstances [38, 39]. In particular, we find that beneficial strategies learned by agents in a simulated environment with a simplified error model that allows for fast simulation of QEC [40] are also successful for more realistic noise models. This showcases that agents trained on simulations off-line could be employed to optimize physical quantum memories on-line in real-time. One reason for this adaptivity is that the agent obtains no information about the specifics of the noise, i.e., the latter appears to the agent as a black box environment. This further implies that our approach is hardware-agnostic, i.e., it can be applied to the same problem on different physical platforms, including trapped ions [8], superconducting qubits [41], or topolog-

Summary of main results

Reinforcement Learning framework

- for optimizing and adapting QEC codes, using
- arbitrary topological QEC codes,
- arbitrary decoders & noise models,
- adaptable to any optimizer (RL paradigm)
- implementable on arbitrary platforms,
- applicable off-line (simulation) & in-situ

Simulations using

- surface code quantum memory
- up to 68 physical qubits,
- optimal linear-time decoder (SQUAB)
- Projective Simulation model for RL

Simulations demonstrate agent's ability

- to determine optimal QEC codes
- for simple standard noise channels
- as well as non-isotropic noise, and for
- transfer learning in changing environments

ical qubits [42]. Finally, note that the presented adaptive QEC framework in Fig. 1 goes beyond the particular code, decoder, noise model, and RL method used here, and hence offers potential for extensions in all of these regards.

The remainder of this paper is structured as follows. In Sec. III, we briefly review the surface code in the stabilizer formalism [5] with emphasis on its inherent adaptability. Sec. IV introduces the concept of RL and the specific algorithm used. In Sec. V various different scenarios within our framework are investigated. In Sec. VI, we discuss the efficiency of our simulations and analyze the prospect of using results obtained in a simulated environment for more realistic, experimental settings. We conclude with a summary and outlook in Sec. VII.

III. ADAPTABLE QUANTUM MEMORY

A. Surface Code Quantum Memory

The purpose of a quantum memory is to encode logical information in such a way that small enough perturbations do not change the stored information. This is achieved through QEC, which can, in parts, be understood within the stabilizer formalism [5]. In the following, we will briefly illustrate this formalism for the example of a surface code quantum memory [26].

The surface code can be defined on any lattice embedded on a 2D manifold where physical qubits are located on lattice edges (see Fig. 2) [27]. For simplicity, we consider lattices embedded on tori. However, the methods introduced in this paper can be easily extended to lattices with boundaries [28]. Now, let us define products of Pauli operators for each vertex v and plaquette p as follows,

$$A_v = \prod_{i\in\mathcal{N}(v)} X_i, \tag{1}$$

$$B_p = \prod_{i\in\mathcal{N}(p)} Z_i. \tag{2}$$

Here, $\mathcal{N}(x)$ is the set of edges adjacent to a vertex $x = v$ or plaquette $x = p$ and X_i, Z_i are Pauli-operators acting on the i^{th} qubit (indices enumerate physical qubits located on the edges of the lattice). We define the surface code's stabilizer group $\mathcal{S}$ as the group generated by all A_v and B_p under multiplication. This group is Abelian, since all generating operators commute and have common eigenvectors. The $+1$ eigenspace defines the codespace $\mathcal{C}$ of the surface code. That is, a vector $|\psi\rangle \in \mathcal{C}$ iff $S|\psi\rangle = |\psi\rangle\ \forall S \in \mathcal{S}$. Elements of $\mathcal{S}$ are called stabilizers. Measuring the observables associated to these stabilizers allows checking whether the system state is still within the codespace. For a surface code defined on

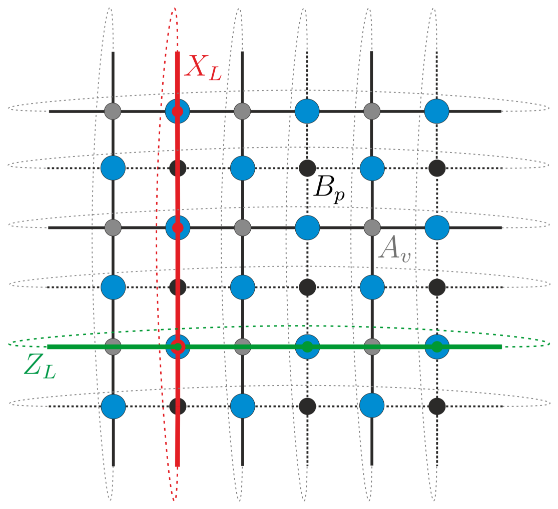


FIG. 2. Standard surface code on a 3×3 square lattice embedded on a torus. Blue colored dots represent physical qubits, black and grey dots and their connecting lines to blue dots represent Z- and X-type stabilizers respectively. The black dotted line represents the dual lattice. The thick green line depicts a possible path for a Z_{L} string and the thick red line a path for a X_{L} string. This is also the initial code considered in Sec. V.

a torus, the code space is isomorphic to $(\mathbb{C}^2)^{\otimes 2}$. It encodes 2 logical qubits into n physical qubits where n is the number of edges.

To each logical qubit one further associates a set of logical Pauli operators which define the centralizer of $\mathcal{S}$, i.e., operators that commute with $\mathcal{S}$. Trivial logical operators are stabilizers since they act as the identity on the logical subspace. In the surface code, nontrivial logical operators are strings of tensor products of Pauli operators along topologically nontrivial cycles, i.e., noncontractible paths, of the torus. Each logical qubit is defined by a logical Z-operator along a noncontractible path of the lattice and a logical X running along a topologically distinct noncontractible path of the dual lattice (see Fig. 2). Since such paths necessarily cross an odd number of times, they anticommute.

Logical operators change the encoded information. Thus, it is desirable to detect and correct errors before they accumulate to realize a logical operation. Errors which are neither stabilizers nor logical operators can be detected by stabilizer measurements since they anticommute with some stabilizers. The length of the shortest path of any noncontractible loop on the torus is the distance d of the surface code. That is, the distance is the minimal number of physical errors that need to occur in order to realize a logical operator. Any number of errors below d can be detected but not necessarily corrected and less than $d/2$ errors can always be corrected in the stabilizer formalism [5].

B. Adaptable Surface Code

So far, we have assumed the standard notion of a surface code defined on a square lattice on a torus (see Fig. 2). However, surface codes can be defined on arbitrary lattices. Since the surface code stabilizers in Eqs. (1) and (2) are identified with vertices and plaquettes of the lattice, adapting the lattice can change the underlying code and its performance dramatically [29]. This feature makes surface codes particularly useful for biased noise models, e.g., noise models where X- and Z-errors occur with different probability. Even the loss tolerance is affected by the underlying lattice.

In our framework, we will be exploring the space of lattices to make use of the adaptability of the surface code. To search this space in a structured fashion, we introduce a set of elementary lattice modifications.

Specifically, we consider modifications that can be performed with a set of basic moves which were first illustrated in Ref. [26]. The two basic moves are exemplified in Fig. 3: (i) As illustrated in Fig. 3(a), addressing the primal lattice, two non-neighboring vertices can be connected across a plaquette which is effectively split by the additional edge. That is, the number of Z-stabilizers is increased while an additional physical qubit is added. (ii) Conversely, the same operation can be performed on the dual lattice producing an additional vertex (i.e., X-stabilizer) as illustrated in Fig. 3(b). These moves can be implemented fault-tolerantly [1] and map a surface code to a surface code while changing the underlying stabilizer group [26]. This can be considered as simple version of code deformation [25] which is employed here to explore the space of surface codes. Indeed, code deformation can in principle be employed in a much more general way to map topological stabilizer codes to other topological stabilizer codes. However, for our needs the two basic moves explained above already allow for sufficiently diverse modifications. In particular, the number of available moves increases with the lattice size.

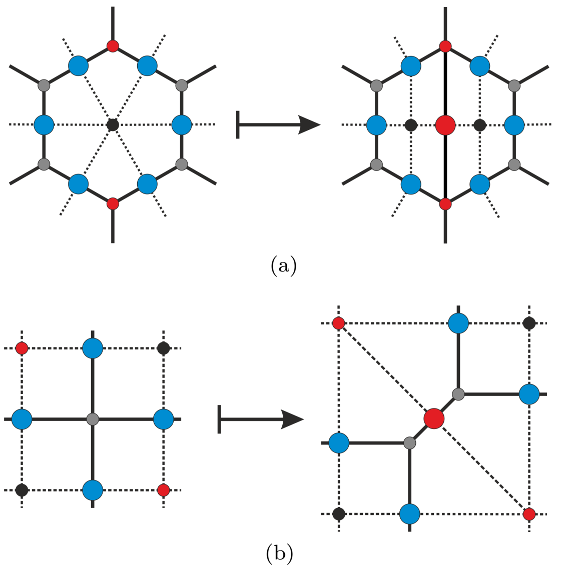


FIG. 3. An illustration of the basic moves that fault-tolerantly map surface to surface codes while changing the underlying lattice and hence, the stabilizer group. (a) Two vertices (here chosen at the top and bottom in red) are connected across a plaquette (central black dot) which is effectively split into two by a new edge. (b) Two plaquettes (red dots on top left and bottom right) are connected across a vertex (central grey dot) which is effectively split into two by a new edge. In both cases the number of physical qubits (blue dots) is increased by one (central red) qubit. Note that move (b) can be understood as a move of type (a) on the dual lattice and vice versa.

