Quantum Circuit Discovery for Fault-Tolerant Logical State Preparation with Reinforcement Learning

Remmy Zen,[1,][*] Jan Olle,[1] Luis Colmenarez,[2,3] Matteo Puviani,[1] Markus Müller,[2,3] and Florian Marquardt[1,4]

[1]*Max Planck Institute for the Science of Light, Staudtstraße 2, 91058 Erlangen, Germany*
[2]*Institute for Quantum Information, RWTH Aachen University, 52056 Aachen, Germany*
[3]*Peter Grünberg Institute, Theoretical Nanoelectronics,*
Forschungszentrum Jülich, 52425 Jülich, Germany
[4]*Department of Physics, Friedrich-Alexander Universität Erlangen-Nürnberg, Staudtstraße 5, 91058 Erlangen, Germany*
(Dated: May 20, 2024)

The realization of large-scale quantum computers requires not only quantum error correction (QEC) but also fault-tolerant operations to handle errors that propagate into harmful errors. Recently, flag-based protocols have been introduced that use ancillary qubits to flag harmful errors. However, there is no clear recipe for finding a fault-tolerant quantum circuit with flag-based protocols, especially when we consider hardware constraints, such as qubit connectivity and available gate set. In this work, we propose and explore reinforcement learning (RL) to automatically discover compact and hardware-adapted fault-tolerant quantum circuits. We show that in the task of fault-tolerant logical state preparation, RL discovers circuits with fewer gates and ancillary qubits than published results without and with hardware constraints of up to 15 physical qubits. Furthermore, RL allows for straightforward exploration of different qubit connectivities and the use of transfer learning to accelerate the discovery. More generally, our work opens the door towards the use of RL for the discovery of fault-tolerant quantum circuits for addressing tasks beyond state preparation, including magic state preparation, logical gate synthesis, or syndrome measurement.

I. INTRODUCTION

Quantum systems are highly fragile due to their susceptibility to errors caused by decoherence. Furthermore, quantum operations are imperfect and error-prone. Therefore, in order to harness quantum systems for computation, the error rates must be significantly reduced. Quantum error correction (QEC) is essential to protect quantum information from these errors, allowing us to perform complex and reliable computations [1, 2]. The basic idea behind QEC is to encode logical qubits into multiple noisy physical qubits in such a way that we can detect and correct errors without destroying the logical state. Although the implementation of QEC is a challenging task, recently we have seen several experimental breakthroughs of QEC with different quantum computing platforms [3–9], first quantum circuits carried out on up to 48 logical qubits [10, 11] and crossing the break-even point of beneficial QEC [12–14].

QEC operations are often expressed using a sequence of quantum gates that forms a quantum circuit. However, these gates are faulty and multi-qubit gates proliferate the errors, compromising the scalability of QEC. In general, the more gates, the more errors are introduced, making QEC less effective [15–17]. Therefore, we want to minimize the number of possible faulty operations that can lead to harmful errors: this is achieved by designing *fault-tolerant* (FT) circuits [18]. In FT circuits, all faults (gates, measurements, errors, resets) that our QEC code cannot correct become less likely to occur below a specific threshold as the distance of the code increases (see

Sec. II C for more details). In consequence, only by using FT schemes we can ensure systematic improvement in correction as the size of the code scales. Therefore, FT is of paramount importance in making scalable quantum computers [2, 15–17]. Several classes of FT protocols have been proposed [6, 18–22]. Among the first ones was Shor-type error correction [23] which relies on additional GHZ states and repeated measurements to check for errors. Another scheme is Steane-type error correction, which uses additional logical qubits to detect errors [24–26]. Both approaches suffer from a large qubit overhead. Recently, *flag fault-tolerant error correction* [20, 27–31] was introduced as a way to achieve fault-tolerant protocols with a minimal number of ancilla qubits, e.g. sometimes requiring only one extra qubit. For instance, in the specific case of preparing a state fault-tolerantly, a flag fault-tolerant protocol uses a *verification circuit* after the encoding circuit that utilizes a few extra ancilla qubits, known as *flag qubits*, to flag harmful errors while keeping the logical state intact. There are already examples of flag verification circuits in state preparation on several QEC codes [32–36]. They have also been shown to be effective in reducing logical error rates in experimental realizations [3, 6, 7, 37–39].

Despite their success, flag-based protocols are typically handcrafted. Furthermore, flag-based protocols have so far been implemented in devices with all-to-all qubit connectivity. A *transpilation* process [40–42] can be applied to the circuit to respect the qubit connectivity and gate set, but it will generally make it non-FT. In other words, the automatic *compilation* [43–47] of FT circuits has not been widely explored yet.

In this work, we present a novel approach based on reinforcement learning (RL) to automatically discover

[*] remmy.zen@mpl.mpg.de
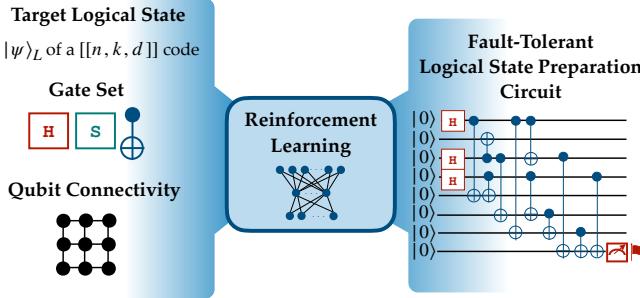
FIG. 1. Discovery of fault-tolerant logical state preparation circuit with reinforcement learning (RL). Given the target logical state $|\psi\rangle_L$ of a specified $[[n, k, d]]$ code, a gate set, and a qubit connectivity, we use RL to automatically discover circuits for preparing $|\psi\rangle_L$ fault-tolerantly with flag qubits.

fault-tolerant quantum circuits for QEC. RL is an approach in which an agent learns to make decisions by interacting with an environment in order to maximize its reward through guided trial and error. We apply our method to the task of logical state preparation. Specifically, our approach is based on the automatic discovery of quantum circuits that fault-tolerantly prepare the logical states of a given QEC code under hardware constraints. That is, we can constrain the qubit connectivity and the gate set based on the quantum platforms of interest, as shown in Fig. 1. The RL agent needs to find an optimal strategy by applying a discrete gate at each step, guided by reward signals.

Recently, reinforcement learning (RL) [48] has emerged as a useful tool for solving various problems in quantum technologies [49]. It has been applied to quantum error correction [50–55], quantum control and state preparation [56–61], and quantum compilation [62–68] among many others. It has been shown that RL stands out in quantum state preparation when the gates applied by the agent are discrete [56]. Furthermore, RL is suitable for our task because it can be formulated as a goal-oriented task that is specified in the reward signals. In addition, we find that RL is capable of efficiently navigating large and complex quantum circuit spaces. Finally, RL enables efficient and flexible automated discovery through transfer learning, i.e., reusing trained RL agents for similar but different quantum circuit problem instances. This is not possible, for example, in the method proposed in Ref. [69], where finding a fault-tolerant quantum circuit is framed as a Satisfiability Modulo Theory (SMT) problem.

We test our method on several QEC codes, including the 5-qubit perfect code, the 7-qubit Steane code, the 9-qubit Shor code, and the 15-qubit Reed-Muller code. Our first RL approach is to separate the task of finding a FT logical qubit encoding protocol into a logical state preparation task followed by a verification circuit synthesis task. Individually, the RL method for each task already produces quantum circuits that have better or similar performance to existing circuits. More interest-

ingly, by integrating the logical state preparation and verification circuit synthesis tasks, a single RL agent can directly prepare FT logical states and is able to outperform all other available approaches. Thus, this work establishes RL as a viable approach for FT quantum circuit synthesis tasks that go beyond the preparation of logical states.

The paper is organized as follows. In Sec. II, we give a brief background on FT QEC and RL. In Sec. III, we describe our general reinforcement learning framework for fault-tolerant logical state preparation. The preparation of a FT logical state can be divided into two successive tasks: the preparation of the logical state, described in Sec. IV, followed by the synthesis of the verification circuit, described in Sec. V. In Sec. VI, we go beyond the separation of tasks and present our main integrated approach, where we directly prepare FT logical states. In Sec. VII we summarize our work and discuss further extensions.

II. BACKGROUND

A. Quantum Error Correction

Here, we briefly review basic concepts from stabilizer quantum error correcting (QEC) codes and introduce the notation that will be used in this paper. Readers familiar with these concepts are invited to skip to Sec. II B.

The main idea of quantum error correction is to introduce redundancy by encoding k logical qubits into $n > k$ noisy physical qubits. In this work, we focus on a specific type of QEC codes called *stabilizer codes* [70]. Given the Pauli group of n qubits, the set of stabilizers S is a subgroup such that all elements of S commute with each other and $-I \notin S$. If S is generated by the set $G = \langle g_1, \ldots, g_{n-k} \rangle$, then the code space corresponds to the joint $+1$ subspace of all generators g_i, hosting logical quantum states $|\psi\rangle$, for which $g_i|\psi\rangle = |\psi\rangle$ for all generators. Within the code space, each code word can be transformed into one another using the *logical operators* Z_L^i, X_L^i, with $i = 1, ..., k$, where Z_L^i and X_L^i commute with all elements of the stabilizer group and satisfy $[Z_L^i, X_L^j] = 2Z_L^i X_L^j \delta_{ij}$, where δ_{ij} is the Kronecker delta. For instance, for the case of a QEC code hosting a single logical qubit, $k = 1$, $Z_L^1|0\rangle_L = |0\rangle_L$, $Z_L^1|1\rangle_L = -|1\rangle_L$ and $X_L^1|0\rangle_L = |1\rangle_L$. Thus, once S is chosen, the choice of logical operators fixes the codewords $|0\rangle_L$ and $|1\rangle_L$ and all their linear combinations.

The *weight* of a Pauli operator is the number of non-identity components within that operator. The minimum weight among all possible choices of logical operators defines the *distance* d of the QEC code. A QEC code is able to detect $d-1$ errors and correct $\lfloor (d-1)/2 \rfloor$ errors. A distance d QEC code encoding k logical qubits into n physical qubits is denoted as $[[n, k, d]]$.

A QEC code can be defined solely by its stabilizer generators g_i. When the stabilizer generators consist of ei-

ther X or Z Pauli matrices, such that they can be related to two independent classical codes C_X and C_Z for the X and Z stabilizers, they are called *Calderbank-Shor-Steane (CSS) codes* [71, 72]. Due to their simplicity and connection to classical codes, CSS codes are at the frontier of theoretical and practical implementations of QEC [3, 13, 37]. Two famous examples of CSS codes are the surface/toric code [17, 73] and the color code [74, 75]. In our work we consider the search for FT and non-FT encoding circuits for several CSS codes (including color codes) and the 5-qubit code, which is a non-CSS code (see Appendix F for the code definitions).

B. Logical Qubit Encoding Circuit

Once a QEC code and the logical operators are chosen, the next step is to find a way to encode the desired logical states. For stabilizer codes, one approach is to measure the stabilizers and apply conditional local operations that bring the state back into the code space [5, 7]. This approach relies on stabilizer measurements, which has the disadvantage of being susceptible to measurement errors, and forces repeated measurements according to the code distance to ensure FT, resulting in a large gate count. An alternative approach is to find a unitary circuit U that encodes such information using the given code [3, 4, 6, 37]. For instance, encoding a logical zero $|0\rangle_L$ implies finding a circuit that performs the task $|0\rangle_L = U|0\rangle^{\otimes n}$. Unlike stabilizer measurement encodings, unitary encodings avoid repeated stabilizer measurements, potentially reducing the number of gates. Importantly, even after choosing a QEC code and codeword, there is no unique recipe for finding an encoding unitary U.

NISQ devices often have specific constraints, such as limited qubit connectivity and *native gate set* availability. To fulfill these constraints, a transpilation process is commonly applied to the circuit. The whole procedure typically involves mapping the qubits in the circuit to physical qubits, routing the qubits based on the connectivity by inserting swap gates, decomposing gates into native gates, and optimizing the final circuit [40–42]. Since the procedure involves inserting and decomposing gates, this process will, in general, increase the size of the circuit.

Due to their simplicity and relevance, we restrict ourselves to logical Pauli eigenstates only. Thus, we can focus only on Clifford circuits, where the logical state $|\psi\rangle = U|0\rangle^{\otimes n}$ is always determined by its *stabilizer tableau* [76]. In particular, a tableau of a single logical codeword contains the $n-k$ stabilizer generators and the k logical operators, which can be represented as a binary matrix that scales quadratically with respect to n. Appendix A shows more details on the tableau representation. While different tableaus may represent the same state, their canonical form [70] remains the same. A *canonical tableau* can be obtained by applying Gaussian elimination to the tableau [76]. This means that differ-

ent encoding circuits preparing the same logical state will have the same representation. We will use this representation later as an input to the RL agent. The canonical representation helps the RL agent to learn more effectively and efficiently by reducing complexity and ensuring consistency of the input space.

It has been proven that Clifford circuits can be efficiently simulated using classical computers [76]. Despite its simplicity, finding a compact circuit is still not trivial [77, 78]. Several methods have been proposed to prepare arbitrary stabilizer states [76, 79–82]. Some methods have also been developed specifically for the preparation of logical states of stabilizer QEC codes [83–87]. However, these techniques generally do not include any hardware constraints, nor do they output fault-tolerant quantum circuits, the latter of which we will focus on next.

C. Fault-Tolerant State Preparation

In practice, quantum gates are faulty and thus introduce errors in state preparation. A simple but effective model for gate failures is to consider the perfect gate to be applied only with probability $1 - p$, where p is the probability that a fault occurs when the gate is applied. In this work, we consider any fault consisting of bit flips (X Pauli), phase flips (Z Pauli), or both (Y Pauli). Therefore, single qubit gates have 3 error generators $\mathcal{E} = \{\sigma_k\}/I$ and two qubit gates have 15 error generators $\mathcal{E} = \{\sigma_k \otimes \sigma_m\}/(I \otimes I)$, where $k, m = 0, 1, 2, 3$ denotes the Pauli matrices including the identity. More formally, the errors introduced by the gates are modeled by introducing a depolarizing channel after the gates:

$$G\rho G^{\dagger} = (1-p)G\rho G^{\dagger} + \sum_{E \in \mathcal{E}} \frac{p}{|\mathcal{E}|} EG\rho G^{\dagger}E, \qquad (1)$$

where G is the ideal gate, p is the probability of having a gate error, and $|\mathcal{E}|$ is the number of elements in the set of all error generators $\mathcal{E}$. This is the standard modeling of gate errors, often referred to as *circuit-level noise* [3, 6, 8, 84].

An error E can be propagated through the circuit in such a way that a unitary $U_E = \tilde{E}U$ can always be written as the error-free U followed by the propagated error $\tilde{E}$. For Clifford unitaries, $\tilde{E}$ remains a single Pauli error obtained by propagating E through the individual gates one by one [84]. There are two classes of errors that we consider according to their propagated version: (i) $\tilde{E}$ is a member of the stabilizer group, thus acting trivially on the stabilizer states, or its weight is small enough that it can be removed by QEC, in which case, we say the error is *tolerable*. (ii) Its weight is large enough that it cannot be corrected, causing a logical failure after a QEC cycle. We call such errors *harmful*.

In practice, any circuit that is not carefully designed will have components whose failures lead to harmful er-

rors. For example, even a single gate failure with probability p can lead to a logical error. Therefore, increasing the code distance of the QEC code would not suppress the logical error rate, because there would always be uncorrectable events with probability p. Formally, for a code of distance d able to correct errors of weight $t = \lfloor (d-1)/2 \rfloor$, the logical error rate in a FT architecture p_L scales as $p_L \sim p^{t+1}$ for p below the threshold [15, 16, 23], which ensures an ever decreasing logical error rate when increasing the size (number of physical qubits) of the QEC code. In contrast, if a harmful error occurs with probability p, the expected gain from QEC is lost, no matter how large d is. In other words, all error events with probability p^α, $\alpha < (d+1)/2$ should be *tolerable* in the sense that they are corrected after QEC cycles. A circuit or component that fulfills the latter condition is called *fault-tolerant* (FT) [20, 21, 28, 29, 69, 88–91]. As an example, let us consider a code with $d = 3$ that corrects any single qubit error. Some gate failures in this code can produce weight-two harmful errors. Therefore, $p_L \sim p$ if the circuit design is non-FT. If the encoding circuit is made fault-tolerant, then all errors coming from a single gate failure become tolerable, and only errors coming from two gate failures are harmful, hence $p_L \sim p^2$.