IV. REINFORCEMENT LEARNING AND PROJECTIVE SIMULATION

In generic RL settings [21], one considers an agent that is interacting with an environment (see Fig. 1). At each

[1] Here, fault-tolerance requires changes to be local w.r.t. the lattice topology. More specifically, the circuit that implements the changes has to act on a small number of physical qubits. In this way, errors can spread only to a limited number of qubits even if each gate in the circuit fails.

time step the agent obtains information from the environment in terms of perceptual input – percepts – and responds through certain actions. Specific behavior, i.e., sequences and combinations of percepts and actions, trigger reinforcement signals – rewards. Typically, the agent adapts its behavior to maximize the received reward per time step over the course of many rounds of interactions. The RL method as we use it makes no assumptions about the environment and applies even if the environment is a black box. In our case, the environment is a surface code memory subject to noise and its classical control system, including the means to estimate logical error rates. Here, knowledge of the environment is also restricted: The agent is never directly provided with information about the noise model. This is because, in practice, acquiring knowledge about the noise requires an involved process tomography. Since the agent does not aim to characterize the noise process, but rather to alleviate its effects, such an involved noise estimation procedure is not necessary.

The specific learning agent that is considered in this paper is based on the Projective Simulation (PS) model for RL. PS is a physics-motivated framework for artificial intelligence developed in Ref. [32]. The core component of a PS agent is its clip network which is comprised of units of episodic memory called *clips* (see Fig. 4). There are two sets of clips constituting the basic network, *percept* and *action* clips. In an interactive RL scenario, an agent *perceives* the environment through the activation of a percept clip $s_i \in P$ and responds with an *action*. The latter, in turn, is triggered by an action clip $a_j \in A$. Percept clips can be regarded as representations of the possible states of the environment, as perceived by the agent. Similarly, action clips can be seen as internal representations of operations an agent can perform on the environment. A two-layered clip network as in Fig. 4 can be represented by a directed, bipartite graph where the two disjoint sets comprise the percepts P and actions A, respectively. In this network each percept (clip) $s_i \in P$, $i \in [1, N^{(t)}]$ (where $N^{(t)}$ is the number of percepts at time step t) is connected to an action (clip) $a_j \in A$, $j \in [1, M_i]$ (with M_i being the number of actions available for a percept s_i) via a directed edge (i, j) which represents the possibility of taking action a_j given the situation s_i with probability $p_{ij} := p(a_j|s_i)$. The agent's policy governing its behavior in this RL setting is defined by the transition probabilities in the episodic memory. Learning is manifested through the adaption of the clip network via the creation of new clips and the update of transition probabilities. Each time a new percept is encountered, it is included into the set P. A time-dependent weight, called h-value, $h_{ij}^{(t)}$ is associated with each edge (i, j). The transition probability from percept s_i to action a_j is given by the so-called softmax function [21] of the weight $\sigma_\beta(h_{ij}^{(t)})$,

$$p_{ij}^{(t)} = \frac{e^{\beta h_{ij}^{(t)}}}{\sum_{k=1}^{M_i} e^{\beta h_{ik}^{(t)}}}, \tag{3}$$

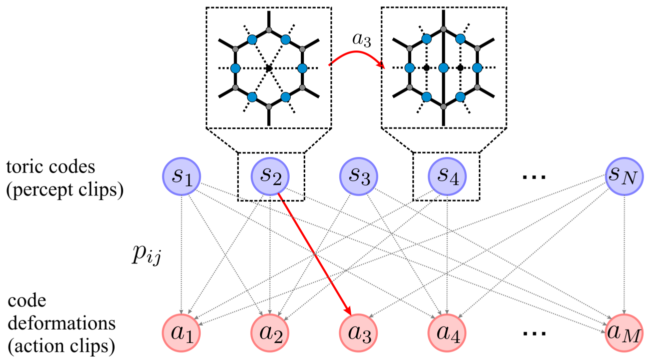


FIG. 4. Two-layered clip network of a PS agent. Percept clips in the first (upper) layer are colored blue and represent surface codes. Action clips are colored red and represent code deformations. Edges (i, j) carry weights that can be associated with transition probabilities p_{ij} in accordance with Eq. (3). If an action, say a_3, is performed upon seeing a percept, say s_2, the code associated with s_2 is adapted according to a_3. This results in a new code, say s_4, that is perceived by the agent. Note that not every percept is necessarily connected to every action since the set of available moves in Fig. 3 depends on the lattice, i.e. the percept.

where $\beta > 0$ is the softmax parameter.

When a percept is triggered for the first time all h-values are set to 1 such that the transition probabilities are initially uniform. That is, the agent's behavior is fully random in the beginning. Since random behavior is rarely optimal, changes in the transition probabilities are required. The environment reinforces such changes by issuing nonnegative rewards $\lambda^{(t)}$ in response to an action of the agent. Then, the agent must ask the question, given a percept s_i which is the action a_j that will maximize the received reward. Therefore, the transition probabilities are updated in accordance to the environment's feedback such that the chance of the agent to receive a reward in the future is increased. In other words, the environment's feedback $\lambda^{(t)}$ controls the update of the h-matrix with entries h_{ij}. However, there are many other contributions to the update independent of the environment's immediate feedback. Particularly noteworthy are contributions that reinforce exploratory over exploitative behavior. A detailed description of how the feedback is processed in the agent's memory is given in Appendix A.

Once an agent has learned to optimize the reward per interaction step, the clip network in Fig. 4 is a representation of the agent's policy: Nonuniform transition probabilities p_{ij} mark a systematic decision-making aimed at maximizing the reward per time step while accounting for exploratory behavior. Hence, the network can also be interpreted as an agent's memory encoding knowledge of the reward landscape of the environment.

V. OPTIMIZING QUANTUM MEMORIES - A REINFORCEMENT LEARNING PROBLEM

In this section, we show that RL can be employed within the framework of Fig. 1 to successfully optimize QEC codes w.r.t. their resource efficiency in settings with up to 68 qubits. In our framework for adaptive quantum error correction, we consider a PS agent that interacts with a specific, simulated environment – the surface code quantum memory subject to noise and its classical control managing the QEC procedure. We use the term environment in the sense it is used in RL rather than referring to the source of noise as is common in, say, open-system quantum physics.

To be more precise, we start each trial by initializing a distance-three surface code quantum memory with 18 qubits (edges) defined on a square lattice Λ embedded on a torus (see Fig. 2). Note that the choice of initial code is ad hoc and could be replaced by any other small-scale surface code designed to suite a given architecture. The code is subject to an unknown Pauli noise channel $\mathcal{E}$ which may change in time and may differ for different physical qubits. The classical control simulates the QEC procedure under this noise model and estimates the logical error rate P_{L}. Having a basic set of moves at its disposal (see Fig. 3), the agent is tasked with growing the lattice until the logical error rate is below a threshold $P_{\mathrm{L}}^{\mathrm{rew}}$ or at most 50 additional qubits have been added. This choice of an upper limit should be viewed in light of recent progress in the development of quantum devices with more than 50 qubits [23, 24]. Note the difficulty of the simulation task. A single trial requires to simulate the perfomance of up to 50 QEC codes. The basic set of moves is rather generic and could be restricted further to suite the requirements of a given hardware. For instance, actions could be restricted to the boundary of a system. Once the desired logical error rate is reached, the agent receives a reward $\lambda = 1$, the trial ends and the quantum memory is reset to Λ. If the desired logical error rate is not reached before 50 qubits have been introduced, the code is reset without reward. The details of the algorithm, specifically the environment, are presented in Appendix B and the agent's clip network, i.e., episodic memory, is visualized in Fig. 4. Note that this scenario – although restricted to surface codes – is already extremely complex and versatile. In Appendix C, we analyze the difficulty of this problem and show that there is no hope of solving it by random search.