There is no unique way to render a circuit FT [3, 21, 28, 69, 89, 92, 93]. Recently, *flag verification circuits* [20, 28, 29] have been proposed for turning non-FT circuits into FT ones. The flag verification procedure is based on coupling additional ancilla flag qubits to the main register in such a way that the error-free state is unperturbed and the flag qubit(s) always have the same measurement outcome. When faults that lead to harmful errors occur in any component of the circuit, the flag qubit(s) are triggered, i.e., flip the measurement outcome of the flag ancilla qubit(s). Thus, harmful errors are flagged out by the flag qubit, allowing us to do postselection on the ancilla measurement outcomes. One can then apply a repeat-until-success mechanism (rejecting outcomes with triggered flag qubits and accepting outcomes without triggered flag qubits) and apply QEC to remove the tolerable errors.

D. Reinforcement Learning

Reinforcement learning (RL) [48] aims to train an agent to take an optimal set of actions in an environment (here a simulation of our physical system). This goal is achieved by maximizing the expected returns or the cumulative rewards via a guided trial-and-error approach. In this work, we focus on model-free reinforcement learning, where the agent does not know about the model of the environment. Formally, an RL agent observes the state of the RL environment s_t, applies a discrete action a_t at time step t that changes the state of the environment from s_t to s_{t+1}, and receives an instantaneous reward r_t. An *episode* is a trajectory of states and actions $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$ from the initial state s_0 to the terminal state s_T. An RL agent learns a *policy* function π_θ parameterized by θ, which maps each state of the environment to a probability distribution over all possible actions. $\pi_\theta(a_t|s_t)$ gives the probability of applying action a_t for a given state s_t of the environment. The RL agent is trained to maximize the expected returns (cumulative reward) over multiple episodes $\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} r_t]$.

Policy gradient methods [94] optimize the objective function $\mathcal{J}(\theta)$ with gradient ascent. In this work, we use a deep reinforcement learning algorithm where a deep neural network is used to compute π_θ, where θ corresponds to the weights and biases of a neural network. We use a state-of-the-art variant of policy gradient methods called Proximal Policy Optimization (PPO) [95]. In PPO, we use two networks: an actor and a critic network. The former determines the action taken by the agent, while the latter measures the quality of the action taken by the agent. Both networks take the representation of the observation as input. The actor network outputs the probability of taking each discrete action, while the value network outputs a value that corresponds to the expectation value of the cumulative reward. During training, the parameter θ of the networks is updated in such a way that the objective is satisfied.

## III. REINFORCEMENT LEARNING FRAMEWORK FOR QUANTUM CIRCUIT DISCOVERY
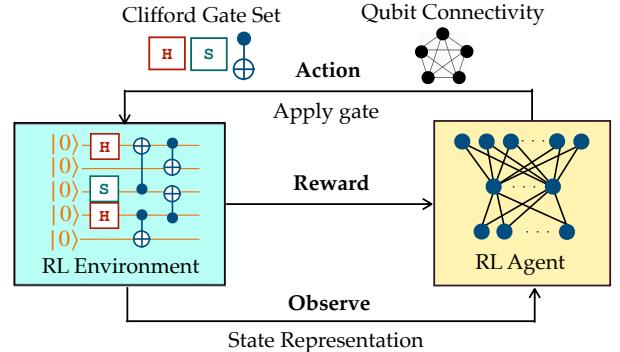
FIG. 2. The general RL framework in this work. The circuit is the environment, where its state is represented by its stabilizer canonical tableau. At each step, the RL agent observes the environment and applies a discrete Clifford gate as an action from the specified available gate set (e.g., the Hadamard gate H, the phase gate S, and the CNOT gate), taking into account qubit connectivity constraints. Subsequently, the agent receives a reward depending on the given task and the quality of the proposed circuit.

Here we first introduce the general RL framework as shown in Fig. 2. In this work, an RL agent is trained to output circuits (suggesting a sequence of gates) for a given task (i.e. logical state preparation, verification circuit synthesis, or integrated fault-tolerant logical state

preparation). At each step, the RL agent observes the state of a quantum circuit and applies a discrete Clifford gate to the quantum circuit as an action. A trajectory stops when the number of gates is greater than a preset maximum number (counting as a failure) or when it reaches the success criteria defined by the task. We assume that all physical qubits in the circuit are initialized in the $|0\rangle$ state. The hardware constraints, such as the set of available Clifford gates and the qubit connectivity of the device considered, determine the set of possible actions that the agent can take. The reward is then given according to how well the quantum circuit proposed by the agent fulfills the task, which will be explained in the following sections.

We must then choose a representation of the RL agent's observation. The most common representation is to directly observe the quantum circuit [96, 97] or to observe the state vector of the state that the circuit represents [56, 58, 59, 98]. However, multiple quantum circuits could represent the same state, and the state vector representation scales exponentially with n. Since we are focusing on stabilizer codes, we can use the stabilizer tableau of the circuit as a representation of the state of the environment. Even better, we can use the canonical tableau as the representation, so that different circuits producing the same output state have the same representation. This representation scales quadratically with n.

Although the state representation is polynomial in n, a brute-force search of the circuit scales exponentially with the number of gates L. Suppose we choose a gate set consisting of G_1 one-qubit gates and G_2 two-qubit gates with all-to-all qubit connectivity. At each step, the agent must decide over $nG_1 + (n^2 - n)G_2$ possible actions, which scales quadratically with n. Furthermore, if we assume that a circuit has L gates, then the space of all possible solutions grows exponentially as $(nG_1 + (n^2 - n)G_2)^L$, making search algorithms infeasible.

As a side note, instead of using a discrete Clifford gate set, one can also use a continuous gate set with a parameterized circuit and use a variational approach as in Ref. [99]. However, in a variational approach, the state can no longer be efficiently described within the stabilizer formalism. Furthermore, one has to design an ansatz and optimize the parameters, which generally does not scale well due to barren plateaus [100].

We use the PUREJAXRL library [101] for the implementation of the PPO algorithm, which is written with the JAX [102] library to allow very fast parallel training on a GPU. We then implement the environment for each task using JAX. This means that the simulation of the Clifford circuits and the computation of the rewards run very fast in parallel on the GPU. Therefore, we train multiple agents in parallel and each agent is trained on multiple environments also in parallel. The code is available online [103]. The details of the hyperparameters used and the training process are described in Appendix B.
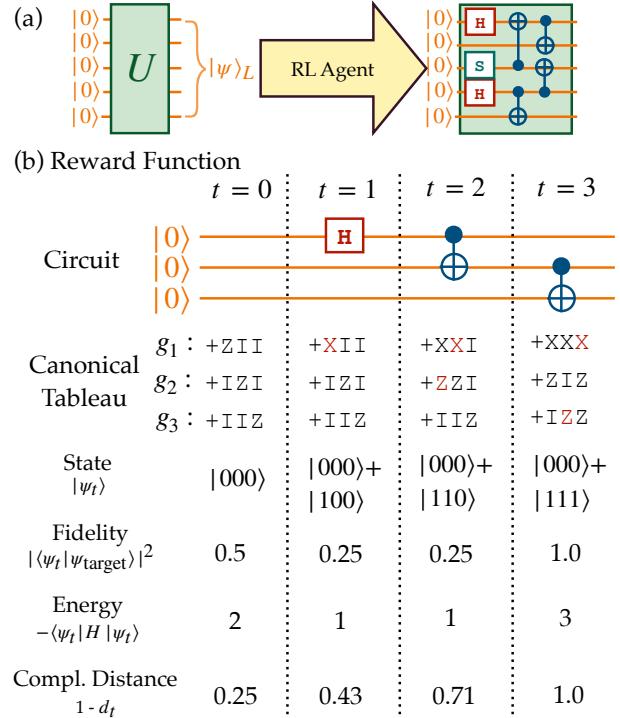

FIG. 3. Description and reward function for the logical state preparation task. (a) The logical state preparation task outputs a circuit U that prepares a target logical state $|\psi\rangle_L$ of a $[[n, k, d]]$ code. (b) The preparation of the state $|\psi_{\text{target}}\rangle = |000\rangle + |111\rangle$ (normalization factors are not shown for simplicity) from the initial state $|\psi_0\rangle = |000\rangle$. We show the value of the three possible functions at each time step t for the reward: fidelity $|\langle\psi_t|\psi_{\text{target}}\rangle|^2$, energy $\sum_i \langle\psi_t|H|\psi_t\rangle$ used in [99], and our proposed complementary tableau distance $1 - d_t$. In this case, the proposed complementary tableau distance is monotonically increasing, which is easier for RL algorithms to learn compared to the other functions.

IV. LOGICAL STATE PREPARATION

A. Task Description and Reward Function

The goal of the logical state preparation task is to find a circuit U that prepares the target stabilizer state (see Fig. 3(a)). The task requirement is the canonical tableau T_{target} of the target stabilizer state $|\psi_{\text{target}}\rangle$. Note that although in this paper we focus on preparing logical states of a stabilizer code, this task is general enough to prepare any stabilizer state.

In any RL application, it is of utmost importance to design a good reward function according to the goal. A natural choice of the reward function is the *fidelity* of the state [56, 57, 59, 98, 104, 105]. For a given state at time step t, $|\psi_t\rangle$, the fidelity can be computed as $|\langle\psi_t|\psi_{\text{target}}\rangle|^2$, however it suffers from the sparse reward problem [48]. As an illustration, consider preparing $|\psi_{\text{target}}\rangle = |111\rangle$ from an initial state of $|\psi_0\rangle = |000\rangle$. The agent would have to apply the Pauli X gate to every qubit

that changes the state from $|000\rangle$ to $|100\rangle$ to $|110\rangle$ to $|111\rangle$. However, the fidelity value only changes on the last step, since $|\langle 000|111\rangle|^2 = |\langle 100|111\rangle|^2 = |\langle 110|111\rangle|^2 = 0$. This makes the RL agent harder to train because it does not get immediate feedback.

Since we are preparing a stabilizer state, there are seemingly better rewards that we can use, but there are still drawbacks. In Ref. [99], finding a logical state of a stabilizer code is framed as finding the ground state of a Hamiltonian $H = -\sum_{i=1}^{n-k} g_i - \sum_{j=1}^{k} O_L^j$, where g are the generators of the target state and O_L are the logical operators. This then allows one to compute the *energy* as $E = \sum_i \langle \psi_t | H | \psi_t \rangle$. If $|\psi_t\rangle = |\psi_{\text{target}}\rangle$, then the ground state energy $E_0 = -n$. Ref. [99] used E as a cost function for the variational optimization of a parameterized circuit. In our case, we use $-E$ instead, since we want to maximize the cumulative reward. Although the computation of this function scales linearly with n, it still suffers from the sparse reward problem. One can see that there are only $2n$ possible discrete energy values ranging from $-n$ to n.

We introduce another measure that does not suffer from the sparse reward problem, and the computation of its value does not scale exponentially. We refer to it as the tableau distance d_t, which is the distance between the tableau describing the output state of the currently proposed quantum circuit and the tableau of the target state. We convert the tableaus into binary vectors and measure the binary distance d_t between the two. Here, we use the Jaccard distance (see discussion in Appendix C). We normalize the d_t so that it ranges from 0 to 1 and we use the *complementary tableau distance* $1 - d_t$ since the training of RL maximizes the cumulative reward.

Fig. 3(b) illustrates how the three possible functions (fidelity, energy, and complementary tableau distance) for the reward change for preparing $|000\rangle + |111\rangle$ (normalization factors are not shown for simplicity) from $|000\rangle$. We see that in this case, unlike the other functions, the proposed complementary tableau distance $1 - d_t$ always increases when gates are applied, giving good feedback to the RL agent. We can also see that from $t = 1$ to $t = 2$, applying the correct gate does not change the fidelity and energy functions, which is not a good feedback to the RL agent. This is not the case for $1 - d_t$. Our empirical numerical experiments also showed that using our proposed complementary tableau distance function as a reward leads to faster convergence of the training of the RL agent compared to using the rewards based on the fidelity and the energy.

Finally, one can give the reward only at the last time step (e.g. $r_t = 1 - d_L$ at $t = L$, otherwise $r_t = 0$). However, this means that the agent does not receive immediate feedback after performing an action. Instead, we use the reward shaping technique [48] by giving a small intermediate value at each step so that the training converges faster. Therefore, at each time step t, we give the difference of the complementary tableau distance between t

and $t - 1$, or more formally,

$$r_t = d_{t-1} - d_t. \tag{2}$$

In this case, the cumulative reward $\sum_{t=0}^{L} r_t$ is still $1 - d_L$. A trajectory stops when the complementary tableau distance is greater than a threshold ϵ close to 1 (success) or the number of gates is greater than a threshold L (failure).

As a side note, one might notice that the reward function does not have a term that minimizes the number of gates. This is intrinsically embedded in the RL formulation, which we explain in more detail in Appendix E. Additionally, it is straightforward to extend the reward function to consider different objectives or constraints. For example, to minimize the number of two-qubit gates, we could add a term that gives a higher cost for two-qubit gates than for single-qubit gates.

B. Results

We apply our approach to prepare logical states of different QEC codes. Our goal is not only to demonstrate the generality of our approach by benchmarking it as broadly as possible but also to address the ongoing and timely challenge of identifying optimized circuits. We are interested in the preparation of logical states of the following QEC codes. The first code that we consider is the smallest complete error-correcting code, the $[[5,1,3]]$ perfect code [106], which has been realized experimentally, for example in [6, 7]. This code is non-CSS. We then consider several CSS codes. The first quantum error correction code, the $[[9,1,3]]$ Shor code [107], which has been realized experimentally, for example in [108, 109]. We also consider 2D and 3D color codes [74, 75]. The $[[7,1,3]]$ Steane code [71] is the smallest CSS and triangular 2D color code that has been realized experimentally, for example in [3, 6, 37, 110, 111]. We also explore the distance 5 2D color code, which is the $[[17,1,5]]$ code [75]. Finally, we consider the smallest error-correcting 3D color code, the $[[15,1,3]]$ Reed-Muller code [74, 112]. The stabilizer generators of these codes are listed in Appendix F for completeness.

In all of the codes mentioned above, we can choose $Z^{\otimes n}$ as the logical Z_L operator. This means that we prepare the $|0\rangle_L$ states of these codes, except for the $[[9,1,3]]$ Shor code, where the same choice corresponds to the $|+\rangle_L$ state. The preparation of other logical states can be achieved by changing the target logical operator accordingly. As an evaluation metric, we measure the *circuit size*, which corresponds to the number of gates in the circuit.

We first discuss the preparation of logical states on a device with all-to-all qubit connectivity and a gate set consisting of the gates H, S, and CNOT, which we refer to as the *standard gate set*. This connectivity and gate set is realistic, for example, in trapped-ion-based quantum computers [117].
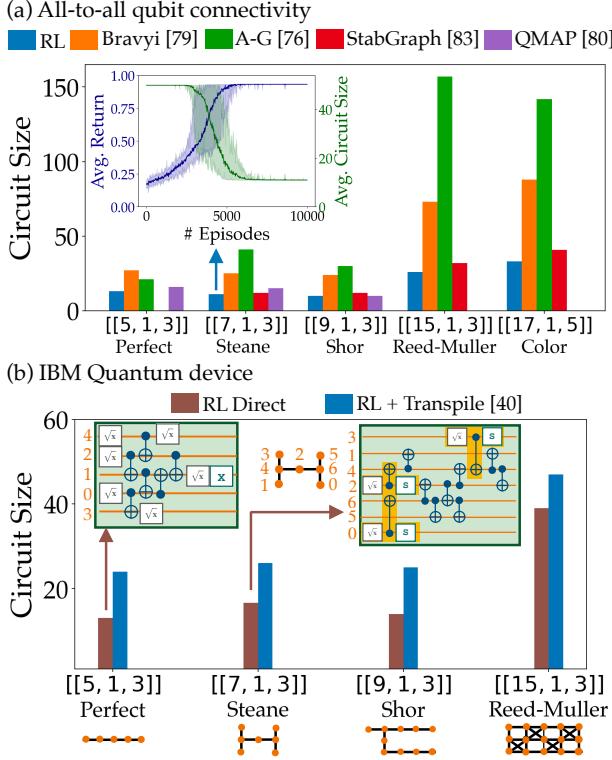
FIG. 4. Results for the logical state preparation task. (a) The minimum circuit size of different methods for logical state preparation of different QEC codes with all-to-all qubit connectivity and H, S, and CNOT gates. StabGraph [83] does not work for non-CSS codes such as the $[[5, 1, 3]]$ perfect code. QMAP [80] could not prepare the state of the $[[15, 1, 3]]$ and the $[[17, 1, 5]]$ code in the allotted maximum time of 12 hours. The inset shows an example of the training progress for preparing the $|0\rangle_L$ state of the $[[7, 1, 3]]$ Steane code. (b) Comparison of circuit size from an RL agent that includes the connectivity and gate set during training (RL Direct) with respect to RL-prepared circuits for all-to-all qubit connectivity that have been transpiled with QISKIT [40] (RL + Transpile). Results shown for various IBM Quantum device connectivities [113–116] using CNOT, $\sqrt{X}$, X, and $S = R_z(\pi/2)$ gates. The inset shows examples of RL-prepared circuits for the $|0\rangle_L$ state of the $[[5, 1, 3]]$ perfect and the $[[7, 1, 3]]$ Steane code.

We compare our RL method with four different Clifford circuit synthesis methods, where one provides the tableau and the respective methods automatically synthesize a Clifford circuit. Two of them are available in the QISKIT [118] library, based on the algorithm provided by Bravyi et al. [79] and Aaronson-Gottesman [76]. We also compare with StabGraph [83], which works only for CSS codes and uses graph states, and QMAP [80], which converts the problem into a Boolean satisfiability (SAT) problem and solves it with a SAT solver. For QMAP, we use the MAX-SAT algorithm, use the depth as the optimization target, and then minimize the number of gates. We find that using the number of gates as the optimization target of the MAX-SAT algorithm is very slow even for small n.

Fig. 4(a) shows the comparison of the smallest circuit size between different methods for preparing logical states of different codes. We see that the RL method always prepared a smaller circuit size compared to the other methods. StabGraph [83] is specialized in preparing logical states of CSS codes, therefore it does not work for the $[[5, 1, 3]]$ perfect code. QMAP [80] also did not finish the logical state preparation for $n > 10$ in the allotted maximum time of 12 hours. The inset of Fig. 4(a) shows the training progress for the preparation of the $|0\rangle_L$ for the $[[7, 1, 3]]$ Steane code. The shaded area indicates the minimum and maximum values over 10 agents trained in parallel, where each agent saw 16 environments in parallel. The entire training takes approximately 100 seconds on a single NVIDIA Quadro RTX 6000 GPU and produces 10 circuits. On average, the 10 agents converge after seeing about 6000 episodes. In Appendix G, we show some examples of circuits prepared by the RL agent and discuss some of the strategies that the RL agent learned. For example, we see that in some cases the agent would first try to get the correct tableau without worrying about the sign, and then uses Z gates (two S gates) to fix the sign.

So far, the agent is used only once after training to generate circuits for a specific logical state. However, an advantage of the deep RL method is that one can reuse the agent trained for one task and retrain it for another task. This is commonly referred to as *transfer learning* [119, 120]. For example, one can take the agent that prepares the $|0\rangle_L$ state and reuse it to train another agent that prepares the $|+\rangle_L$ state and the $|+i\rangle_L$ state more efficiently. We show these results in Appendix H.

We now show that the RL method is robust enough to adapt to different realistic qubit connectivities and gate sets from different hardware platforms by constraining the actions that the RL agent can take. We illustrate this by focusing on several IBM Quantum devices. The IBM Quantum devices have CNOT, X, $\sqrt{X}$, and R_Z gates (parameterized rotation along the z-axis) as their native gate set. Instead of using an arbitrary R_Z gate, we choose to include the S gate, which can be translated into a $R_Z(\pi/2)$ gate.

We prepare the $|0\rangle_L$ state of the $[[5, 1, 3]]$ perfect code on the IBMQ Manila [113] connectivity, the $|0\rangle_L$ state of the $[[7, 1, 3]]$ Steane code on the IBMQ Jakarta [114] connectivity, the $|+\rangle_L$ state of the $[[9, 1, 3]]$ Shor code on the IBMQ Guadalupe [115] connectivity, and the $|0\rangle_L$ state of the $[[15, 1, 3]]$ quantum Reed-Muller code on the IBMQ Tokyo [116] connectivity. These connectivities are shown at the bottom of Fig. 4(b). We trained several agents and take the circuit with the minimum circuit size.

We refer to the RL method that directly restricts the connectivity and gate set in the training as *RL Direct*. We compare it to the *RL + Transpile* method, where we take the RL-prepared circuit for all-to-all qubit connectivity and transpile it with the QISKIT transpiler [40]. Fig. 4(b) shows the comparison of circuit size between the

two methods. We see that circuits from the RL Direct method always have a smaller circuit size as compared to circuits obtained with the RL + Transpile method. This shows that restricting the actions of the RL agent based on the hardware constraint during the training is better than transpiling a circuit from all-to-all qubit connectivity.

The inset of Fig. 4(b) shows examples of a circuit prepared by the RL agent for the $|0\rangle_L$ of the $[[5,1,3]]$ perfect code on the IBMQ Manila connectivity and the $[[7,1,3]]$ code on the IBMQ Jakarta connectivity. Interestingly, we see that in the circuit for the $[[7,1,3]]$ code, the agent learns a new gate sequence $\sqrt{X}$, CNOT, and S (shaded in yellow in the figure). This gate sequence is equivalent to a H gate followed by a CNOT gate. The agent discovers this gate sequence because the H gate is not available as a native gate on IBMQ devices. Appendix I shows more examples of logical state preparation circuits on IBMQ devices.

In terms of efficiency, the training of the RL agent to prepare the $|0\rangle_L$ of the $[[7,1,3]]$ Steane code for the IBMQ Jakarta connectivity takes approximately 200 seconds on a single NVIDIA Quadro RTX 6000 GPU. One could argue that this is much slower than transpiling a circuit for all-to-all qubit connectivity. However, as we have shown, the resulting circuit size is smaller and the training only needs to be done once. Furthermore, the training can be accelerated through transfer learning. In Appendix J, we show a technique where the agent trained to prepare a logical state for all-to-all qubit connectivity can be reused and retrained to prepare the same state with different connectivity.

In summary, we have shown that RL can prepare logical states of different QEC codes with smaller circuit sizes than other methods in all-to-all qubit connectivity. We also show that by directly incorporating the hardware constraint by restricting the connectivity and gate set in the training is better than transpiling a circuit for all-to-all qubit connectivity. Furthermore, we can reuse a trained RL agent to speed up the training of the RL agent for different but similar problems.

V. VERIFICATION CIRCUIT SYNTHESIS

A. Task Description and Reward Function

The goal of the verification circuit synthesis task is to synthesize a circuit V and use the ancilla flag qubits to flag harmful errors and thereby render the encoding protocol fault-tolerant (see Fig. 5). The task requirement is the sequence of gates that form the circuit U to prepare the target logical state $|\psi\rangle_L$ and the number of the ancilla flag qubits n_A. It is possible that several circuits represent the same unitary U, but the propagated error would be different. The ancilla flag qubits are initialized in state $|0\rangle$ and are always placed last in the qubit ordering. For a given circuit, it is usually not known a priori
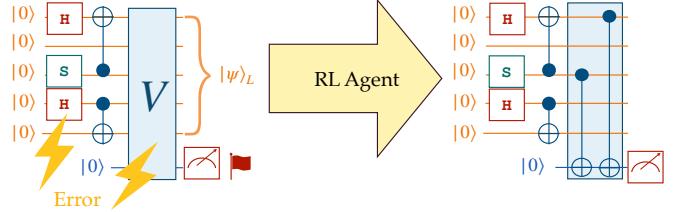
FIG. 5.   The verification circuit synthesis task prepares a circuit V that uses flag qubits to flag harmful errors, thereby rendering a state preparation fault-tolerant.

how many ancilla qubits are needed to flag all of the harmful errors. Therefore, it is possible that the agent cannot find a solution for a given number of ancillas. On the other hand, it is also possible that the agent will not use some ancillas if n_A is larger than needed.

There are underlying principles for designing FT flag verification circuits [20, 28], although they do not provide a direct recipe for FT circuit design. For instance, in the case of $d = 3$ CSS codes, state verification is equivalent to measuring some logical operators [32]. However, for higher distances or non-CSS codes, a straight-up measurement of one of the logical operators does not necessarily lead to an FT verification. Our goal is to use reinforcement learning to automatically find verification circuits under very general constraints.

We consider three criteria that must be met for this task. The first and most important criterion is to ensure that all harmful errors are flagged. While applying gates to the ancilla, it is possible that the state will change. Therefore, preserving the logical state is the second criterion. Finally, we do not want that the data (non-ancilla) qubits are entangled with the flag qubits, since this will destroy the logical state when we measure the flag. Thus, the third criterion is that the final state is a separable or product state of the data qubits and the flag qubits such that $VU|00\ldots0\rangle = |\psi\rangle_L|\phi\rangle_F$, where $|\phi\rangle_F$ is the state of the flag qubits. In summary, the RL agent must flag all harmful errors while preserving the logical state and keeping it disentangled from the flag qubits.

For the first criterion, we reward the agent based on the number of harmful errors that are flagged. We first apply circuit-level noise to the circuit U and obtain the set of all possible error operators $\mathcal{E}$ in the circuit. When the agent applies a gate, which is faulty, we update the set $\mathcal{E}$ by propagating errors from the applied gate and also the old errors. The set $\mathcal{E}$ may grow with new errors or shrink because some errors may become obsolete.

At each time step t, we compute f_t (f for flag) given as,

$$f_t = \sum_{E \in \mathcal{E}} \begin{cases} 0 & \text{if } E \text{ is } I \text{ and a flag is triggered,} \\ 1 & \text{if } E \text{ is tolerable,} \\ 1 & \text{if } E \text{ is harmful and a flag is triggered,} \\ 0 & \text{if } E \text{ is harmful and flags are not triggered.} \end{cases}$$

$$(3)$$

The first term is used to prevent the agent from choosing a naive strategy like always flagging the ancilla (e.g. applying an X gate to the flag qubits).

We then normalize f_t by dividing it by the total number of errors $|\mathcal{E}|$. Note that some errors may initially have a large weight, but can be reduced by multiplication with a member of the stabilizer group. For instance, an error with weight 4 that is a member of the stabilizer group will have weight 0 and become a tolerable error. Therefore, to consider whether an error is tolerable or harmful, we compute the minimum weight of each error by multiplying it by all members of the stabilizer group when computing the reward. For instance, for the 5-qubit code, we check all $2^5 = 32$ elements of the stabilizer group, including the state-dependent logical operator.

One might notice that if an error E is tolerable and the flag is triggered, the agent still gets a reward. This is inevitable, since it is not possible to both flag all of the harmful errors and with the same circuit construction unflag all of the tolerable errors. We can consider flagged tolerable errors as "unlucky" cases - note that this does not compromise FT, of course. One could add additional terms in the reward function to minimize this.

In principle, accurate FT circuit design requires consideration of every type of error that can occur in the circuit. However, CSS codes provide a further simplification in the design of fault-tolerant schemes. For instance, multiple Z errors usually lead to logical failures of the type $Z_L|\Psi\rangle$ (assuming Z_L consists only of Z operators). Therefore, by restricting to $|0\rangle_L$, we make multiple Z errors tolerable, since $Z_L|0\rangle_L = (+1)|0\rangle_L$. Thus, only X and Y errors at the end of the encoding circuit are potentially harmful. The same applies to the preparation of $X_L|+\rangle_L = |+\rangle_L$, if X_L consists only of X operators, it is sufficient to consider Z and Y errors. Synthesis of FT encoding circuits for codewords $|0\rangle_L$ and $|+\rangle_L$ of CSS codes is then easier in the sense that either X or Z is not harmful by construction. However, this is not true for non-CSS codes, because the stabilizers have both X and Z Paulis, so the logical operators consist of both X and Z operators.

For the second criterion, we need to make sure that the circuit preserves the logical state $|\psi\rangle_L$. Here we can use the three possible functions discussed in Sec. IV. In this case, we reuse our proposed complementary tableau distance to measure the distance between the canonical tableau of the target logical state and the current error-free canonical tableau of the data qubits.

For the third criterion, we directly enforce the state to be a separable state of the data and ancilla flag qubits. This is necessary in order not to change the error-free logical state after the measurement of the ancilla qubits. In the stabilizer formalism, the latter is achieved by targeting the stabilizer generators of the ancilla in the current error-free canonical tableau to be Z in the location of the ancilla and I in the others. We can extract the canonical tableau of the ancilla qubits by taking the submatrix of the canonical tableau where the rows are n to $n + n_A$.

Therefore, we define a value p_t (p for product state) that measures the complementary tableau distance of the current error-free canonical tableau with the target tableau according to the above criteria. For an illustration of the reward calculation, see Appendix D.

We again use the reward shaping technique, which gives the reward function,

$$r_t = \mu_f(f_t - f_{t-1}) + \mu_d(d_{t-1} - d_t) + \mu_p(p_t - p_{t-1}), \ (4)$$

where μ defines the weight for each individual reward. A trajectory stops when all of the harmful errors are flagged, the prepared state is the logical state, and the data qubits and flag qubits are a product state (success) or the number of gates is greater than a threshold L (failure).

B. Results

Let us take non-FT state preparation circuits from the literature and use the RL method to synthesize the verification circuits. We then compare them with known verification circuits.

We use three metrics to compare different verification circuits. (i) First, we compare the number of two-qubit gates (one-qubit gates do not propagate errors) and the number of flag qubits. (ii) The second metric is the *acceptance rate*. A state outcome is accepted if, after running the circuit with noise, the flag qubits are not triggered. To determine the acceptance rate numerically, we simulate 10^7 noisy circuit trajectories for each varying error probability p by adding circuit-level noise using the STIM [121] library and count the number of accepted state outcomes. (iii) The final metric is the *logical error rate p_L*. When a state outcome is accepted, we perform a perfect round of error correction on the data qubits. We can then check if the decoded state is correct, otherwise, a logical error has occurred. As discussed in Sec. II C, p_L of a fault-tolerant circuit should scale proportional to p^2 for distance-3 codes, while it scales as p for non-fault-tolerant circuits.

In our numerical experiments, we choose to use the standard gate set (H, S, and CNOT) combined with the CZ gate. The training of the agent starts with one flag qubit, and if the training does not converge, the number of flag qubits is incremented by one until a solution is found. We have also found empirically that prohibiting the agent from applying gates between the data qubits helps to speed up training convergence. In Appendix K, we show how different values of the weights μ in the reward affect the acceptance and logical error rates.