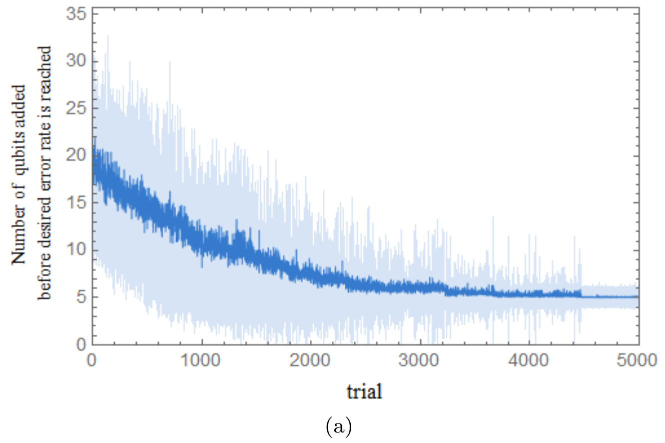
In the following, we verify that the task outlined above is indeed a RL problem where learning is beneficial. To this end, we start by considering some instructive cases where the solutions are known before moving on to a more complicated scenario with unknown optimal strategy. The variables that are being changed in between scenarios are the error channel $\mathcal{E}$ or the rewarded logical error rate $P_{\mathrm{L}}^{\mathrm{rew}}$. However, if not stated otherwise, we set $P_{\mathrm{L}}^{\mathrm{rew}} = 0.001$. For illustrative reasons, the error models $\mathcal{E}$ that appear in the following are represented by single-qubit Pauli error channels. In fact, the simulated quantum memory will be subject to a so-called erasure channel [43, 44] which models the behavior of the actual Pauli channel. This particular choice of error model is motivated by the availability of an optimal decoder for erasure noise on a surface code [40]. In Sec. VI A, this simplified channel is analyzed in more detail and we verify that it is suitable to model actual error channels.

A. QEC codes for i.i.d. error channels

As a first simple task, we consider two straightforward scenarios with known good solutions as they will nonetheless be crucial in the evaluation of the behavior of the RL agent. First, let us consider an error channel that can only cause Z-errors with probability $p = 0.1$ on each individual, physical qubit independently. Formally, we can write this as a quantum channel acting on each single-qubit state ρ as,

$$\mathcal{E}(\rho) = p\, Z\rho Z + (1-p)\rho. \tag{4}$$

Assuming the initial logical error rate is above threshold $P_{\mathrm{L}}^{\mathrm{rew}}$, the RL agent is then tasked with modifying the 18-qubit surface code by increasing the number of physical qubits according to the rules described in Fig. 3. At the beginning of each trial, an agent has a reservoir of 50 additional qubits at its disposal to reduce the logical error rate below $P_{\mathrm{L}}^{\mathrm{rew}} = 0.001$. Fig. 5(a) shows that the agent indeed learns a strategy, i.e., a policy, which adds – on average – 5 physical qubits to the initial code in order to reduce the logical error rate as desired. Fig. 5(a) can be understood as follows. In the very beginning, the agent has no knowledge about the environment and performs random modifications of the surface code. For the specific error channel in Eq. (4), a random strategy requires on average 20 additional physical qubits to reach the desired logical error rate $P_{\mathrm{L}}^{\mathrm{rew}}$. What follows are many trials of exploring the space of surface codes. During each trial, the agent's episodic memory is reshaped and modified as described in Sec. IV and Appendix A. This learning process effectively increase the probability for the agent to select sequences of actions which lead to a reward quickly. As can be seen from Fig. 5 the length of a rewarded sequence of actions is gradually reduced with each trial. Ideally, the agent's behavior converges to a policy that requires a minimum number of additional qubits. In Fig. 5(a) we observe that agents converge towards a policy which requires, on average, 5 additional qubits. Let us now confirm that the strategy found by the RL agents agrees with the best known strategies for this problem. The error channel in Eq. (4) can only cause logical Z-errors. That is, we are looking for the surface code with the maximum number of X-stabilizers and minimum number of Z-stabilizers [29]. Starting from the surface code in Fig. 2 we must therefore ask the question how to increase the number of X-stabilizers given the available actions in Fig. 3. Since X-stabilizers are



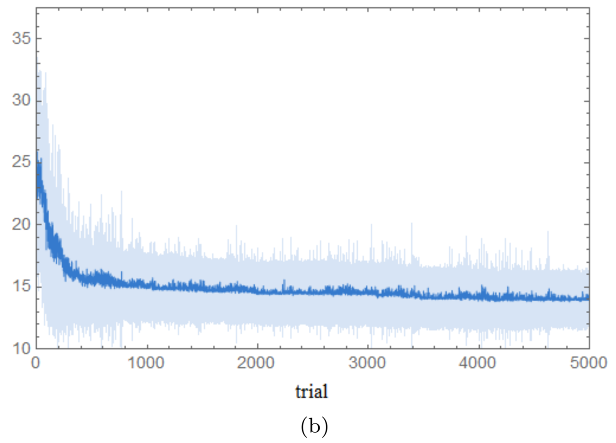


FIG. 5. The figure shows the results for two simple tasks as considered in Sec. V A which confirm that RL learning can be applied to the problem of optimizing quantum memories. The area shaded in light blue is the standard deviation from the average over 60 agents. Given the available set of actions, shown in Fig. 3, the PS agent can add, one-by-one, qubits to the initial code (see Fig. 2) until the logical error rate is below $P_{\mathrm{L}}^{\mathrm{rew}} = 0.001$. (a) The memory is subjected to the noise channel in Eq. (4) with $p = 0.1$. The initial logical error rate is $P_{L}^{\mathrm{init}} \approx 0.006$. The number of qubits to reach the desired error rate $P_{\mathrm{L}}^{\mathrm{rew}}$ per trial is depicted. (b) The same task is shown with the error channel of Eq. (5) such that $P_{L}^{\mathrm{init}} \approx 0.005$. Different learning parameters (see Appendix A) are reflected in different learning times, standard deviations and convergence behavior. Generally, there is a trade off between learning time and standard deviation as can be seen from the comparison between (a) and (b). Shaded areas represent the standard deviation. Details about parameters are given in Appendix D.

identified with vertices in the graph, repeatedly applying an action as displayed in Fig. 3(b) will increase the number of X-stabilizers. We hence expect a sequence of these actions to provide good strategies in this case, resulting in a surface code on a lattice with low connectivity. Indeed, we can confirm this by looking at surface codes constructed by agents that have been successfully applied to this task. In Fig. 6, a particularly interesting example solution is shown: It can be seen that the distance of the code w.r.t. Z-errors along one cycle has been increased from 3 to 4 by inserting 4 new qubits at very specific points in the lattice. In other words, any logical Z-operator crossing the lattice from left to right in Fig. 6 is a product of at least 4 single-qubit Z-operators. Just looking at the initial lattice in Fig. 2 it is not obvious that this can be done with only 4 actions. This is already a nontrivial result for this, at first glance, simple problem.

Now, let us consider the well-understood scenario of a depolarizing error channel. That is, each qubit in the quantum computer is subject to an i.i.d. quantum channel of the form

$$\mathcal{E}(\rho) = p_X\, X\rho X + p_Z Z\rho Z + (1 - p_X - p_Z)\rho, \qquad (5)$$

where we choose $p_X = p_Z = 0.09$ such that either X- or Z-errors can happen with probability 0.09 everywhere. Otherwise the task remains the same as before. We can see from Fig. 5(b) that the agent again learns to improve the logical error rate while optimizing over the number of required qubits.

It is well known [29] that the optimal surface code to protect against depolarizing noise is defined on a square (self-dual) lattice. That is, the best known strategy to grow the surface code using the actions available (see Fig. 3) is to retain the same connectivity both on the primal and dual lattice such that there are the same number

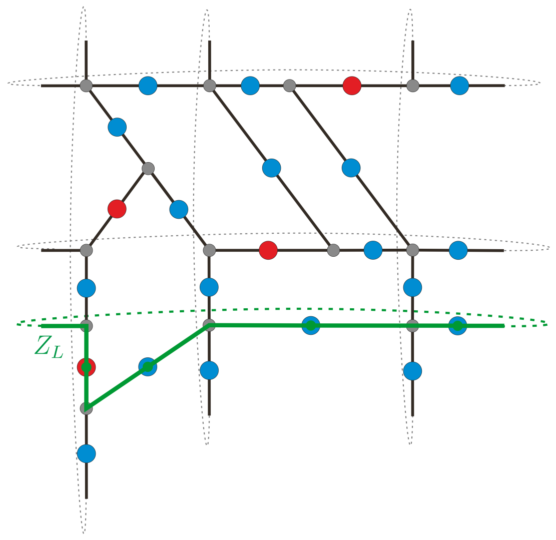


FIG. 6. The best agents only require 4 additional qubits to solve the task described in Sec. V A with the error channel in Eq. (4). Here, an exemplary solution is displayed. That is, a surface code with a logical error rate below $P_{\mathrm{L}}^{\mathrm{rew}} = 0.001$. Red circles represent qubits that have been added to the code. Removing all red qubits, we recover the initial code (see Fig. 2). As expected, all actions increase the number of X-stabilizers in order to protect against Z-errors. The thick green line depicts a possible path for a Z_{L} string crossing 4 qubits.

of X- and Z-stabilizers. Indeed, looking at some examples from agents trained on this task, we can confirm that the agents also learn this strategy: The most successful agents end up creating surface codes where the number of X- and Z-stabilizers differ by at most 1. On average, the ratio between number of X- and Z-stabilizers is 1.06 with standard deviation 0.15. The corresponding primal and dual lattices tend to have similar average connectivities, too. The average ratio between primal and dual connectivity is 1.06 with standard deviation 0.15.