First, we synthesize the verification circuit for CSS codes. As discussed in Sec. II C, CSS codes have the favorable property that some errors are tolerable. We illustrate this by synthesizing the verification circuit for the $|0\rangle_L$ preparation of the $[[7, 1, 3]]$ Steane code with $Z_L = Z^{\otimes 7}$. The circuit was proposed in [32] (part of the
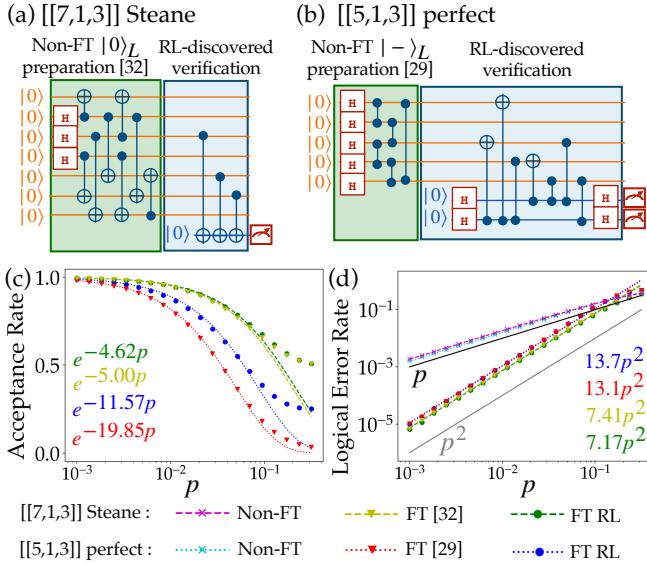
FIG. 6. Results for the verification circuit synthesis task. Examples of RL-discovered verification circuits (shaded in blue) for a given non-fault tolerant (FT) preparation (shaded in green) of (a) the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code from Ref. [32] and (b) the $|-\rangle_L$ state of the $[[5,1,3]]$ perfect code from Ref. [29]. In Ref. [32], the verification circuit uses 1 flag qubit and 3 two-qubit gates, which is the same as the circuit discovered by the RL agent. In Ref. [29], the verification circuit uses 6 flag qubits (or 2 flag qubits with 2 qubit resets) and 15 two-qubit gates, while the RL-discovered circuit in (b) uses only 2 flag qubits and 7 two-qubit gates. Comparison of the acceptance rate (c) and logical error rate (d) with different simulated error probability p for the circuits shown in (a) and (b) compared to non-FT circuits and circuits in [32] and [29].

circuit in Fig 6a shaded in green) and has been experimentally realized in [3, 6, 37, 38]. The RL agent discovers verification circuits with the same number of flag qubits and two-qubit gates as the one in [32]. Part of the circuit in Fig 6a, shaded in blue, shows an example of the verification circuit discovered by the RL agent. We show other discovered circuits in Appendix L. We observe that the RL agent learns to measure the stabilizer-equivalent logical Z operator $IIZIZZI$ without being explicitly told. Although the discovered circuit has the same number of flag qubits and two-qubit gates, we see in Fig. 6(c),(d) that the acceptance rate and the logical error rate of the RL-discovered circuit are marginally better than the verification circuit proposed in [32].

We now move on to the synthesis of verification circuits for non-CSS codes. We choose to synthesize the verification circuit for the $|-\rangle_L$ preparation of the $[[5,1,3]]$ perfect code with $X_L = XXXXX$ proposed in [29] and experimentally realized in [6, 7]. The blue-shaded part of the circuit in Fig 6b shows an example of the verification circuit discovered by RL. The RL agent learns to measure the stabilizer $IIZXZ$ in the first ancilla and the stabilizer $XIXZZ$ in the second ancilla. In Fig. 6(c),

we see that the RL-discovered circuit has a higher acceptance rate compared to the circuit in [29] due to the smaller circuit size and fewer flag qubits. Nevertheless, in Fig. 6(d), we see that the logical error rate is slightly worse than the circuit in [29]. However, the RL agent also discovers circuits with a lower logical error rate than the circuit in [29], at the expense of requiring 3 flag qubits. We show this circuit in Appendix L.

In this work, we consider only $d = 3$ codes. The discovery of verification circuits for larger codes is challenging, but we could in principle extend our RL approach to higher distance codes. We illustrate the discussion with $d = 5$ codes. In this case, the logical error rate p_L must be such that $p_L \sim p^3$, so that errors from two gate failures must be propagated. Therefore, we only need to change the way the error is propagated when training the RL agent. If L is the number of two-qubit gates, we need to propagate $O(L)$ errors for $d = 3$ and $O(L^2)$ errors for $d = 5$. In addition, L is also generally higher for $d = 5$ codes than for $d = 3$ codes. Our RL approach can be further improved by restricting the action space or by designing a better reward function, which we leave for future work.

In summary, we have shown that the RL method can be used to discover verification circuits for given non-FT logical state preparation circuits. We even show a case where the RL method discovers a better circuit than the existing circuit in the literature. Furthermore, interestingly the RL method can also discover variants of verification circuits with different tradeoffs in terms of logical error rates, acceptance rates, and the number of flag qubits.

VI. INTEGRATED FAULT-TOLERANT LOGICAL STATE PREPARATION

A. Task Description and Reward Function
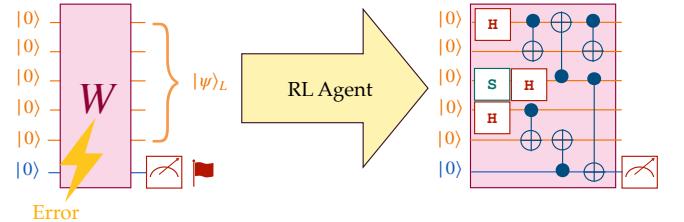
FIG. 7. The integrated fault-tolerant logical state preparation task outputs a circuit W that directly prepares $|\psi\rangle_L$ of a $[[n,k,d]]$ code in a fault-tolerant way.

Individually, we have shown that RL methods are able to achieve competitive results for the tasks of logical state preparation and verification circuit synthesis. Here, we go beyond the separation of the tasks and present our main approach that integrates them to directly prepare logical states in a fault-tolerant manner.

We expect that this integration will allow the RL agent to devise a more effective strategy compared to separating the task for two main reasons. First, it will take error propagation into account when preparing the logical state. In addition, we expect the agent to perform better when preparing a fault-tolerant logical state under limited qubit connectivity. When we consider the two goals separately, instead, the RL agent does not take into account which data qubits are connected to the flag qubits when preparing the logical state.

The goal is to find a circuit W that prepares a logical state in a fault-tolerant way (see Fig. 7). The task requirement is the tableau of the target stabilizer state s_{target} and the number of available flag qubits n_A. Note that, with respect to the previous two tasks, it is possible though not necessary that the circuit W found by the RL agent is also decomposable into the state preparation circuit U and the verification circuit V.

The reward used for this task is the same as in Eq. (4). However, in this case, the RL agent starts from scratch, so the set of error operators $\mathcal{E}$ is initially empty and grows as the agent performs actions by adding gates to the circuit construction attempt.

B. Results

1. All-to-all qubit connectivity

We first compare our two RL approaches to prepare a logical state in a fault-tolerant manner on all-to-all qubit connectivity. The first approach separates the task into *logical state preparation* (LSP) followed by *verification circuit synthesis* (VCS), which we refer to as LSP+VCS. The second one, instead, is our main approach that directly prepares the fault-tolerant logical state, which we refer to as *integrated fault-tolerant logical state preparation* (IFT-LSP).

We will discuss the preparation of the following logical states. For the CSS codes considered (i.e. the $[[7,1,3]]$ Steane code, $[[9,1,3]]$ Shor code, and $[[15,1,3]]$ Reed-Muller code), we prepare the $|0\rangle_L$ state with $Z_L = Z^{\otimes n}$ and the $|+\rangle_L$ state with $X_L = X^{\otimes n}$. For the non-CSS code (i.e. the $[[5,1,3]]$ perfect code), we prepare the $|1\rangle_L$ state with $Z_L = ZZZZZ$ and the $|-\rangle_L$ state with $X_L = XXXXX$. For this task, we also consider the $[[9,1,3]]$ Surface-17 code [17], which has been realized experimentally, for example in [5, 8, 13].

In our numerical experiments, we again use the standard gate set combined with the CZ gate. The training of the agent starts with one flag qubit, and if the training does not converge, the number of flag qubits is incremented by one until a solution is found.

We compare the minimum number of two-qubit gates and the number of ancillas needed to prepare fault-tolerant logical states with the two RL approaches (LSP+VCS and IFT-LSP) and existing circuits in Table I. IFT-LSP is better than LSP+VCS at preparing

TABLE I. The comparison of fault-tolerant logical state preparation circuits on all-to-all qubit connectivity between our two RL approaches and existing circuits. We show the minimum number of two-qubit gates and the number of flag qubits in parentheses. Bold text indicates methods with the lowest number of two-qubit gates. The first RL approach is the LSP+VCS, where we separate the task by first performing the logical state preparation (LSP in Sec. IV) followed by the verification circuit synthesis (VCS in Sec. V). The second RL approach is our main approach, which is the integrated fault-tolerant logical state preparation (IFT-LSP in Sec. VI). We see that IFT-LSP always finds circuits with less or a similar number of two-qubit gates than LSP+VCS or existing circuits.

Code	State	LSP+VCS	IFT-LSP	Existing
$[[5,1,3]]$ perfect	$|1\rangle_L$	14 (2)	**12 (2)**	-
	$|-\rangle_L$	**12 (2)**	**12 (2)**	20 (6) [29]
$[[7,1,3]]$ Steane	$|0\rangle_L$	11 (1)	11 (1)	11 (1) [32]
	$|+\rangle_L$	11 (1)	11 (1)	-
$[[9,1,3]]$ Shor	$|0\rangle_L$	6 (0)	6 (0)	-
	$|+\rangle_L$	11 (1)	11 (1)	-
$[[9,1,3]]$ Surface-17	$|0\rangle_L$	11 (1)	11 (1)	-ᵃ
	$|+\rangle_L$	11 (1)	11 (1)	-
$[[15,1,3]]$ Reed-Muller	$|0\rangle_L$	29 (2)	**25 (1)**	**25 (1)** [34]
	$|+\rangle_L$	**31 (1)**	**31 (1)**	32 (1) [34]

ᵃRef. [35] shows the FT preparation of $|0\rangle_L$ state of the $[[9,1,3]]$ surface-17 code with 8 two-qubit gates and 0 flag qubits. However, the connectivity of qubits is different, namely a 3×3 grid without ancilla qubits. See also footnote [123].

two states: $|1\rangle_L$ of $[[5,1,3]]$ perfect code and $|0\rangle_L$ of $[[15,1,3]]$ Reed-Muller code. This is most likely because the LSP does not take error propagation into account when preparing the state. Compared to existing circuits in the literature, our RL approaches find a smaller number of two-qubit gates in two states: $|-\rangle_L$ of $[[5,1,3]]$ perfect code and $|+\rangle_L$ of $[[15,1,3]]$ Reed-Muller code. The first case is already shown in Fig. 6(b), while the second case needs one two-qubit gate less than the existing one. The circuits are shown in Appendix M. In terms of efficiency, both RL approaches are comparable. For example, to prepare the $|0\rangle_L$ of the $[[7,1,3]]$ Steane code, IFT-LSP needs about 150 seconds, while LSP+VCS needs about 180 seconds on a single NVIDIA Quadro RTX 6000 GPU.

Fig. 8(a) and (b) show an example circuit for the fault-tolerant preparation of the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code and the $|1\rangle_L$ state of the $[[5,1,3]]$ perfect code discovered by IFT-LSP, respectively (see Appendix M for other examples of RL-discovered circuits). We can see that to prepare the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code, the RL agent measures the stabilizer-equivalent logical Z operator $IIZZIIZ$. When preparing the $|1\rangle_L$ state of the $[[5,1,3]]$ perfect code, the agent measures the stabilizer-equivalent logical Z oper-
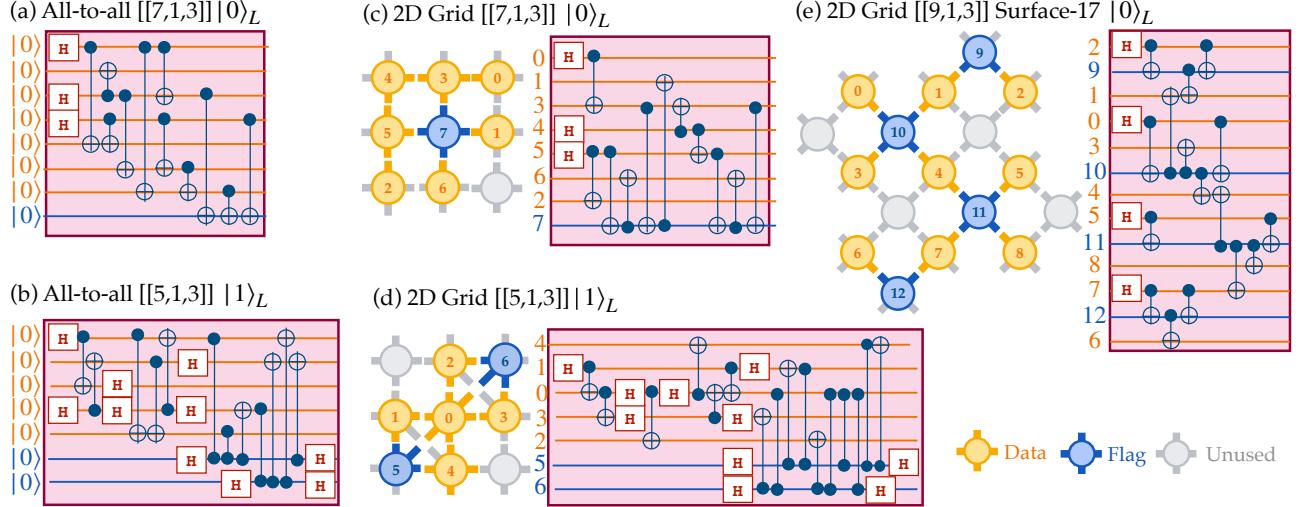
FIG. 8. Results for the RL-based integrated fault-tolerant logical state preparation (IFT-LSP). An example of a fault-tolerant circuit prepared by an RL agent for (a) the $|0\rangle_L$ state of the $[[7, 1, 3]]$ Steane code and (b) the $|1\rangle_L$ state of the $[[5, 1, 3]]$ perfect code in all-to-all qubit connectivity. Parts (c) and (d) show learned fault-tolerant circuits for the same logical state preparation task on a 2D grid connectivity based on Google Sycamore [122] (c) and IBMQ Tokyo [116] (d) devices. In (e), we show, for the first time, flag-based fault-tolerant $|0\rangle_L$ state preparation of the $[[9, 1, 3]]$ Surface-17 code on a 2D grid connectivity and qubit placement taken from Refs. [5]. Note that the flag qubits (in blue) are measured, but for simplicity, the measurement is not shown. Unused qubits or connections (in gray) mean that they are available for use by the RL agent, but are not used in the solution found by the agent.

ator $ZXIXZ$ via the first ancilla and $XXIZI$ via the second ancilla.

As a side note, one can prepare other states by changing the logical operators. Alternatively, we can also apply logical gates to a prepared state. For example, it is known that the logical H gate in the $[[7, 1, 3]]$ Steane code is transversal (applying H to each physical qubit), so it is still fault-tolerant. Thus, one can prepare $|+\rangle_L$ by applying logical H to the prepared $|0\rangle_L$. However, this is not obvious for example in the $[[5, 1, 3]]$ perfect code. In Ref. [6], the authors always prepare the $|-\rangle_L$ fault-tolerantly first, and then rotate the logical basis state to another state. With our RL approach, instead, we can automatically discover fault-tolerant preparation circuits for other states.

2. Restricted qubit connectivity

We now move to a more general and practically relevant case where we show fault-tolerant logical state preparation on a device with restricted qubit connectivity. There are some handcrafted recipes for specific codes, such as encoding the $|0\rangle_L$ state of the $[[9, 1, 3]]$ surface-17 code in a 1D array [35] and encoding a magic state of the $[[4, 1, 2]]$ code in an IBMQ device [124]. Here, we want to use RL instead to automatically discover such circuits.

Note that transpiling a fault-tolerant circuit prepared for all-to-all qubit connectivity generally does not work, since it does not guarantee that the transpiled circuit is fault-tolerant. Additionally, we find that separating

the task (LSP+VCS) fails in some cases. The first case is when a data qubit is connected only to the ancillas. In this scenario, one would have to use the ancilla as a "bridge" to the corresponding data qubit. The second case is when the VCS fails because the LSP does not take the position of the ancilla into account when preparing the logical state. In contrast, our main approach (IFT-LSP) works in these conditions. We discuss these two cases in more detail and give examples in Appendix N.

We illustrate our main approach by preparing fault-tolerant logical states on a 2D grid, which is common in quantum chips based on superconducting qubits (e.g. Google Sycamore [122], IBM Quantum devices, Rigetti Ankaa). We first demonstrate the fault-tolerant preparation of the $|0\rangle_L$ for the $[[7, 1, 3]]$ Steane code on a 3×3 grid based on the Google Sycamore device. Fig. 8(c) shows an example of an RL-discovered circuit. Impressively, the RL agent discovers a circuit with the same number of two-qubit gates and flag qubits as in the all-to-all qubit connectivity shown in Fig. 8(a). Fig. 8(d) shows an example of the fault-tolerant preparation of the $|1\rangle_L$ for the $[[5, 1, 3]]$ perfect code on a 2D grid based on the IBMQ Tokyo [116] device. Compared to the circuit on all-to-all qubit connectivity, it has the same number of flag qubits and requires only 4 additional two-qubit gates. The RL approach also manages to discover circuits for the preparation of other logical states, including for the $[[9, 1, 3]]$ Shor code, which we show in Appendix P.