B. QEC codes for correlated error channels

Next, let us tackle a complicated, practical scenario where the optimal strategy is unknown: We consider spatial dependencies on top of an i.i.d. error channel. This particular situation is motivated by the common problem of manufacturing defects. Correlated errors arising e.g., from manufacturing defects can be present in any fault-tolerant architecture because active error correction requires multi-qubit operations such as stabilizer measurements. Most architectures using topological codes have a spatial layout which can be associated with the lattice of the surface code [41, 45] such that correlated errors will most likely be local. Consider for instance an ion trap quantum computer using an arrangement of multi-zone Paul trap arrays [46]. Dephasing is the prevalent noise in ion traps [8], so we would already have to consider an asymmetric error channel. Moreover, due to e.g., a manufacturing defect, two Paul traps in this arrangement could fail and produce X-errors on internal qubits.

To be precise, consider the error channel in Eq. (5), with $p_X = 0.02$ and $p_Z = 0.1$. This is similar to the simplest case in Sec. V A where $p_X = 0$. In our construction, a correlated error as motivated above is modeled by an increased X-error rate on edges in the neighborhood $\mathcal{N}(i)$ of a vertex or plaquette i. Here, we choose two neighboring plaquettes i, j and modify the error channel as follows,

$$\mathcal{E}_k(\rho) = p_{X,k}\, X\rho X + p_{Z,k} Z\rho Z + (1 - p_{X,k} - p_{Z,k})\rho, \tag{6}$$

where k labels the qubits. We further assume that all qubits have base error rates of $p_{X,k} = 0.02$ and $p_{Z,k} = 0.1$. In addition, the base probability $p_{X,k}$ of an X-error on qubit k is increased by 0.5 if $k \in \mathcal{N}(i) \cup \mathcal{N}(j)$ and $p_{X,k} = 1$ if $k \in \mathcal{N}(i) \cap \mathcal{N}(j)$. This serves two purposes. First, we can evaluate the behavior of the agent with regard to the two special plaquettes i, j. How is the lattice structure changed in and around these plaquettes? Second, we can understand how the agent handles a deterministic error on the edge neighboring both i, j. Will this edge be treated differently? In fact, it is far from clear what the optimal strategy is given the available actions displayed in Fig. 3. Nevertheless, we observe that the PS agent learns to protect the surface code against this error channel while optimizing over the resources used (see Fig. 7).

Now, let us evaluate successful stategies employed by the best agents in this scenario. In Fig. 8 the relevant part of the final dual lattice as constructed by a successful agent is depicted. We can see that the agent increases the number of Z-stabilizers in the immediate vicinity of the flawed plaquettes, decreasing the connectivity of affected plaquettes. Interestingly, the agent also finds a way of moving the flawed plaquettes apart given the available actions, thereby removing any deterministic error appearing on the edge between these plaquettes. At the same time, due to the prevalent Z-errors, connectivity throughout the lattice is balanced between vertices and plaquettes: the average connectivity of the dual lattice is 3.9 and 4.1 for the primal lattice. Similarly, the ratio between X- and Z-stabilizer is 14/15.

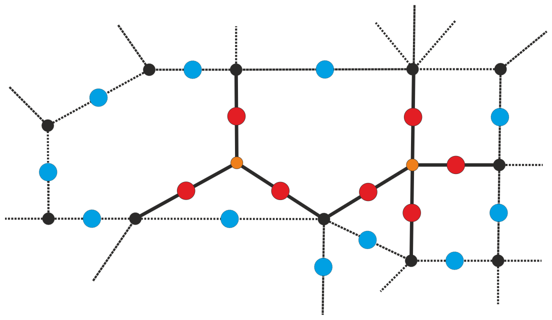


FIG. 8. Part of the final dual lattice constructed by one of the most successful agents in the task described in Sec. V B where noise has spatial dependencies (see Eq. (6)). Qubits that are affected by high X-error probabilities are colored in red and surround flawed plaquettes colored in orange.

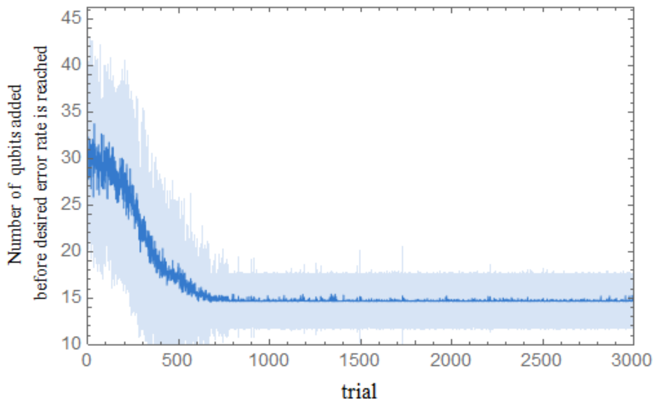


FIG. 7. Learning curve for the task specified in Sec. V B averaged over 60 agents. The shaded area shows the standard deviation. Here the quantum memory is subject to a noise channel with spatial dependencies as in Eq. (6). The initial logical error rate is as high as $P_L^{\mathrm{init}} \approx 0.28$. Details about parameters are given in Appendix D.

VI. SIMULATION VS. EXPERIMENT

In the previous sections, the main focus has been to determine whether RL agents in our framework are able to adapt and optimize a surface code quantum memory to various types of noise. We have found that this is indeed the case, showcasing the prospect of RL in on-line optimization of quantum memories. Although we have evaluated our procedures only via simulations, our results suggest that such approaches can be successful also in practice in future laboratory experiments. This is because our framework for optimizing QEC codes in Fig. 1 is independent of whether the environment is a real or simulated quantum device. In either case, the interface between environment and agent remains unchanged. For instance, we estimate the logical error rate of the quantum memory using a Monte Carlo simulation. In a real device, this can be done by preparing a quantum state and by measuring the logical operators after a fixed amount of time. Repeating this measurement provides an estimation of the lifetime of the quantum memory. Moreover, the code deformations which constitute the actions available to an agent are designed with a physical device in mind [25, 26].

Ultimately, one is of course interested in performing optimization on actual quantum devices. However, as long as these devices are sufficiently small-scale to allow for simulations that are (much) faster than the timescale of operating the actual device[2], it is advantageous to perform the bulk of optimizations off-line in simulations before transferring the results to the actual device for further optimizations. In other words, to fully capitalize on the important features of transfer-learning and pretraining in quantum-applied machine learning, it is crucial to ensure that the simulations are as efficient as possible, and that the chosen RL model is able to transfer experience from simulations to real environments. In the following, we therefore discuss the efficiency and practicality of our simulations and show that the RL agents we consider are capable of transfer learning.

A. Simulation Efficiency of QEC

To provide sufficiently efficient off-line simulation, the individual components of our QEC simulation have been carefully selected. For instance, note that stabilizer QEC can be simulated efficiently classically [48–51]. However, estimating the logical error rate, which requires a large number of samples from QEC simulations, remains computationally expensive. In our simulations, we hence make use of a simplified error model to allow for faster estimations of logical error rates. The simplified error model is based on the quantum erasure channel [43] since there exists a linear-time maximum likelihood decoder for surface codes over this channel which is optimal both in performance and speed [40]. The use of this software, SQUAB [30, 31], within our RL framework is hence providing the means for sufficiently fast exploration of the space of surface codes. In essence, the erasure error channel is very similar to a Pauli error channel where the location of an error is known exactly. More specifically, we introduce an asymmetric erasure channel where we choose two erasure probabilities p_e^X, p_e^Z that can have spatial and temporal dependencies. Since X and Z stabilizers are separated in the surface code, error correction of X and Z errors can also be treated independently. In the simulation of X errors over the erasure channel, we erase each qubit ρ in the surface code with probability p_e^X, and replace it by a mixed state of the form $\rho' = \frac{1}{2}(\rho + X\rho X)$. The set of erased qubits is known. The simulation proceeds analogously for Z errors.

Since our simulations cover setups with up to 68 qubits, we have indeed good reason to believe that our simulations are sufficiently efficient to optimize QEC codes for near-term quantum devices without the need of experimental data. However, one may argue that our software only provides a simulation for a simplified noise model, the erasure channel. In order to prove that the results are relevant in practice, we compare logical error rates of a set of small surface codes subject to erasure errors to the rates obtained from simulating standard Pauli noise on the same codes. To this end, we assume that each qubit is affected independently by a Z-type Pauli error and we perform error correction with both SQUAB, and the Union-Find decoder introduced in Ref. [47]. We report the average logical error rate for each code after 10,000 trials with an erasure rate $p_e^Z = 0.1$ (SQUAB) and Z-error rate $p_Z = 0.1$ (Union Find). In Fig. 9 we observe qualitatively the same behavior: codes that are considered to perform well according to the estimation using SQUAB are also considered to perform well using the Union-Find decoder. The difference between the two error channels lies predominantly in the magnitude of the logical error rates. It is therefore better to select codes using SQUAB, allowing for a faster and thus more precise exploration of the space of topological codes.

B. Transfer learning in QEC

The usefulness of off-line simulations for QEC code optimization emerges from the application of the results to on-line settings. In this transition, deviations of the error model used in the simulation from the actual noise, or dynamical changes of the latter can lead to non-optimal performance if no further action is taken. Here, machine

[2] Here it is relevant to acknowledge variations in hardware constraints and adaptability of currently available setups. For instance, limitations on how freely qubits that are physically present but not yet part of a specific code structure can be connected to the latter vary strongly between different platforms such as trapped ions [8, 45, 46] and superconducting qubits [41].

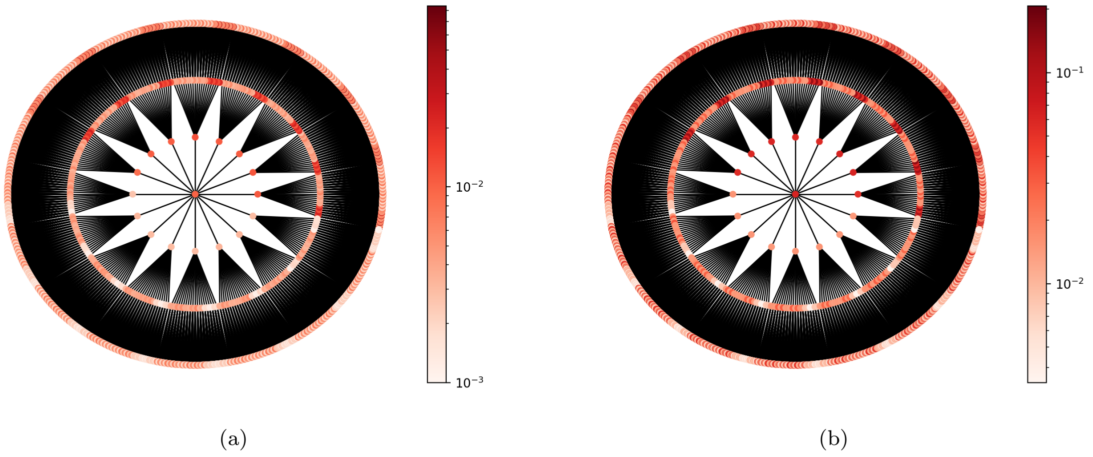


FIG. 9. Comparison of logical error rates estimated by (a) SQUAB [40] and (b) the Union Find decoder [47]. Each node in the tree corresponds to an individual surface code on a torus and edges imply that two codes are connected by an action as in Fig. 3. The central node is the 2×2 surface code on a square lattice. We explore all codes up to a distance 3 from the root. The color shading indicates the logical error rate. (a) The logical error rate is estimated using SQUAB with 10,000 trials and erasure rates $p_e^Z = 0.1$, $p_e^X = 0$. (b) The logical error rate is estimated using the Union-Find decoder with 10,000 trials and a Z-error rate $p_Z = 0.1$. While the logical error rates differ in both cases, the behavior is qualitatively the same for a code subject to an asymmetric erasure channel and the same code affected by Pauli noise. That is, codes found to provide lower logical error rates w.r.t. other codes for one noise model also provide lower logical error rates than other codes for the other error channel.

learning comes into play. That is, a central agenda of machine learning is to develop learning models that are capable of successfully applying previously obtained knowledge in different environments. This possibility, called transfer learning [38], is an active area of research in general AI [39]. Transfer learning can provide a significant improvement in performance as compared to optimization approaches without learning mechanisms, as well as considerable reductions of learning times in comparison to untrained RL agents. Put simply, agents capable of transfer learning do not have to start from scratch every time the environment or task changes slightly [37, 52–54]. As we will discuss in this section, within the RL setting we consider here, agents trained on the original simulations may *transfer* their experience from simulations to practical applications. The usefulness of this approach will of course depend on how similar the simulated error model is to the real error channel.

In order to demonstrate the potential of transfer learning in the context of optimizing and adapting quantum memories, we consider a scenario where the agent is first trained on one error model, and then encounters a slightly different error model. This change in noise could occur not only because the agents are switched from a simulated training scenario to an actual experiment but also, e.g., due to environmental changes, equipment malfunctions or malicious attacks. Generally, under the assumption that the noise can only vary slightly, we can expect learning to be beneficial since strategies that helped in one scenario should still be favorable in protecting against similar error channels. Then, a RL algorithm can further optimize over a given policy and re-adjust to the new environment. This scenario capitalizes on the strength of RL since it requires a long-term memory that can be exploited between tasks. If exploration of the search space is encouraged over exploitation of rewards, the agent's memory contains global information about the environment. In other words, given some sub-optimal QEC code, such an agent knows a strategy to adapt this code efficiently such that the desired logical error rate is reached. In fact, the feedback which is given only if a certain logical error rate has been achieved, is specifically designed to encourage exploration of the search space.

The specific choice of error model for this scenario is motivated by the results in Fig. 9. There we observe that the main difference between the simulated erasure and more realistic Pauli error channels lies in the magnitude of the estimated logical error rate. Therefore, we consider a quantum memory which is at first subject to the noise channel in Eq. (4) with $p = 0.14$ and the agent is tasked again to reduce the logical error rate below $P_{\mathrm{L}}^{\mathrm{rew}} = 0.001$. Then, after having learned a beneficial strategy for finding a good QEC code, the agent is confronted with an increased error rate of $p = 0.16$. Fig. 10(a) shows that, in this second stage of the task, the agent benefits from having learned to protect the quantum memory against the first error channel. In particular, we see from Fig. 10(a) that agents not initially trained on the first noise channel behave randomly and do not find reasonably good strategies to deal with the high noise level. This is because at a physical error rate of $p = 0.16$ the initial code requires many modifications before the desired logical error rate can be reached. In fact, it requires that many modifications that a random search is just not sufficient to find

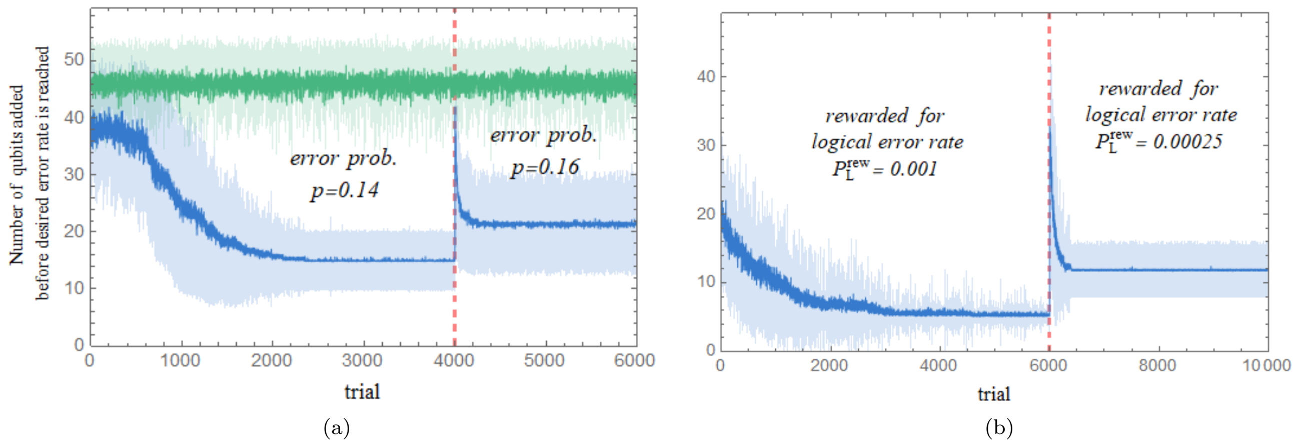


FIG. 10. This figure shows the results for two tasks as considered in Sec. VI B which explore the potential of transfer learning. The agent is tasked to optimize a QEC code w.r.t. its resource efficiency. Resource efficiency is reflected in the number of qubits added to the code before a desired logical error rate $P_{\mathrm{L}}^{\mathrm{rew}}$ is reached. The shaded area shows the standard deviation. (a) At first, the code is subject to an error channel as in Eq. (4) with $p = 0.14$ and $P_{\mathrm{L}}^{\mathrm{rew}} = 0.001$. The initial logical error rate is $P_L^{\mathrm{init}} \approx 0.019$. At trial 4,000 the error probability is increased to $p = 0.16$. The blue curve depicts the results in this scenario averaged over 60 agents. As comparison, the green line is the average over 40 agents that are directly tasked to protect against an error channel with $p = 0.16$ without having been trained on the simpler noise model. These agents encounter an initial logical error rate of $P_L^{\mathrm{init}} \approx 0.028$. (b) In this task the agent is tasked to reduce the logical error rate below $P_{\mathrm{L}}^{\mathrm{rew}} = 0.001$ while the code is subject to the error channel in Eq. (4) with $p = 0.1$. After 6,000 trials the same agent is tasked to reduce the error rate even further below $P_{\mathrm{L}}^{\mathrm{rew}} = 0.00025$. Details about parameters are given in Appendix D.

a reasonably good code in the alloted time of 6,000 trials. This showcases the benefit of transfer learning for quantum error correction.