The logical state of a surface code is commonly prepared fault-tolerantly by using repeated stabilizer measurements [125, 126]. Here, we for the first time show RL-

FIG. 11. Transfer learning for integrated fault-tolerant logical state preparation from an all-to-all qubit connectivity to a 2D grid connectivity. Part (a) shows the average return and (b) shows the circuit size during training for the fault-tolerant $|0\rangle_L$ preparation of the $[[7, 1, 3]]$ Steane code with and without transfer learning. The training without transfer learning also converges, but requires more training. The central part (c) shows that when we directly use the transferred agent without training, the agent retains the knowledge of placing Hadamard gates as shown previously in Fig. 10. This is in contrast to the case without transfer learning shown in (d), where the agent applies only a long sequence of self-cancelling CNOTs. In this case, the agent has not learned anything, so the probability of each gate is still uniform. The CNOT gate between qubit 7 and 5 is just a random gate chosen by the agent.

circuits shown in Fig. 8(a), (b), and (d)).

We can try to investigate the strategy that the RL agent learns by looking at the circuits and the action probabilities during the training. We illustrate this in Fig. 10. We see that in the initial training steps, the agent applies Hadamard gates to initialize some qubits in the $|+\rangle$, which is a known strategy for CSS codes [83]. Next, the agent learns to prepare the logical state circuit without flagging harmful errors. Finally, the agent learns to flag the harmful errors until the training converges. We illustrate this strategy for the integrated fault-tolerant preparation of the $|0\rangle_L$ state of the $[[7, 1, 3]]$ Steane code on all-to-all qubit connectivity in Fig. 10.

Finally, we want to explore the possibility of transfer learning in this task. Transfer learning is a powerful technique in machine learning that leverages knowledge or a strategy gained from solving one problem to solve related but different problems. However, transfer learning does not always work effectively because it depends heavily on the similarity of the problems [119].

We show a transfer learning technique where we reuse the agent that was trained to prepare a fault-tolerant logical state in an all-to-all qubit connectivity scenario to prepare the same state for the situation of restricted connectivity. The transfer learning process is explained in detail in Appendix J. We find that transfer learning helps to make the training converge faster. Additionally, we see that the transferred agent retains the strategy from the previous training. We illustrate this by comparing the integrated fault-tolerant preparation of the $|0\rangle_L$ state of the $[[7, 1, 3]]$ Steane code on a 2D grid without and with transfer learning from all-to-all qubit connectivity in Fig. 11.

In summary, we have shown that the integrated fault-tolerant logical state preparation (IFT-LSP) approach always finds circuits with better or similar performance, both compared to circuits known in the literature as well as compared to circuits found by separating the task into the two subtasks (LSP+VCS). We have also demonstrated state-of-the-art RL-based fault-tolerant logical state preparation for restricted qubit connectivity scenarios with different connectivity and qubit placements. Furthermore, with transfer learning, we can reuse an RL agent trained for all-to-all qubit connectivity to accelerate the training for restricted qubit connectivity.

VII. CONCLUSIONS AND OUTLOOK

We have presented reinforcement learning (RL) approaches to discover quantum circuits for fault-tolerant (FT) logical state preparation of QEC codes based on flag qubit protocols. We have started with the non-FT logical state preparation task and showed that RL prepares the logical state with a smaller circuit size than other methods for the all-to-all qubit connectivity scenarios. We have also highlighted that including the hardware constraint directly in the training yields quantum circuits with a smaller circuit size than transpiling a circuit for all-to-all qubit connectivity. We have then synthesized verification circuits to perform FT logical state preparation. We have demonstrated that RL can discover verification circuits that perform better than or equal to existing circuits in the literature. We have shown that the main approach that we advocate in this work, where we integrate the subtasks into the challenge of direct integrated fault-tolerant logical state preparation (IFT-LSP), performs even better than separating the tasks. Furthermore, we have demonstrated RL-based fault-tolerant logical state preparation under constrained connectivity for different qubit connectivity and placements. Finally, we have investigated and shown that transfer learning can help speed up the training process of the RL agent.

In this work, we have demonstrated the first steps in using an RL approach for the automatic discovery of quantum circuits for fault-tolerant protocols in quantum error correction. Our approach could naturally be extended and applied to different tasks, such as the dis-

covery of quantum circuits for fault-tolerant magic state preparation [36, 69], syndrome measurement [127, 128], logical gates, error correction cycles, and other quantum error correction subroutines. On the one hand, exploring these scenarios will not require a completely different approach, since verification-like circuits can be used to render the tasks fault-tolerant. However, such extensions will require a careful design of the appropriate reward function, set of actions, and observations to effectively train the RL agent. It will also be interesting to explore several avenues for scalability: on the one hand, this could include for instance devising modular formulations of the FT compilation tasks, e.g., by including more complex building blocks, such as sub-circuits for specific tasks such as stabilizer readout, in the set of actions available to the RL agent. On the other hand, it would be exciting to explore the potential of collaborative multi-agent RL scenarios, which may allow one to apply the techniques proposed in this work to larger-distance and concatenated error correction codes.

[1] J. Preskill, Quantum Computing in the NISQ era and beyond, Quantum **2**, 79 (2018), arXiv:1801.00862 [cond-mat, physics:quant-ph].

[2] D. Gottesman, An introduction to quantum error correction and fault-tolerant quantum computation, in *Proceedings of Symposia in Applied Mathematics*, Vol. 68, edited by S. Lomonaco (American Mathematical Society, Providence, Rhode Island, 2010) pp. 13–58.

[3] L. Postler, S. Heußen, I. Pogorelov, M. Rispler, T. Feldker, M. Meth, C. D. Marciniak, R. Stricker, M. Ringbauer, R. Blatt, P. Schindler, M. Müller, and T. Monz, Demonstration of fault-tolerant universal quantum gate operations, Nature **605**, 675 (2022).

[4] I. Cong, H. Levine, A. Keesling, D. Bluvstein, S.-T. Wang, and M. D. Lukin, Hardware-Efficient, Fault-Tolerant Quantum Computation with Rydberg Atoms, Phys. Rev. X **12**, 021049 (2022).

[5] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann, G. J. Norris, C. K. Andersen, M. Müller, A. Blais, C. Eichler, and A. Wallraff, Realizing repeated quantum error correction in a distance-three surface code, Nature **605**, 669 (2022).

[6] C. Ryan-Anderson, N. C. Brown, M. S. Allman, B. Arkin, G. Asa-Attuah, C. Baldwin, J. Berg, J. G. Bohnet, S. Braxton, N. Burdick, J. P. Campora, A. Chernoguzov, J. Esposito, B. Evans, D. Francois, J. P. Gaebler, T. M. Gatterman, J. Gerber, K. Gilmore, D. Gresh, A. Hall, A. Hankin, J. Hostetter, D. Lucchetti, K. Mayer, J. Myers, B. Neyenhuis, J. Santiago, J. Sedlacek, T. Skripka, A. Slattery, R. P. Stutz, J. Tait, R. Tobey, G. Vittorini, J. Walker, and D. Hayes, Implementing Fault-tolerant Entangling Gates on the Five-qubit Code and the Color Code (2022), arXiv:2208.01863 [quant-ph].

[7] M. H. Abobeih, Y. Wang, J. Randall, S. J. H. Loenen, C. E. Bradley, M. Markham, D. J. Twitchen, B. M. Terhal, and T. H. Taminiau, Fault-tolerant operation of a logical qubit in a diamond quantum processor, Nature **606**, 884 (2022).

[8] Y. Zhao, Y. Ye, H.-L. Huang, Y. Zhang, D. Wu, H. Guan, Q. Zhu, Z. Wei, T. He, S. Cao, F. Chen, T.-H. Chung, H. Deng, D. Fan, M. Gong, C. Guo, S. Guo, L. Han, N. Li, S. Li, Y. Li, F. Liang, J. Lin, H. Qian, H. Rong, H. Su, L. Sun, S. Wang, Y. Wu, Y. Xu, C. Ying, J. Yu, C. Zha, K. Zhang, Y.-H. Huo, C.-Y. Lu, C.-Z. Peng, X. Zhu, and J.-W. Pan, Realization of an Error-Correcting Surface Code with Superconducting Qubits, Phys. Rev. Lett. **129**, 030501 (2022).

[9] Y. Wang, S. Simsek, T. M. Gatterman, J. A. Gerber, K. Gilmore, D. Gresh, N. Hewitt, C. V. Horst, M. Matheny, T. Mengle, B. Neyenhuis, and B. Criger, Fault-Tolerant One-Bit Addition with the Smallest Interesting Colour Code (2023), arXiv:2309.09893 [quant-ph].

[10] D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter, J. P. Bonilla Ataides, N. Maskara, I. Cong, X. Gao, P. Sales Rodriguez, T. Karolyshyn, G. Semeghini, M. J. Gullans, M. Greiner, V. Vuletić, and M. D. Lukin, Logical quantum processor based on reconfigurable atom arrays, Nature 10.1038/s41586-023-06927-3 (2023).

[11] K. Mayer, C. Ryan-Anderson, N. Brown, E. Durso-Sabina, C. H. Baldwin, D. Hayes, J. M. Dreiling, C. Foltz, J. P. Gaebler, T. M. Gatterman, J. A. Gerber, K. Gilmore, D. Gresh, N. Hewitt, C. V. Horst, J. Johansen, T. Mengle, M. Mills, S. A. Moses, P. E. Siegfried, B. Neyenhuis, J. Pino, and R. Stutz, Benchmarking logical three-qubit quantum Fourier transform encoded in the Steane code on a trapped-ion quantum computer (2024), arXiv:2404.08616 [quant-ph].

[12] V. V. Sivak, A. Eickbusch, B. Royer, S. Singh, I. Tsioutsios, S. Ganjam, A. Miano, B. L. Brock, A. Z. Ding, L. Frunzio, S. M. Girvin, R. J. Schoelkopf, and M. H. Devoret, Real-time quantum error correction beyond break-even, Nature 616, 50 (2023).

[13] R. Acharya, I. Aleiner, R. Allen, et al., Suppressing quantum errors by scaling a surface code logical qubit, Nature 614, 676 (2023).

[14] M. P. da Silva, C. Ryan-Anderson, J. M. Bello-Rivas, A. Chernoguzov, J. M. Dreiling, C. Foltz, F. Frachon, J. P. Gaebler, T. M. Gatterman, L. Grans-Samuelsson, D. Hayes, N. Hewitt, J. Johansen, D. Lucchetti, M. Mills, S. A. Moses, B. Neyenhuis, A. Paz, J. Pino, P. Siegfried, J. Strabley, A. Sundaram, D. Tom, S. J. Wernli, M. Zanner, R. P. Stutz, and K. M. Svore, Demonstration of logical qubits and repeated error correction with better-than-physical error rates (2024), arXiv:2404.02280 [quant-ph].

[15] D. Aharonov and M. Ben-Or, Fault-Tolerant Quantum Computation with Constant Error Rate, SIAM J. Comput. 38, 1207 (2008).

[16] E. Knill, R. Laflamme, and W. H. Zurek, Resilient quantum computation: error models and thresholds, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences 454, 365 (1998).

[17] A. Kitaev, Fault-tolerant quantum computation by anyons, Annals of Physics 303, 2 (2003).

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

[19] I. K. Sohn and J. Heo, An Introduction to Fault-Tolerant Quantum Computation and its Overhead Reduction Schemes, in 2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN) (IEEE, Prague, Czech Republic, 2018) pp. 44–46.

[20] R. Chao and B. W. Reichardt, Flag Fault-Tolerant Error Correction for any Stabilizer Code, PRX Quantum 1, 010302 (2020).

[21] S. Heußen, D. F. Locher, and M. Müller, Measurement-free fault-tolerant quantum error correction in near-term devices (2023).

[22] J. Preskill, Reliable Quantum Computers, Proc. R. Soc. Lond. A 454, 385 (1998), arXiv:quant-ph/9705031.

[23] P. Shor, Fault-tolerant quantum computation, in Proceedings of 37th Conference on Foundations of Computer Science (1996) pp. 56–65, iSSN: 0272-5428.

[24] A. M. Steane, Active Stabilization, Quantum Computation, and Quantum State Synthesis, Phys. Rev. Lett. 78, 2252 (1997).

[25] S. Huang, K. R. Brown, and M. Cetina, Comparing Shor and Steane Error Correction Using the Bacon-Shor Code (2023), arXiv:2312.10851 [quant-ph].

[26] L. Postler, F. Butt, I. Pogorelov, C. D. Marciniak, S. Heußen, R. Blatt, P. Schindler, M. Rispler, M. Müller, and T. Monz, Demonstration of fault-tolerant Steane quantum error correction (2023), arXiv:2312.09745 [quant-ph].

[27] T. J. Yoder and I. H. Kim, The surface code with a twist, Quantum 1, 2 (2017), arXiv:1612.04795 [quant-ph].

[28] C. Chamberland and M. E. Beverland, Flag fault-tolerant error correction with arbitrary distance codes, Quantum 2, 53 (2018), arXiv:1708.02246 [quant-ph].

[29] R. Chao and B. W. Reichardt, Quantum error correction with only two extra qubits, Phys. Rev. Lett. 121, 050502 (2018), arXiv:1705.02329 [quant-ph].

[30] C. Chamberland and K. Noh, Very low overhead fault-tolerant magic state preparation using redundant ancilla encoding and flag qubits, npj Quantum Inf 6, 91 (2020).

[31] C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross, Topological and Subsystem Codes on Low-Degree Graphs with Flag Qubits, Phys. Rev. X 10, 011022 (2020).

[32] H. Goto, Minimizing resource overheads for fault-tolerant preparation of encoded states of the Steane code, Sci Rep 6, 19578 (2016).

[33] A. Paetznick and B. W. Reichardt, Fault-tolerant ancilla preparation and noise threshold lower bounds for the 23-qubit Golay code (2013).

[34] F. Butt, S. Heußen, M. Rispler, and M. Müller, Fault-Tolerant Code Switching Protocols for Near-Term Quantum Processors (2023), arXiv:2306.17686 [quant-ph].

[35] H. Goto, Y. Ho, and T. Kanao, Measurement-free fault-tolerant logical-zero-state encoding of the distance-three nine-qubit surface code in a one-dimensional qubit array, Phys. Rev. Res. 5, 043137 (2023).

[36] C. Chamberland and A. W. Cross, Fault-tolerant magic state preparation with flag qubits, Quantum 3, 143 (2019), arXiv:1811.00566 [quant-ph].

[37] C. Ryan-Anderson, J. Bohnet, K. Lee, D. Gresh, A. Hankin, J. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. Brown, T. Gatterman, S. Halit, K. Gilmore, J. Gerber, B. Neyenhuis, D. Hayes, and R. Stutz, Realization of Real-Time Fault-Tolerant Quantum Error Correction, Phys. Rev. X 11, 041058 (2021).

[38] J. Hilder, D. Pijn, O. Onishchenko, A. Stahl, M. Orth, B. Lekitsch, A. Rodriguez-Blanco, M. Müller, F. Schmidt-Kaler, and U. Poschinger, Fault-Tolerant Parity Readout on a Shuttling-Based Trapped-Ion Quantum Computer, Phys. Rev. X 12, 011032 (2022).

[39] I. Pogorelov, F. Butt, L. Postler, C. D. Marciniak, P. Schindler, M. Müller, and T. Monz, Experimental fault-tolerant code switching (2024), arXiv:2403.13732 [quant-ph].

[40] Qiskit transpiler, https://qiskit.org/documentation/apidoc/transpiler.html, accessed: 2024-01-17.

[41] F. Hua, M. Wang, G. Li, B. Peng, C. Liu, M. Zheng, S. Stein, Y. Ding, E. Z. Zhang, T. Humble, and others, QASMTrans: A QASM Quantum Transpiler Framework for NISQ Devices, in Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (2023) pp. 1468–1477.

[42] E. Younis and C. Iancu, Quantum Circuit Optimization and Transpilation via Parameterized Circuit Instantiation, in 2022 IEEE International Conference on Quan-

tum Computing and Engineering (QCE) (IEEE, Broomfield, CO, USA, 2022) pp. 465–475.