So far, we have kept the rewarded logical error rate fixed. However, it is in general also desirable to be able to adapt to stricter thresholds if required. We therefore also wish to test if transfer learning is useful in this scenario. In the second scenario let us now verify that, in the transition from one task to another, the agent can make use of its knowledge. Here, the error channel is again fixed to that of Eq. (4) with $p = 0.1$. However, after having learned the optimal strategy from before, the agent is now tasked to decrease the logical error rate even further to a quarter of the initial value. Fig. 10(b) confirms again that the agent can indeed draw on its knowledge from the first stage of the task to find a solution for the second stage of the problem.

VII. DISCUSSION

Reinforcement learning (RL) has recently seen a great deal of success, from human-level performance in Atari games [55] to beating the world champion in the game of Go [56]. In 2017 RL was included in the list of 10 breakthrough technologies of the year in the MIT technology review [57]. As machine learning is claiming its place in the physicist's toolbox [58], RL is starting to appear in quantum physics research [11, 15, 20, 37, 59, 60].

In this work, we have presented a RL framework for adapting and optimizing quantum error correction (QEC) codes. This framework is based on a RL agent that interacts with a quantum memory (and its classical control), providing the latter with instructions for modifications of the code to lower the logical error rate below a desired threshold. For the simulations discussed here, the quantum memory is realized as a surface code to which the agent may add qubits by way of fault-tolerant code deformations [25, 26]. The agent receives a reward once the specified logical error rate is reached. The internal structure of the agent has been modeled within the Projective Simulation [32] approach to RL. The classical control system estimates the logical error rate in the simulated QEC procedure using an optimal linear-time decoder [30, 31].

Our results demonstrate that the agents learn to protect surface code quantum memories from various types of noise, including simple i.i.d. errors, but also more complicated correlated and non-isotropically distributed errors. In particular, this RL approach provides interesting solutions for nontrivial QEC code optimization problems. A particularly noteworthy feature is the ability of the agents to transfer their experience from one noise model or task to another. This suggests that such a QEC strategy based on RL can be used to switch from off-line optimization to on-line adaptive error correction. That is, provided a reasonable guess for the type of expected errors, one may start by optimizing a QEC code during a simulation. Then, the resulting code can be used as a blueprint for application in actual hardware. There, one may continue to dynamically optimize the QEC code using a RL agent previously trained on the off-line environment. The scope of our simulations has been designed specifically with such applications to current state-of-the-art quantum computing platforms [22–24] in mind. Starting with 18 initial qubits, the agents we consider are able to extend this number by up to 50 additional qubits, thus also accounting for expected near-term technology developments. A potential bottle neck for extensions to much larger qubit numbers lies in the scaling behavior of the learning complexity. There, one expects that the increase in learning time scales unfavorably with the increase in the size of the percept space (and action space, both of which depend on the number of qubits). We envisage that this problem can be circumvented through parallel processing by assigning individual agents to different surface code patches of a fixed size. All agents can then operate in parallel, allowing one to exploit a number of already available RL techniques for parallel learning that explore the use of shared experience between agents [61, 62].

Nonetheless, let us note that the presented general framework for RL-based optimization of error correction codes goes beyond the specific simulations we have performed here. That is, the approach is directly applicable also if one wishes to consider RL paradigms other than PS, QEC codes other than surface codes, or noise models other than those considered. A particular contributor to this flexibility is that both the errors and the error decoding appear as part of a black-box environment to the agent, who only ever perceives whether the logical error rate is above or below threshold. For instance, one is left with the option of choosing different decoders, and even of incorporating machine learning into the decoder itself [12–20]. Having this freedom in choosing the decoder is particularly relevant if different types of QEC codes are considered, e.g., if one allows fault-tolerant code switching through code deformation [25] or lattice surgery [63] as part of the optimization.

ACKNOWLEDGMENTS

HPN and HJB achnowledge support from the Austrian Science Fund (FWF) through the project DK-ALM: W1259-N27. HJB was also supported by the Ministerium für Wissenschaft, Forschung, und Kunst Baden-Württemberg (AZ:33-7533.-30-10/41/1). NF acknowledges support from the Austrian Science Fund (FWF) through the project P 31339-N27, the START project Y879-N27, and the joint Czech-Austrian project MultiQUEST (I 3053-N27 and GF17-33780L). HPN would like to acknowledge helpful comments from Davide Orsucci, Robert Raussendorf and Lea Trenkwalder. ND would like to thank Michael Beverland, Krysta Svore, Dave Wecker and Nathan Wiebe for useful discussions.

Appendix A: Learning in Projective Simulation

In Sec. IV we have given a brief introduction the projective simulation (PS) model for reinforcement learning (RL). Here we complete the description with a detailed account of how the h-values that govern the transition probabilities in Eq. (3) are updated. In other words, we explain how learning manifests itself in the PS model. The update rule from time step t to $t+1$ is

$$h^{(t+1)} = h^{(t)} + \lambda^{(t)} g^{(t)} + \gamma(1 - h^{(t)}) \tag{A1}$$

where the g-matrix, or so-called glow matrix, redistributes rewards λ to past experiences such that experience that lie further in the past are rewarded only by a decreasing fraction of $\lambda^{(t)}$. It contains the long-term memory of the agent. A long-term memory is crucial when rewards are delayed, i.e., not every action results into a reward. The glow matrix is updated in parallel with the h-matrix. At the beginning, g is initialized as an all-zero matrix. Every time an edge (i, j) is traversed during the decision-making process, the associated glow value g_{ij} is set to $\frac{M_i}{M_0}$ where M_i is the number of actions available for percept i and M_0 is the number of initial actions. In order to internalize how much of the reward λ is issued to past experiences in Eq. (A1) the g-matrix is also updated after each interaction with the environment. Therefore, we introduce the so-called glow parameter $\eta \in [0, 1]$ of the PS model and define an update rule as follows,

$$g^{(t+1)} = (1 - \eta) g^{(t)}. \tag{A2}$$

Besides glow, the agent is also subject to a forgetting mechanism, which is presented by the parameter γ in Eq. (A1). Effectively, the h-values are decreased by $\gamma \in [0, 1]$ (but never below 1) whenever an update is initialized. Here, the forgetting mechanism is used specifically to reinforce exploratory behavior. In order to save memory, we also introduce a deletion mechanism where unused percept clips are deleted, i.e., if the average of outgoing h-values is below a value $1 + \delta$. However, deletion can only be applied to a specific clip once a certain number of rewarded interactions τ with the environment have passed since its creation. This parameter τ is hence called immunity time. Moreover, after any given trial, all clips that have been created during said trial are deleted if no reward was given at all. Note that finding the optimal values for $\beta, \eta, \gamma, \tau$ and δ is generally a hard problem. However, it can be learned by the PS model itself [64] at the expense of longer learning times. However, hyper-parametrization is often easier and faster in PS than in comparable RL models [35].

Appendix B: Details of the environment

In Sec. V, we describe the quantum memory environment the PS agent is interacting with. The core component of the classical control (see Fig. 1) is the SQUAB algorithm [47] which simulates error correction on an arbitrary surface code. Note that error channels Eq. (4)-(6) are simulated through erasure channels in SQUAB and hence yield error rates that differ slightly from their actual Pauli counterparts. Fortunately, the use of erasure channels is well motivated (see Sec. VI A).

The remainder of the environment is dedicated to the processing of percepts and actions. In the following, we give the details of what constitutes percepts and actions. Roughly, a percept represents the code structure of the current surface code. A surface code is fully described by the graph it is defined on. A graph can be represented by its adjacency matrix A where rows represent vertices. Each vertex v is represented by a set of edges adjacent to v, $A_v = (e_1, e_2, ...)$. Edges are just labeled according to their first appearance in A. That is, $A_0 = (0, 1, ...)$. Since we require a spatial resolution of the code, each vertex v and plaquette p is assigned a label $v, p \in \{0, 1, 2, ...\}$. Edges, however, have no meaningful label. The specific code representation considered in this paper is hence a tuple of ordered adjacency matrices for the primal and dual graph. Actions have a straightforward representation that encodes Fig. 3. Each action is a vector $a = (d, v, p_1, p_2)$. $d = 0, 1$ decides whether or not this action acts on the dual lattice. v labels a vertex on the corresponding lattice and p_1, p_2 are two non-neighboring plaquettes sharing edges with v on the lattice's dual. In the spirit of our basic moves from Fig. 3(b), a labels the vertex v that is being split such that plaquettes p_1, p_2 become connected by an edge.

Appendix C: Searching the space of surface codes

Having a set of basic moves at our disposal (see Fig. 3), we are inclined to run an exhaustive search over the space of all surface codes. However, as we will see below, this problem is intractable. Assume that we start from a topological code and we search for a better code by increasing the code size. In order to preserve the topological structure of the quantum code, additional qubits are injected by adding an extra edge to the lattice or its dual as in Fig. 3. Such an action increases the number of physical qubits by one, while preserving the number of logical qubits of the code. For instance, the 3×3 square lattice admits 18 primal actions and 18 dual actions (2 per square face). A single extra qubit hence allows to reach 36 more topological codes. With two extra qubits, we already obtain 1440 new codes. Some of these codes may be identical but we treat them as distinct codes here in order to avoid the extra cost of comparing all new codes. That means that we navigate in a tree whose root is the initial code. Each node represents a topological code and the successors of a node are obtained by adding an edge as in Fig. 3.