[43] M. Maronese, L. Moro, L. Rocutto, and E. Prati, Quantum Compiling, in *Quantum Computing Environments*, edited by S. S. Iyengar, M. Mastriani, and K. L. Kumar (Springer International Publishing, Cham, 2022) pp. 39–74.

[44] L. Schmid, D. F. Locher, M. Rispler, S. Blatt, J. Zeiher, M. Müller, and R. Wille, Computational Capabilities and Compiler Development for Neutral Atom Quantum Processors: Connecting Tool Developers and Hardware Experts (2023), arXiv:2309.08656 [quant-ph].

[45] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, t|ket⟩ : A Retargetable Compiler for NISQ Devices, Quantum Sci. Technol. **6**, 014003 (2021), arXiv:2003.10611 [quant-ph].

[46] N. Paraskevopoulos, F. Sebastiano, C. G. Almudever, and S. Feld, SpinQ: Compilation strategies for scalable spin-qubit architectures (2023), arXiv:2301.13241 [quant-ph].

[47] F. Kreppel, C. Melzer, D. O. Millán, J. Wagner, J. Hilder, U. Poschinger, F. Schmidt-Kaler, and A. Brinkmann, Quantum Circuit Compiler for a Shuttling-Based Trapped-Ion Quantum Computer, Quantum **7**, 1176 (2023), arXiv:2207.01964 [quant-ph].

[48] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, 2nd ed., Adaptive computation and machine learning series (The MIT Press, Cambridge, Massachusetts, 2018).

[49] M. Krenn, J. Landgraf, T. Foesel, and F. Marquardt, Artificial intelligence and machine learning for quantum technologies, Phys. Rev. A **107**, 010101 (2023).

[50] T. Fösel, P. Tighineanu, T. Weiss, and F. Marquardt, Reinforcement Learning with Neural Networks for Quantum Feedback, Phys. Rev. X **8**, 031084 (2018).

[51] H. P. Nautrup, N. Delfosse, V. Dunjko, H. J. Briegel, and N. Friis, Optimizing Quantum Error Correction Codes with Reinforcement Learning, Quantum **3**, 215 (2019), arXiv:1812.08451 [quant-ph].

[52] P. Andreasson, J. Johansson, S. Liljestrand, and M. Granath, Quantum error correction for the toric code using deep reinforcement learning, Quantum **3**, 183 (2019), arXiv:1811.12338 [cond-mat, physics:quant-ph].

[53] R. Sweke, M. S. Kesselring, E. P. L. Van Nieuwenburg, and J. Eisert, Reinforcement learning decoders for fault-tolerant quantum computation, Mach. Learn.: Sci. Technol. **2**, 025005 (2021).

[54] J. Olle, R. Zen, M. Puviani, and F. Marquardt, Simultaneous Discovery of Quantum Error Correction Codes and Encoders with a Noise-Aware Reinforcement Learning Agent (2023), arXiv:2311.04750 [quant-ph].

[55] M. Puviani, S. Borah, R. Zen, J. Olle, and F. Marquardt, Boosting the Gottesman-Kitaev-Preskill quantum error correction with non-Markovian feedback (2023), arXiv:2312.07391 [quant-ph].

[56] X.-M. Zhang, Z. Wei, R. Asad, X.-C. Yang, and X. Wang, When does reinforcement learning stand out in quantum control? A comparative study on state preparation, npj Quantum Inf **5**, 85 (2019).

[57] M. Bukov, A. G. Day, D. Sels, P. Weinberg, A. Polkovnikov, and P. Mehta, Reinforcement Learning in Different Phases of Quantum Control, Phys. Rev. X **8**, 031086 (2018).

[58] R. Porotti, A. Essig, B. Huard, and F. Marquardt, Deep Reinforcement Learning for Quantum State Preparation with Weak Nonlinear Measurements, Quantum **6**, 747 (2022), arXiv:2107.08816 [quant-ph].

[59] T. Haug, W.-K. Mok, J.-B. You, W. Zhang, C. Eng Png, and L.-C. Kwek, Classifying global state preparation via deep reinforcement learning, Mach. Learn.: Sci. Technol. **2**, 01LT02 (2021).

[60] V. Sivak, A. Eickbusch, H. Liu, B. Royer, I. Tsioutsios, and M. Devoret, Model-Free Quantum Control with Reinforcement Learning, Phys. Rev. X **12**, 011059 (2022).

[61] S. Giordano and M. A. Martin-Delgado, Reinforcement-learning generation of four-qubit entangled states, Phys. Rev. Research **4**, 043056 (2022).

[62] L. Moro, M. G. A. Paris, M. Restelli, and E. Prati, Quantum compiling by deep reinforcement learning, Commun Phys **4**, 178 (2021).

[63] Y.-H. Zhang, P.-L. Zheng, Y. Zhang, and D.-L. Deng, Topological Quantum Compiling with Reinforcement Learning, Phys. Rev. Lett. **125**, 170501 (2020), arXiv:2004.04743 [cond-mat, physics:quant-ph].

[64] Q. Chen, Y. Du, Q. Zhao, Y. Jiao, X. Lu, and X. Wu, Efficient and practical quantum compiler towards multi-qubit systems with deep reinforcement learning (2022), arXiv:2204.06904 [quant-ph].

[65] Z. He, L. Li, S. Zheng, Y. Li, and H. Situ, Variational quantum compiling with double Q-learning, New J. Phys. **23**, 033002 (2021).

[66] W. Gong, S. Jiang, and D.-L. Deng, No-go theorem and a universal decomposition strategy for quantum channel compilation, Phys. Rev. Research **5**, 013060 (2023).

[67] L. M. Trenkwalder, E. Scerri, T. E. O'Brien, and V. Dunjko, Compilation of product-formula Hamiltonian simulation via reinforcement learning (2023), arXiv:2311.04285 [quant-ph].

[68] F. Preti, M. Schilling, S. Jerbi, L. M. Trenkwalder, H. P. Nautrup, F. Motzoi, and H. J. Briegel, Hybrid discrete-continuous compilation of trapped-ion quantum circuits with deep reinforcement learning (2023), arXiv:2307.05744 [quant-ph].

[69] N. Shutty and C. Chamberland, Decoding Merged Color-Surface Codes and Finding Fault-Tolerant Clifford Circuits Using Solvers for Satisfiability Modulo Theories, Phys. Rev. Applied **18**, 014072 (2022), arXiv:2201.12450 [quant-ph].

[70] D. Gottesman, Stabilizer Codes and Quantum Error Correction (1997), arXiv:quant-ph/9705052.

[71] A. Steane, Multiple-particle interference and quantum error correction, Proc. R. Soc. Lond. A **452**, 2551 (1996).

[72] A. R. Calderbank and P. W. Shor, Good quantum error-correcting codes exist, Phys. Rev. A **54**, 1098 (1996).

[73] A. Y. Kitaev, Quantum computations: algorithms and error correction, Russ. Math. Surv. **52**, 1191 (1997).

[74] H. Bombín, Gauge color codes: optimal transversal gates and gauge fixing in topological stabilizer codes, New J. Phys. **17**, 083002 (2015).

[75] H. Bombin and M. A. Martin-Delgado, Topological Quantum Distillation, Phys. Rev. Lett. **97**, 180501 (2006).

[76] S. Aaronson and D. Gottesman, Improved simulation of stabilizer circuits, Phys. Rev. A **70**, 052328 (2004).

[77] S. Bravyi, J. A. Latone, and D. Maslov, 6-qubit Optimal Clifford Circuits, npj Quantum Inf **8**, 79 (2022), arXiv:2012.06074 [quant-ph].

[78] V. Kliuchnikov and D. Maslov, Optimization of Clifford Circuits, Phys. Rev. A **88**, 052307 (2013), arXiv:1305.0810 [quant-ph].

[79] S. Bravyi, R. Shaydulin, S. Hu, and D. Maslov, Clifford Circuit Optimization with Templates and Symbolic Pauli Gates, Quantum **5**, 580 (2021), arXiv:2105.02291 [quant-ph].

[80] S. Schneider, L. Burgholzer, and R. Wille, A SAT Encoding for Optimal Clifford Circuit Synthesis, in *Proceedings of the 28th Asia and South Pacific Design Automation Conference* (ACM, Tokyo Japan, 2023) pp. 190–195.

[81] D. Winder, Q. Huang, A. M.-v. de Griend, and R. Yeung, Architecture-Aware Synthesis of Stabilizer Circuits from Clifford Tableaus (2023), arXiv:2309.08972 [quant-ph].

[82] P. Niemann, R. Wille, and R. Drechsler, Efficient synthesis of quantum circuits implementing clifford group operations, in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)* (IEEE, Singapore, 2014) pp. 483–488.

[83] D. Amaro, M. Müller, and A. K. Pal, Scalable characterization of localizable entanglement in noisy topological quantum codes, New J. Phys. **22**, 053038 (2020).

[84] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*, 10th ed. (Cambridge University Press, Cambridge ; New York, 2010).

[85] N. Rengaswamy, R. Calderbank, S. Kadhe, and H. D. Pfister, Logical Clifford Synthesis for Stabilizer Codes, IEEE Trans. Quantum Eng. **1**, 1 (2020), arXiv:1907.00310 [quant-ph].

[86] A. Mondal and K. K. Parhi, Systematic Design and Optimization of Quantum Circuits for Stabilizer Codes (2023), arXiv:2309.12373 [quant-ph].

[87] D. Tandeitnik and T. Guerreiro, Evolving Quantum Circuits (2022), number: arXiv:2210.05058 arXiv:2210.05058 [quant-ph].

[88] A. M. Steane, Overhead and noise threshold of fault-tolerant quantum error correction, Phys. Rev. A **68**, 042322 (2003).

[89] C. Chamberland, A. Kubica, T. J. Yoder, and G. Zhu, Triangular color codes on trivalent graphs with flag qubits, New J. Phys. **22**, 023019 (2020).

[90] E. Knill, Quantum computing with realistically noisy devices, Nature **434**, 39 (2005).

[91] T. Tansuwannont and D. Leung, Achieving Fault Tolerance on Capped Color Codes with Few Ancillas, PRX Quantum **3**, 030322 (2022).

[92] H. Bombín, M. Pant, S. Roberts, and K. I. Seetharam, Fault-tolerant Post-Selection for Low Overhead Magic State Preparation (2022).

[93] L. Egan, D. M. Debroy, C. Noel, A. Risinger, D. Zhu, D. Biswas, M. Newman, M. Li, K. R. Brown, M. Cetina, and C. Monroe, Fault-Tolerant Operation of a Quantum Error-Correction Code (2021).

[94] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, Policy Gradient Methods for Reinforcement Learning with Function Approximation, Advances in neural information processing systems **12** (1999).

[95] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal Policy Optimization Algorithms (2017), arXiv:1707.06347 [cs].

[96] T. Fösel, M. Y. Niu, F. Marquardt, and L. Li, Quantum circuit optimization with deep reinforcement learning (2021), arXiv:2103.07585 [quant-ph].

[97] M. Ostaszewski, L. M. Trenkwalder, W. Masarczyk, E. Scerri, and V. Dunjko, Reinforcement learning for optimization of variational quantum circuit architectures, Advances in Neural Information Processing Systems **34**, 18182 (2021).

[98] T. Gabor, M. Zorn, and C. Linnhoff-Popien, The applicability of reinforcement learning for the automatic generation of state preparation circuits, in *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (ACM, Boston Massachusetts, 2022) pp. 2196–2204.

[99] X. Xu, S. C. Benjamin, and X. Yuan, Variational circuit compiler for quantum error correction, Phys. Rev. Applied **15**, 034068 (2021), arXiv:1911.05759 [quant-ph].

[100] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, Barren plateaus in quantum neural network training landscapes, Nat Commun **9**, 4812 (2018).

[101] C. Lu, J. G. Kuba, A. Letcher, L. Metz, C. S. de Witt, and J. Foerster, Discovered Policy Optimisation, Advances in Neural Information Processing Systems **35**, 16455 (2022).

[102] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, JAX: composable transformations of Python+NumPy programs (2018).

[103] https://github.com/remmyzen/rlftqc.

[104] S. Z. Baba, N. Yoshioka, Y. Ashida, and T. Sagawa, Deep Reinforcement Learning for Preparation of Thermal and Prethermal Quantum States, Phys. Rev. Applied **19**, 014068 (2023).

[105] J. Mackeprang, D. B. R. Dasari, and J. Wrachtrup, A reinforcement learning approach for quantum state engineering, Quantum Mach. Intell. **2**, 5 (2020).

[106] R. Laflamme, C. Miquel, J. P. Paz, and W. H. Zurek, Perfect Quantum Error Correcting Code, Phys. Rev. Lett. **77**, 198 (1996).

[107] P. W. Shor, Scheme for reducing decoherence in quantum computer memory, Phys. Rev. A **52**, R2493 (1995).

[108] N. H. Nguyen, M. Li, A. M. Green, C. Huerta Alderete, Y. Zhu, D. Zhu, K. R. Brown, and N. M. Linke, Demonstration of Shor Encoding on a Trapped-Ion Quantum Computer, Phys. Rev. Applied **16**, 024057 (2021).

[109] R. Zhang, L.-Z. Liu, Z.-D. Li, Y.-Y. Fei, X.-F. Yin, L. Li, N.-L. Liu, Y. Mao, Y.-A. Chen, and J.-W. Pan, Loss-tolerant all-photonic quantum repeater with generalized Shor code, Optica **9**, 152 (2022).

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

[111] D. Bluvstein, H. Levine, G. Semeghini, T. T. Wang, S. Ebadi, M. Kalinowski, A. Keesling, N. Maskara, H. Pichler, M. Greiner, V. Vuletić, and M. D. Lukin, A quantum processor based on coherent transport of entangled atom arrays, Nature **604**, 451 (2022).

[112] J. T. Anderson, G. Duclos-Cianci, and D. Poulin, Fault-tolerant conversion between the Steane and Reed-Muller quantum codes, Phys. Rev. Lett. **113**, 080501 (2014), arXiv:1403.2734 [quant-ph].

[113] FakeManilaV2, https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.fake_

provider.FakeManilaV2, accessed: 2024-02-20.

[114] FakeJakartaV2, https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.fake_provider.FakeJakartaV2, accessed: 2024-02-20.

[115] FakeGuadalupeV2, https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.fake_provider.FakeGuadalupeV2, accessed: 2024-02-20.

[116] FakeTokyo, https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.fake_provider.FakeTokyo, accessed: 2024-02-20.

[117] J. I. Cirac and P. Zoller, Quantum Computations with Cold Trapped Ions, Phys. Rev. Lett. **74**, 4091 (1995).

[118] Qiskit contributors, Qiskit: An open-source framework for quantum computing (2023).

[119] M. E. Taylor and P. Stone, Transfer Learning for Reinforcement Learning Domains: A Survey, Journal of Machine Learning Research **10** (2009).

[120] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou, Transfer Learning in Deep Reinforcement Learning: A Survey, IEEE Trans. Pattern Anal. Mach. Intell. **45**, 13344 (2023).

[121] C. Gidney, Stim: a fast stabilizer circuit simulator, Quantum **5**, 497 (2021), arXiv:2103.02202 [quant-ph].

[122] F. Arute, K. Arya, R. Babbush, *et al.*, Quantum supremacy using a programmable superconducting processor, Nature **574**, 505 (2019).

[123] Currently, the RL method only looks at weight-1 errors of $d = 3$ codes, thus missing the possible correctable weight-2 errors in the surface code. The latter can be easily implemented in a version of the method tailored to specific codes. However, our goal is to have a method that is as general as possible..

[124] R. S. Gupta, N. Sundaresan, T. Alexander, C. J. Wood, S. T. Merkel, M. B. Healy, M. Hillenbrand, T. Jochym-O'Connor, J. R. Wootton, T. J. Yoder, A. W. Cross, M. Takita, and B. J. Brown, Encoding a magic state with beyond break-even fidelity, Nature **625**, 259 (2024).

[125] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Surface codes: Towards practical large-scale quantum computation, Phys. Rev. A **86**, 032324 (2012).