Imagine that we want to explore a region of the space of surface codes centered at the 3×3 square lattice. We use the software SQUAB [30, 31] to perform 10,000 decoding trials for each code. This costs roughly 0.018 seconds for a single code with 20 physical qubits (using a processor 2.4 GHz Intel Core i5). SQUAB returns an estimation of the maximum likelihood decoder performance over the quantum erasure channel for a given topological code (see Sec. VI A). To simplify, we focus on the residual Z-error after correction and we ignore the X-part of the error. This is similar to the scenario considered in the main text with an error channel of the form Eq. (4). Now, we want to explore all codes up to distance r around the initial, 3×3 surface code on a square lattice (see Fig. 2). In other words, we intend to explore the full "ball" of radius r around the central node. The number of codes $C(r)$ at distance r from the root grows as follows, $C(0) = 1$, $C(1) = 36$, $C(2) = 1,440$, $C(3) = 62,893$, $C(4) = 2,961,504$. Now, adding one extra qubit increases the number of codes by about a factor of 50. Running SQUAB in a ball of radius 5 with 10,000 trials for each code would then require more than 30 days of computation. Similarly, going through all the codes obtained by adding up to 10 qubits would at least require 20 million years of CPU time. This is without even counting the cost of generating the search graph and realizing all the moves. The cost is even more discouraging when a more realistic noise model is considered increasing the simulation cost per node. Hence, it is not reasonable to consider performing a numerical simulation of all the codes within the neighborhood of an initial code in order to select the best candidate.

Exploring the whole neighborhood of a code for a sufficiently large radius is hopelessly difficult. However, we can still use random search techniques to explore and visualize the environment, i.e. the space of surface codes. To this end, we explore a subset of branches and calculate the respective logical error rates at each node. We start by evaluating all the codes at distance one from the root to ensure a minimum number of nodes. Then we continue building each successor of a code in the search tree with probability p_{expl}. Starting from a 3×3 surface code, we are able to partially explore a ball of radius 8 with exploration probability $p_{\mathrm{expl}} = 0.03$ within a few minutes. This leads us to a set of $1,230$ randomly selected codes shown in Fig. 11. We observe that moves which effectively increase the logical error rate are common. Moreover, after unnecessarily increasing the logical error rate, it is still possible to decrease the logical error rate again. In particular, it is very likely that the best codes are hidden among unexplored branches of the search tree. In order to explore in priority the most promising regions of the space of all topological codes, we chose RL.

Appendix D: Parameters

For the results that are described in Sec. V and displayed in Figs. 5–7, we deployed a PS agent as described in Sec. IV and Appendix A. There are many parameters that can be tuned. For the specific results in this

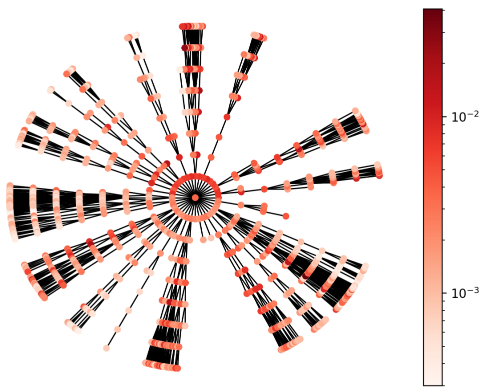


FIG. 11. Exploration of random branches of the ball of radius 8 centered at the 3×3 surface code on a square lattice. Each node corresponds to a surface code on a torus. The color shading indicates the logical error rate. This rate is estimated using SQUAB with 10,000 trials per node at an error rate of $p_Z = 0.15$ where X errors are ignored.

paper, the chosen parameters are displayed in Table. I. In particular, parameters can be tuned on-line (e.g. see Ref. [64]). Considering a task as in Fig. 10(b), it makes perfect sense to also change parameters between settings.

Figure	η	γ	δ	SQUAB iterations
5(a)	0.05	0.01	0.01	1,000,000
5(b)	0.01	0.01	0.01	1,000,000
7	0.05	0.0006	0.001	1,000,000
10(a)	0.05	0.0006	0.001	1,000,000
10(b) trials 0–6,000	0.05	0.01	0.01	1,000,000
10(b) trials 6,000–10,000	0.05	0.0005	0.001	4,000,000

TABLE I. Parameters of the PS agent (see Appendix A) and SQUAB algorithm as used for the various tasks considered in Sec. V. Two parameters remained the same for every setting: the softmax parameter $\beta = 2$, and the immunity time $\tau = 30$.

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, U.K., 2000).

[2] Vedran Dunjko, Yimin Ge, and J. Ignacio Cirac, "Computational speedups using small quantum devices," (2018), arXiv:1807.08970.

[3] Earl Campbell, Ankur Khurana, and Ashley Montanaro, "Applying quantum algorithms to constraint satisfaction problems," (2018), arXiv:1810.05582.

[4] John Preskill, "Fault-tolerant quantum computation," in *Introduction to Quantum Computation*, edited by H.-K. Lo, S. Popescu, and T. P. Spiller (World-Scientific, 1997) Chap. 8, pp. 213–269, arXiv:quant-ph/9712048.

[5] Daniel Gottesmann, *Stabilizer Codes and Quantum Error Correction*, Ph.D. thesis, Caltech (1997), arXiv:quant-ph/9705052.

[6] Barbara M. Terhal, "Quantum error correction for quantum memories," Rev. Mod. Phys. **87**, 307–346 (2015), arXiv:1302.3428.

[7] Thomas Monz, Philipp Schindler, Julio T. Barreiro, Michael Chwalla, Daniel Nigg, William A. Coish, Maximilian Harlander, Wolfgang Hänsel, Markus Hennrich, and Rainer Blatt, "14-Qubit Entanglement: Creation and Coherence," Phys. Rev. Lett. **106**, 130506 (2011), arXiv:1009.6126.

[8] Philipp Schindler, Daniel Nigg, Thomas Monz, J. T. Barreiro, Esteban Martinez, S. X. Wang, Stephan Quint, M. F. Brandl, Volckmar Nebendahl, Christian F. Roos, Michael Chwalla, M. Hennrich, and Rainer Blatt, "A quantum information processor with trapped ions," New J. Phys. **15**, 123012 (2013), arXiv:1308.3096.

[9] A. Vardy, "The intractability of computing the minimum distance of a code," IEEE T. Inform. Theory **43**, 1757–1766 (1997).

[10] Vedran Dunjko and Hans J. Briegel, "Machine learning & artificial intelligence in the quantum domain: a review of recent progress," Rep. Prog. Phys. **81**, 074001 (2018), arXiv:1709.02779.

[11] Thomas Fösel, Petru Tighineanu, Talitha Weiss, and Florian Marquardt, "Reinforcement Learning with Neural Networks for Quantum Feedback," Phys. Rev. X **8**, 031084 (2018), arXiv:1802.05267.

[12] Giacomo Torlai and Roger G. Melko, "Neural Decoder for Topological Codes," Phys. Rev. Lett. **119**, 030501 (2017), arXiv:1610.04238.

[13] Paul Baireuther, Thomas E. O'Brien, Brian Tarasinski, and Carlo W. J. Beenakker, "Machine-learning-assisted correction of correlated qubit errors in a topological code," Quantum **2**, 48 (2018), arXiv:1705.07855.

[14] Christopher Chamberland and Pooya Ronagh, "Deep neural decoders for near term fault-tolerant experiments," Quantum Science and Technology **3**, 044002 (2018), arXiv:1802.06441.

[15] Ryan Sweke, Markus S. Kesselring, Evert P. L. van Nieuwenburg, and Jens Eisert, "Reinforcement learning decoders for fault-tolerant quantum computation," (2018), arXiv:1810.07207.

[16] Paul Baireuther, M. D. Caio, B. Criger, Carlo W. J. Beenakker, and Thomas E. O'Brien, "Neural network decoder for topological color codes with circuit level noise," (2018), arXiv:1804.02926.

[17] Xiaotong Ni, "Neural network decoders for large-distance 2d toric codes," (2018), arXiv:1809.06640.

[18] Nishad Maskara, Aleksander Kubica, and Tomas Jochym-O'Connor, "Advantages of versatile neural-network decoding for topological codes," (2018), arXiv:1802.08680.

[19] Ye-Hua Liu and David Poulin, "Quantum error correction for the toric code using deep reinforcement learning," (2018), arXiv:1811.12338.

[20] Philip Andreasson, Joel Johansson, Simon Liljestrand, and Mats Granath, "Neural belief-propagation decoders for quantum error-correcting codes," (2018), arXiv:1811.07835.

[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (MIT press, Cambridge, 1998).

[22] Nicolai Friis, Oliver Marty, Christine Maier, Cornelius Hempel, Milan Holzäpfel, Petar Jurcevic, Martin B. Plenio, Marcus Huber, Christian Roos, Rainer Blatt, and Ben Lanyon, "Observation of Entangled States of a Fully Controlled 20-Qubit System," Phys. Rev. X **8**, 021012 (2018), arXiv:1711.11092.

[23] J. Zhang, G. Pagano, P. W. Hess, A. Kyprianidis, P. Becker, H. Kaplan, A. V. Gorshkov, Z.-X. Gong, and C. Monroe, "Observation of a many-body dynamical phase transition with a 53-qubit quantum simulator," Nature **551**, 601–604 (2017), arXiv:1708.01044.

[24] Hannes Bernien, Sylvain Schwartz, Alexander Keesling, Harry Levine, Ahmed Omran, Hannes Pichler, Soonwon Choi, Alexander S. Zibrov, Manuel Endres, Markus Greiner, Vladan Vuletić, and Mikhail D. Lukin, "Probing many-body dynamics on a 51-atom quantum simulator," Nature **551**, 579–584 (2017), arXiv:1707.04344.

[25] Héctor Bombín and Miguel Angel Martin-Delgado, "Quantum measurements and gates by code deformation," J. Phys. A: Math. Theor. **42**, 095302 (2009), arXiv:0704.2540.

[26] Sergey Bravyi and Alexei Kitaev, "Quantum codes on a lattice with boundary," (1998), arXiv:quant-ph/9811052.

[27] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill, "Topological quantum memory," J. Math. Phys. **43**, 4452–4505 (2002), arXiv:quant-ph/0110143.

[28] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland, "Surface codes: Towards practical large-scale quantum computation," Phys. Rev. A **86**, 032324 (2012), arXiv:1208.0928.

[29] Keisuke Fujii and Yuuki Tokunaga, "Error and loss tolerances of surface codes with general lattice structures," Phys. Rev. A **86**, 020303 (2012), arXiv:1202.2743.

[30] Nicolas Delfosse, Pavithran Iyer, and David Poulin, "A linear-time benchmarking tool for generalized surface codes," (2016), arXiv:1611.04256.

[31] Delfosse, Nicolas and Iyer, Pavithran, "Squab – a fast benchmarking software for surface quantum computing architectures," (2016), [Online; accessed 20-December-2018].

[32] Hans J. Briegel and Gemma De las Cuevas, "Projective simulation for artificial intelligence," Sci. Rep. **7**, 400 (2012), arXiv:1104.3787.

[33] Julian Mautner, Adi Makmal, Daniel Manzano, Markus Tiersch, and Hans J. Briegel, "Projective Simulation for Classical Learning Agents: A Comprehensive Investigation," New Gener. Comput. **33**, 69–114 (2015), arXiv:1305.1578.

[34] Alexey A. Melnikov, Adi Makmal, Vedran Dunjko, and Hans J. Briegel, "Projective simulation with generalization," Sci. Rep. **7**, 14430 (2017), arXiv:1504.02247.

[35] A. A. Melnikov, A. Makmal, and H. J. Briegel, "Benchmarking projective simulation in navigation problems," IEEE Access **6**, 64639–64648 (2018).

[36] Simon Hangl, Emre Ugur, Sandor Szedmak, and Justus Piater, "Robotic playing for hierarchical complex skill learning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016) pp. 2799–2804, arXiv:1603.00794.

[37] Alexey A. Melnikov, Hendrik Poulsen Nautrup, Mario Krenn, Vedran Dunjko, Markus Tiersch, Anton Zeilinger, and Hans J. Briegel, "Active learning machine learns to create new quantum experiments," Proc. Natl. Acad. Sci. U.S.A. **115**, 1221–1226 (2018), arXiv:1706.00868.

[38] Sebastian Thrun, "Is learning the n-th thing any easier than learning the first?" in *Advances in Neural Information Processing Systems 8*, edited by D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo (MIT Press, 1996) pp. 640–646.

[39] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang, "A survey of transfer learning," Journal of Big Data **3**, 9 (2016).

[40] Nicolas Delfosse and Gilles Zémor, "Linear-Time Maximum Likelihood Decoding of Surface Codes over the Quantum Erasure Channel," (2017), arXiv:1703.01517.

[41] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, 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 John M. Martinis, "Superconducting quantum circuits at the surface code threshold for fault tolerance," Nature **508**, 500–503 (2014), arXiv:1402.4848.

[42] Torsten Karzig, Christina Knapp, Roman M. Lutchyn, Parsa Bonderson, Matthew B. Hastings, Chetan Nayak, Jason Alicea, Karsten Flensberg, Stephan Plugge, Yuval Oreg, Charles M. Marcus, and Michael H. Freedman, "Scalable designs for quasiparticle-poisoning-protected topological quantum computation with majorana zero modes," Phys. Rev. B **95**, 235305 (2017), arXiv:1610.05289.

[43] Charles H. Bennett, David P. DiVincenzo, and John A. Smolin, "Capacities of quantum erasure channels," Phys. Rev. Lett. **78**, 3217–3220 (1997), arXiv:quant-ph/9701015.

[44] M. Grassl, Th. Beth, and T. Pellizzari, "Codes for the quantum erasure channel," Phys. Rev. A **56**, 33–38 (1997), arXiv:quant-ph/9610042.

[45] J. M. Amini, H. Uys, J. H. Wesenberg, S. Seidelin, J. Britton, J. J. Bollinger, D. Leibfried, C. Ospelkaus, A. P. VanDevender, and D. J. Wineland, "Toward scalable ion traps for quantum information processing," New J. Phys. **12**, 033031 (2010), arXiv:0909.2464.

[46] R. Bowler, J. Gaebler, Y. Lin, T. R. Tan, D. Hanneke, J. D. Jost, J. P. Home, D. Leibfried, and D. J. Wineland, "Coherent Diabatic Ion Transport and Separation in a Multizone Trap Array," Phys. Rev. Lett. **109**, 080502 (2012), arXiv:1206.0780.

[47] Nicolas Delfosse and Naomi H. Nickerson, "Almost-linear time decoding algorithm for topological codes," (2017), arXiv:1709.06218.

[48] Richard Cleve and Daniel Gottesman, "Efficient computations of encodings for quantum error correction," Phys. Rev. A **56**, 76–82 (1997).

[49] Scott Aaronson and Daniel Gottesman, "Improved simulation of stabilizer circuits," Phys. Rev. A **70**, 052328 (2004).

[50] David P. DiVincenzo and Peter W. Shor, "Fault-tolerant error correction with efficient quantum codes," Phys. Rev. Lett. **77**, 3260–3263 (1996).

[51] Simon Anders and Hans J. Briegel, "Fast simulation of stabilizer circuits using a graph-state representation," Phys. Rev. A **73**, 022334 (2006).

[52] Tatiana Tommasi and Barbara Caputo, "The more you know, the less you learn: from knowledge transfer to one-

shot learning of object categories," in *British Machine Vision Conference* (2009).

[53] T. Tommasi, F. Orabona, and B. Caputo, "Safety in numbers: Learning categories from few examples with multi model knowledge transfer," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2010) pp. 3081–3088.

[54] Y. Aytar and A. Zisserman, "Tabula rasa: Model transfer for object category detection," in *2011 International Conference on Computer Vision* (2011) pp. 2252–2259.

[55] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis, "Human-level control through deep reinforcement learning," Nature **518**, 529–533 (2015).

[56] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis, "Mastering the game of Go without human knowledge," Nature **550**, 354–359 (2017).

[57] Knight, Will, "Reinforcement learning – by experimenting, computers are figuring out how to do things that no programmer could teach them." (2017), [Online; accessed 20-December-2018].

[58] Lenka Zdeborová, "New tool in the box," Nat. Phys. **13**, 420–421 (2017).

[59] Raban Iten, Tony Metger, Henrik Wilming, Lidia del Rio, and Renato Renner, "Discovering physical concepts with neural networks," (2018), arXiv:1807.10300.

[60] Moritz August and José Miguel Hernández-Lobato, "Taking gradients through experiments: Lstms and memory proximal policy optimization for black-box quantum control," (2018), arXiv:1802.04063.

[61] Matthew R. Kretchmar, "Parallel reinforcement learning," in *The 6th World Conference on Systematics, Cybernetics, and Informatics* (2002) pp. 165–170.

[62] Enda Barrett, Jim Duggan, and Enda Howley, "A parallel framework for bayesian reinforcement learning," Connection Science **26**, 7–23 (2014).

[63] Hendrik Poulsen Nautrup, Nicolai Friis, and Hans J. Briegel, "Fault-tolerant interface between quantum memories and quantum processors," Nat. Commun. **8**, 1321 (2017), arXiv:1609.08062.

[64] Adi Makmal, Alexey A. Melnikov, Vedran Dunjko, and Hans J. Briegel, "Meta-learning within Projective Simulation," IEEE Access **4**, 2110–2122 (2016), arXiv:1602.08017.