[126] Y. Ye, T. He, H.-L. Huang, Z. Wei, Y. Zhang, Y. Zhao, D. Wu, Q. Zhu, H. Guan, S. Cao, F. Chen, T.-H. Chung, H. Deng, D. Fan, M. Gong, C. Guo, S. Guo, L. Han, N. Li, S. Li, Y. Li, F. Liang, J. Lin, H. Qian, H. Rong, H. Su, S. Wang, Y. Wu, Y. Xu, C. Ying, J. Yu, C. Zha, K. Zhang, Y.-H. Huo, C.-Y. Lu, C.-Z. Peng, X. Zhu, and J.-W. Pan, Logical Magic State Preparation with Fidelity Beyond the Distillation Threshold on a Superconducting Quantum Processor (2023), arXiv:2305.15972 [quant-ph].

[127] N. Delfosse and B. W. Reichardt, Short shor-style syndrome sequences (2020), arXiv:2008.05051 [quant-ph].

[128] D. Bhatnagar, M. Steinberg, D. Elkouss, C. G. Almudever, and S. Feld, Low-Depth Flag-Style Syndrome Extraction for Small Quantum Error-Correction Codes, in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)* (2023) pp. 63–69, arXiv:2305.00784 [quant-ph].

[129] S. Kosub, A note on the triangle inequality for the Jaccard distance (2016), arXiv:1612.02696 [cs, stat].

[130] FakeLimaV2, https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.fake_provider.FakeLimaV2, accessed: 2024-02-20.

Appendix A: Tableau representation

Here, we give more details about the representation of quantum circuits as tableaus. Note that in this work, we omit the "destabilizer" generators in the tableau described in [76], since they are not useful for the task at hand.

A tableau can be represented as a $n \times (2n+1)$ matrix of binary variables x_{ij}, z_{ij}, r_i for $i, j \in \{1, \ldots, n\}$. Each row i of the tableau $[x_{i1}, \ldots x_{in}, z_{i1}, \ldots, z_{in}, r_i]$ represents the Pauli matrix of the generators or the logical operators, where the $x_{ij} z_{ij}$ bits determine the j-th Pauli matrix, where 00, 01, 10, and 11 denote I, Z, X, and Y Pauli, respectively, and r_i denotes the phase (1 for negative phase and 0 for positive phase). For instance, a binary vector $[10011|00110|1]$ represents the Pauli $-XIZYX$.

For instance, the tableau for $|0\rangle_L$ of the $[[7, 1, 3]]$ Steane code [71] is a matrix of binary numbers of size 7×15 that contains $7 - 1 = 6$ stabilizer generators and 1 logical operator Z_L. Table II shows the generators of the $[[7, 1, 3]]$ Steane code. Eq. (A1) shows an example of the tableau for $|0\rangle_L$ of the $[[7, 1, 3]]$ Steane code when we choose $Z_L = +ZZZZZZZ$. In this tableau, the first row represents the logical operator $Z_L = +ZZZZZZZ$, the second row represents the first stabilizer operator $+ZIZIZIZ$, and so on.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (A1)$$

As discussed in the main text, reordering the rows of the tableau represents the same state. However, this is not the case for the canonical tableau [76], which is a unique representation of a tableau that represents the same state. We can apply Gaussian elimination to the matrix in Eq. (A1) and get the canonical tableau in Eq. (A2). The Pauli string for this tableau is: $+XIXIXIX$, $+ZIIIIZZ$, $+IXXIIXX$, $+IZIIZIZ$, $+IIZIZZI$, $+IIIXXXX$, and $+IIIZZZZ$. We can reorder the rows or change a row by multiplying other rows in Eq. (A1) and it will still give the same canonical tableau.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \quad (A2)$$

We flatten this matrix into a vector and use it to compute the distance metric and as an input to the neural networks.

Appendix B: Hyperparameters and details of the training

We use the multilayer perceptron architecture for the critic and actor networks to train the PPO algorithm [94]. Both of the networks have two hidden layers with the ReLU activation function. The number of hidden nodes is set to 128. However, in cases where the number of physical qubits is more than 10 with all-to-all qubit connectivity, we increase the number of hidden nodes to 256. The weight matrices are initialized with a uniformly distributed orthogonal matrix with 0.01 scale, while the biases are initialized to zero.

The hyperparameters of the PPO training are as follows [94]. We use the Adam optimizer with a learning rate of 0.001 with an annealing learning rate. We train 10 agents in parallel. Each agent sees batches of 16 environments. We train the agent for a total of one million time steps with an entropy coefficient of 0.05. The network is updated after every 4 epochs and the number of minibatches is set to 4. The discount factor (γ) is set to 0.99, the generalized advantage estimate (GAE) value (λ) is set to 0.95, the clipping parameter (ϵ) is set to 0.2, the value function coefficient is set to 0.5, and the maximum gradient norm clip value is set to 0.5. For harder cases (e.g. larger physical qubits or restricted connectivity), we increase the total time steps to 10 or 30 million and change the learning rate to 0.0005 and the entropy coefficient to 0.1. All experimental results shown in this paper are done on NVIDIA Quadro RTX 6000 GPU. The training is done with the same seed value to ensure the same randomness.

For all experiments, we set the stopping threshold $\epsilon = 0.9999$ and the maximum steps in the trajectory $L = 50$ and in harder cases to $L = 100$. For the reward of verification circuit synthesis in Eq.(4), we set $\mu_f = n$, $\mu_d = \lfloor n/2 \rfloor$, and $\mu_p = 1$. For the reward of fault-tolerant logical state preparation also defined in Eq.(4), we set $\mu_d = n$, $\mu_f = \lfloor n/2 \rfloor$, and $\mu_p = 1$. These values are determined from our numerical experiments by varying the weight, which we discuss in Appendix K and Appendix Q.

Appendix C: Comparison of distance functions

We propose to use the complementary distance $1 - d_t$ between the target canonical tableau (G_{target}) and the canonical tableau of the current circuit at time t (G_{t}) as a reward to the RL agent. We first convert the current and target canonical tableau matrices into binary vectors and compute the distance. In principle, we can take any binary distance measure. A natural choice of metric is the Hamming distance, which fits the current application since it is mostly used in coding theory. However, our empirical experiments showed that the Jaccard distance works better than the Hamming distance.

Let C_{11} be the number of elements in G_{target} and G_t that have a value 1 at the same position, C_{00} the number of elements in G_{target} and G_t that have a value of 0 at the same position, C_{01} the number of elements that have a value of 0 in G_{target} and 1 in G_t at the same position, and C_{10} the number of elements that have a value 1 in G_{target} and 0 in G_t at the same position.

The Hamming distance d_H is defined as follows

$$d_H = \frac{C_{01} + C_{10}}{C_{00} + C_{01} + C_{10} + C_{11}}, \tag{C1}$$

while the Jaccard distance [129] d_J is defined as

$$d_J = \frac{C_{01} + C_{10}}{C_{01} + C_{10} + C_{11}}. \tag{C2}$$

We compare the RL training for preparing the $|0\rangle_L$ state of the $[[7, 1, 3]]$ Steane code using Hamming and Jaccard distance. In Fig. 12, we see that results based on using the Jaccard distance converge faster and more stably than those obtained by using the Hamming distance. As seen in Eq. (C2), the computation of the Jaccard distance does not take into account the C_{00}. In stabilizer language, this means that we do not take into account when the Pauli identity I matches in both target and current tableau. This also means that the Jaccard distance penalizes more dissimilarities compared to the Hamming distance. We hypothesize that since we are using a reward shaping technique, the increase in the inverse distance is larger when using the Jaccard distance, which is better learned by the RL agent. We illustrate this in the inset of Fig. 12, where we test an RL-prepared circuit and compare how the value of inverse distance evolves for each step of applying a gate. We see that the Jaccard distance starts lower and increases more steeply than the Hamming distance. Therefore, we use the Jaccard distance d_J as the distance metric d_t for our experiments.

Appendix D: Calculation of the complementary tableau distance and product state

Here we show an example of distance and product state calculations, as explained in Sec. V, which are part of the reward function for verification circuit synthesis and integrated fault-tolerant logical state preparation tasks.

As an example, consider the preparation of the state $|000\rangle + |111\rangle$ with two ancillas $(n_A = 2)$. The state has the following target canonical tableau: $+XXX$, $+ZIZ$, and $+IZZ$. We need to append the last column of the target canonical tableau with $I^{\otimes n_A}$, so the target canonical tableau for the data qubits is now: $+XXXII$, $+ZIZII$, $+IZZII$, as shown in Fig. 13(a). As mentioned in Sec. V, the stabilizer generators of the ancilla must be of Z-type in the location of the ancilla and I in the others for it to be a product state. Therefore, the target canonical tableau of the ancilla qubits must be: $+IIIZI$ and $+IIIIZ$, as shown in Fig. 13(b).
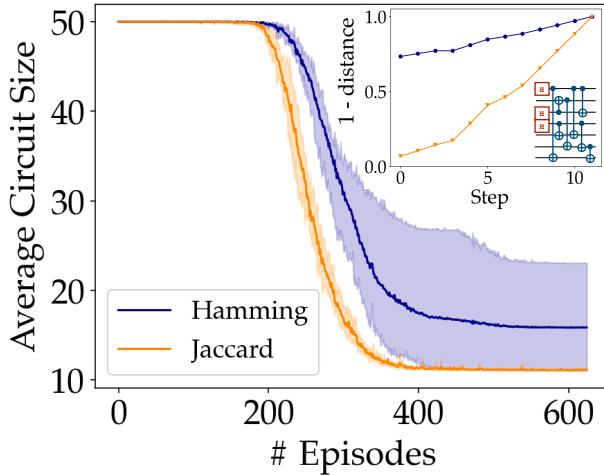
FIG. 12. Comparison of Hamming and Jaccard distance for the reward function. Average circuit size of training an RL agent to prepare the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code using the Hamming and Jaccard distance for the complementary tableau distance reward $1 - d_t$. The color shade shows the standard deviation over five different trainings. The inset shows how the inverse distance value evolves for each step of gate application for the circuit shown.
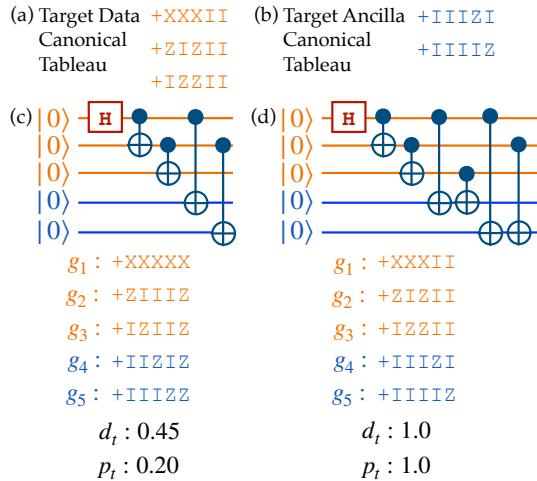

FIG. 13. Illustration of the complementary tableau distance (d_t) and product state (p_t) calculation. We want to prepare the state $|000\rangle + |111\rangle$ with two ancillas ($n_A = 2$). The target canonical tableau can be separated according to the data (a) and ancilla (b) qubits. (c) shows an example of a circuit that almost prepares the target state, but the data qubits are entangled with the ancilla. (d) shows an example of a circuit that prepares the target state and is a product state of the data qubits and the ancillas.

Fig. 13(c) and (d) show two circuit examples. To compute the complementary tableau distance d_t, we take g_1, g_2, and g_3 of the canonical tableau of the circuit and compute the complementary tableau distance with the target data canonical tableau. Since the ancilla qubits are always placed last, we can extract the canonical

tableau of the data qubits by taking the submatrix of the canonical tableau from rows 1 to n. To compute the product state p_t, we first take g_4 and g_5 from the canonical tableau of the circuit. We then compute the d_t between this subtableau and the target ancilla canonical tableau. We can see that the circuit in Fig. 13(c) is still far from the target state because the data qubits are entangled with the ancilla, since both the d_t and p_t are below 1. While the circuit in Fig. 13(d) correctly prepares the state ($d_t = 1$) and is a product state of the data qubits and the ancillas ($p_t = 1$).

Appendix E: Minimizing the number of gates in the reward function
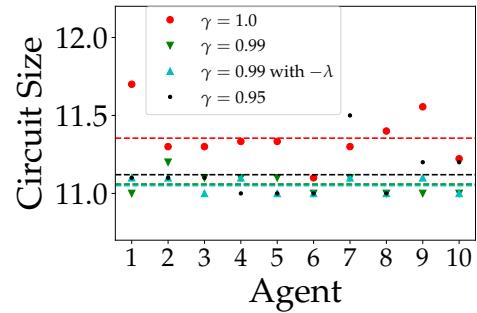
FIG. 14. Illustration of how the discount factor γ affects the circuit size. We train 10 different agents for preparing the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code with $\gamma = \{0.95, 0.99, 1\}$. Additionally, we also tried to set γ to 0.99 and add the $\lambda = -1/50$ (referred in the figure as $\gamma = 0.99$ with $-\lambda$) in the reward function to penalize longer trajectories. The dashed line shows the average circuit size.

One might notice that the reward function in the logical state preparation task does not include a term that minimizes the number of gates to make the preparation circuit compact. The most common technique is to add an extra term $-\lambda$ at each time step to penalize longer sequences, so that the reward function in Eq. 2 is $r_t = d_{t-1} - d_t - \lambda$. The problem is that λ is now a hyperparameter that must be tuned so that it does not become stronger than the complementary tableau distance reward.

However, without this term, one can also use the discount factor γ in the cumulative reward [48]. Instead of maximizing $\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} r_t]$, we instead maximize $\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} \gamma^t r_t]$. The γ value ranges from 0 to 1. It determines how much future rewards are reduced in value compared to immediate rewards. When $\gamma = 1$, the agent values future rewards as much as present rewards, which can lead to longer trajectories. When $\gamma < 1$, the agent places more emphasis on the long-term rewards and may take more steps initially to reach a state that yields higher rewards in the long run, which can lead to shorter trajectories.

We show this empirically in Fig. 14. We see that $\gamma = 1$ leads to a longer circuit than $\gamma < 1$. Furthermore, adding the $-\lambda$ term to penalize longer sequences does not further reduce the circuit size. For all experiments, we do not include the $-\lambda$ term and set $\gamma = 0.99$.

Appendix F: List of stabilizer generators
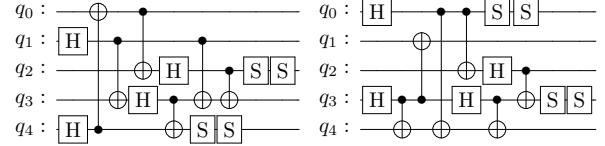
TABLE II. Here we list the stabilizer generators for the $[[5,1,3]]$ perfect code [106], $[[7,1,3]]$ Steane code [71], $[[9,1,3]]$ Shor code [107], $[[9,1,3]]$ Surface-17 code [17], $[[15,1,3]]$ quantum Reed-Muller code or 3D color code [112], and the $[[17,1,5]]$ 2D color code [74]. For the logical operators, we choose Z_L to be $Z^{\otimes n}$ and X_L as $X^{\otimes n}$, where n is the number of physical qubits of the respective code.

$[[5,1,3]]$ perfect	$[[7,1,3]]$ Steane	$[[9,1,3]]$ Shor	$[[9,1,3]]$ Surface-17
IXZZX	ZIZIZIZ	ZZIIIIIII	ZIIZIIIII
XZZXI	XIXIXIX	ZIZIIIIII	IIIZZIZZI
ZZXIX	IZZIIZZ	XXXXXXIII	IZZIZZIII
ZXIXZ	IXXIIXX	IIIZZIIII	IIIIIZIIZ
	IIIZZZZ	IIIZIZIII	IXXIIIIII
	IIIXXXX	XXXIIIXXX	XXIXXIIII
		IIIIIIIZZI	IIIIXXIXX
		IIIIIIIZIZ	IIIIIIXXI

$[[15,1,3]]$ Reed-Muller	$[[17,1,5]]$ 2D Color
ZIZIZIZIZIZIZIZ	XXXXIIIIIIIIIIIII
XIXIXIXIXIXIXIX	ZZZZIIIIIIIIIIIII
IZZIIZZIIZZIIZZ	XIXIXXIIIIIIIIIII
IXXIIXXIIXXIIXX	ZIZIZZIIIIIIIIIII
IIIZZZZIIIIZZZZ	IIIIXXIIXXIIIIIII
IIIXXXXIIIIIXXXX	IIIIZZIIZZIIIIIII
IIIIIIIZZZZZZZZ	IIIIIIXXIIXXIIIII
IIIIIIIXXXXXXXX	IIIIIIZZIIZZIIIII
IIZIIIZIIIZIIIZ	IIIIIIIIXXIIXXIII
IIIIZIZIIIIIIZIZ	IIIIIIIIZZIIZZIII
IIIIIZZIIIIIIIZZ	IIIIIIIIIIXXIIXXI
IIIIIIIIIIZZIIZZ	IIIIIIIIIIZZIIZZI
IIIIIIIIIIIIZZZZ	IIIIIIIXIIIXIIIXX
IIIIIIIIIZIZIZIZ	IIIIIIIIZIIIZIIIZZ
	IIXXIXXIIXXIIXXII
	IIZZIZZIIZZIIZZII
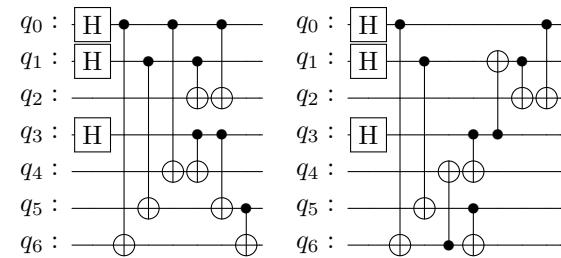
Appendix G: Logical state preparation circuits for all-to-all qubit connectivity with standard gate sets

We show some examples of learned circuits for all-to-all qubit connectivity with the standard gate sets. For all circuits shown below, we choose $Z_L = Z^{\otimes n}$. These circuits are also available online [103].
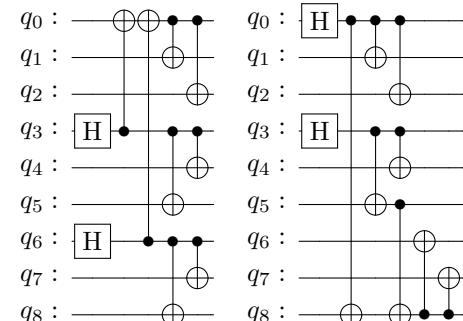
The $|0\rangle_{\mathbf{L}}$ state of the $[[\mathbf{5},\mathbf{1},\mathbf{3}]]$ perfect code. We see that the prepared circuit has a pattern of two S gates at the end of the circuit. This is because the RL agent would first aim to get the correct stabilizer generators without worrying about the sign and then apply Z gates (which can be decomposed into two S gates) to fix the sign.
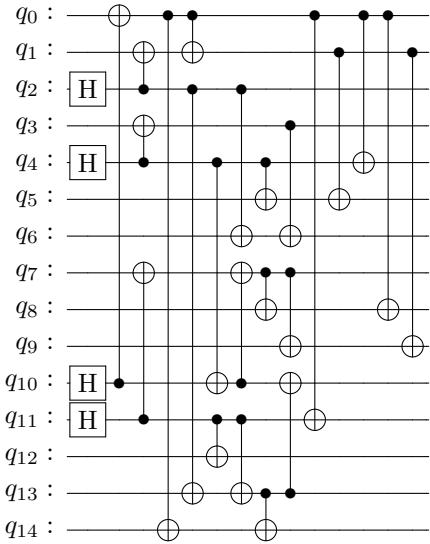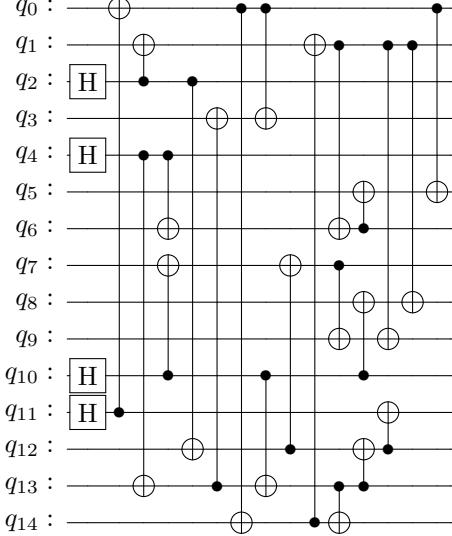
The $|0\rangle_{\mathbf{L}}$ state of the $[[\mathbf{7},\mathbf{1},\mathbf{3}]]$ Steane code. We see that the RL agent learns from scratch to prepare part of the initial state of the physical qubits to $|+\rangle$ by applying an H gate, similar to the strategy in [83]. We find that we can exploit this observation by using the following alternative strategy to speed up the training process. We first select a random subset of physical qubits to $|+\rangle$ and then allow the agent to apply gates other than H.
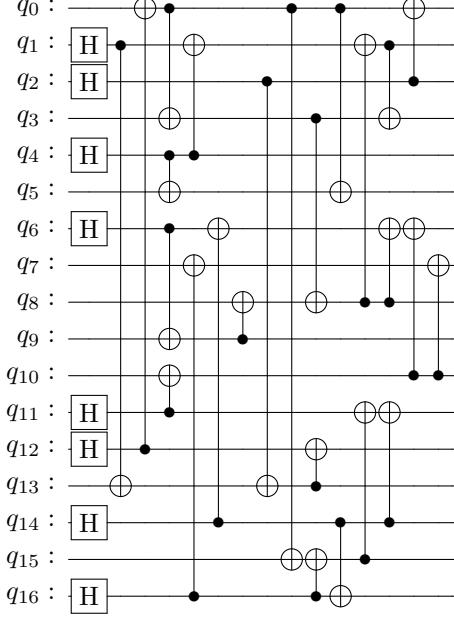
The $|+\rangle_{\mathbf{L}}$ state of the $[[\mathbf{9},\mathbf{1},\mathbf{3}]]$ Shor code.

The $|0\rangle_{\mathbf{L}}$ state of the $[[\mathbf{15},\mathbf{1},\mathbf{3}]]$ Reed-Muller or distance-3 3D color code.

The $|0\rangle_{\mathbf{L}}$ state of the $[[\mathbf{17}, \mathbf{1}, \mathbf{5}]]$ 2D color code

Appendix H: Transfer learning for different logical states
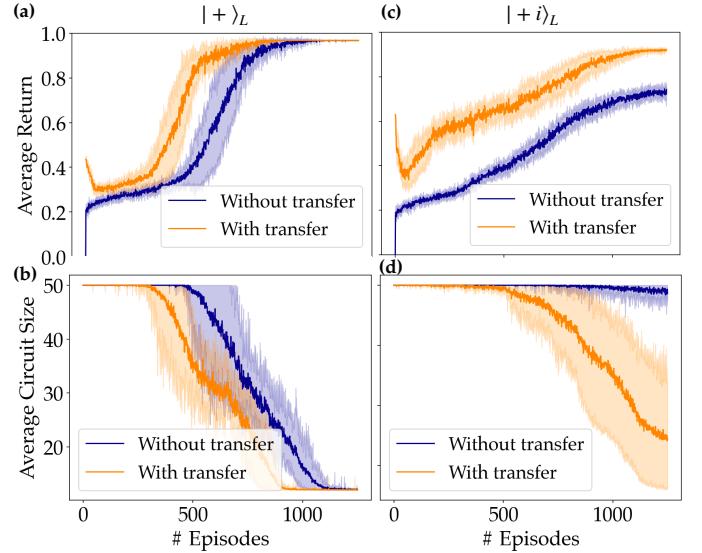
FIG. 15. Transfer learning results for different logical state preparation tasks. We first train the agent to prepare the $|0\rangle_L$ for the $[[7, 1, 3]]$ Steane code with $Z_L = Z^{\otimes 7}$ and then reuse and retrain the agent to prepare $|+\rangle_L$ with $X_L = X^{\otimes 7}$ and then successively $|+i\rangle_L$ with $Y_L = -Y^{\otimes 7}$. (a) and (b) show the average return and circuit size of training without and with transfer learning to prepare $|+\rangle_L$. (c) and (d) show the same for the preparation of $|+i\rangle_L$. The shaded area shows the standard deviation of training 10 different agents.

Here, we show an application of transfer learning where

we can reuse an RL agent trained on one logical state to prepare another logical state with the same code. One might argue that we can apply the logical H gate to the $|0\rangle_L$ state to prepare the $|+\rangle_L$ state, and then apply the logical S gate to prepare $|+i\rangle_L$ state. However, in some codes, the logical H and S gates themselves are not transversal.

We illustrate this by reusing the agent trained to prepare the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code to prepare the states $|+\rangle_L$ and $|+i\rangle_L$ of the same code. In particular, this transfer learning is called a one-to-one policy transfer via policy reuse [120]. In this case, we can directly reuse the networks of the RL agent since the number of input and output nodes does not change.
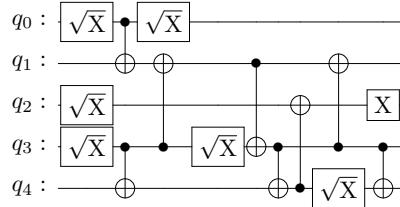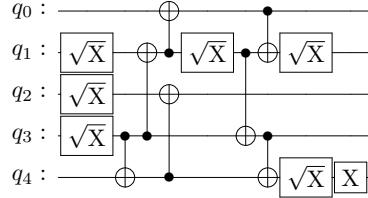
We compare the preparation of the $|+\rangle_L$ and the $|+i\rangle_L$ state of the $[[7,1,3]]$ Steane code without and with transfer learning in Fig. 15. Without transfer learning means that the RL agent is trained from scratch to prepare the states. On the other hand, with transfer learning means that we first train the RL agent to prepare $|0\rangle_L$, and then retrain it to prepare $|+\rangle_L$, and consecutively retrain it to prepare $|+i\rangle_L$.

Fig. 15(a) and 15(b) show the evolution of the average return and the average circuit size evolution during training for the preparation of $|+\rangle_L$ with and without transfer learning. Fig. 15(c) and (d) show the same but for the preparation of $|+i\rangle_L$. Here, we qualitatively compare the performance based on the metrics used in [120] to evaluate transfer learning for deep reinforcement learning. The first metric is the jumpstart performance, which is the initial return value of the agent. We can see that the average return of the RL agent with transfer learning is higher than without. The second metric is the time to threshold, which is the learning time required for the agent to reach a certain performance. We also see qualitatively that the value of the average return with transfer learning is almost always above without transfer learning. Therefore, we have shown that the proposed transfer learning technique is useful to efficiently prepare different logical states.
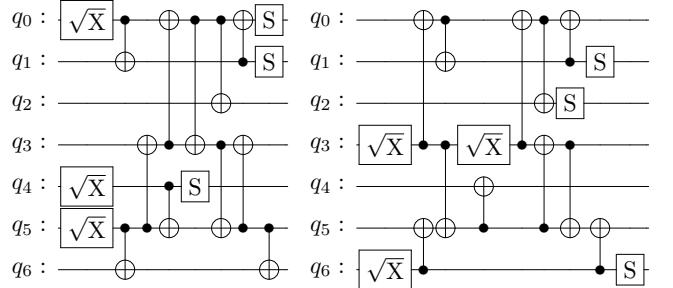
Appendix I: Logical state preparation circuits for IBM Quantum devices

We show here more circuit examples for logical state preparation based on the IBM Quantum device connectivity and gate set. The qubit placement is also given according to the notation in the Qiskit library [118]. If the qubit placement is given as $a, b, c, \ldots$, then it means that qubit 0 (q_0) in the circuit is placed in qubit a on the device, qubit 1 (q_1) is placed in qubit b on the device, and so on. These circuits are also available online [103].
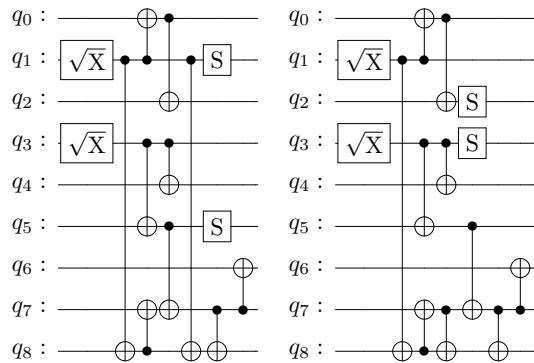
The $|0\rangle_L$ state of the $[[5,1,3]]$ perfect code on the IBMQ Manila [113] connectivity. The qubit placement is 4,3,0,2,1.
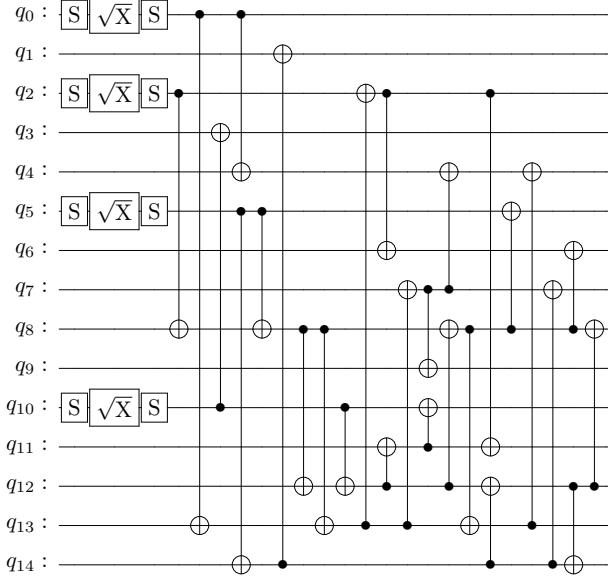
The $|0\rangle_{\mathbf{L}}$ state of the $[[\mathbf{7},\mathbf{1},\mathbf{3}]]$ **Steane code** on the IBMQ Jakarta [114] connectivity. The qubit placement is 5,6,4,3,0,1,2.

The $|+\rangle_{\mathbf{L}}$ state of the $[[\mathbf{9},\mathbf{1},\mathbf{3}]]$ **Shor code** on the IBMQ Guadalupe [115] connectivity. The qubit placement is 5,3,8,7,6,4,0,1,2.

The $|0\rangle_{\mathbf{L}}$ state of the $[[\mathbf{15},\mathbf{1},\mathbf{3}]]$ **Reed-Muller code** on the IBMQ Tokyo [116] connectivity. The qubit placement is 5,17,2,9,10,13,1,11,7,16,4,3,8,6,12.

FIG. 16. Results of transfer learning for logical state preparation on IBM quantum devices. We first train the agent to prepare $|0\rangle_L$ in a setting of all-to-all qubit connectivity, and then reuse and retrain it to prepare the same $|0\rangle_L$ under a different, restricted qubit connectivity. (a) The sketch of the transfer learning on the network trained on four qubits with all-to-all qubit connectivity to square connectivity by removing the output nodes corresponding to invalid actions. (b) The training evolution of preparing $|0\rangle_L$ of the $[[5, 1, 3]]$ perfect code with $Z_L = Z^{\otimes 5}$ on all-to-all qubit connectivity and transferring it to IBMQ Lima [130] connectivity and without transfer. (c) The training evolution of preparing $|0\rangle_L$ of $[[7, 1, 3]]$ Steane code with $Z_L = Z^{\otimes 7}$ on all-to-all qubit connectivity and transferring it to IBMQ Jakarta [114] connectivity and without transfer.

Appendix J: Transfer learning to different connectivity

We show a transfer learning technique that can reuse and retrain the agent trained to prepare a logical state for qubit connectivity G to prepare the same state with different qubit connectivity G'. We assume that G' is a spanning subgraph (having the same number of qubits but only some of the edges) of G.

Since we transfer to a connectivity that is a spanning subgraph of the original connectivity, the possible action space in the new connectivity is a subset of the possible action space in the original connectivity. This is equivalent to removing some of the output nodes in the actor network that correspond to invalid actions in the new connectivity. The input nodes, hidden nodes, and the value network remain the same and can be transferred directly. After the transfer, we use this network as the initial network and fine tune it for the new connectivity.

We illustrate this by choosing a case where RL agents trained for all-to-all qubit connectivity are transferred to a more restricted connectivity. For example, the sketch of the transfer learning technique from four qubits with all-to-all qubit connectivity to a new connectivity where the connections between qubit 1 and 3 and the connections between qubit 2 and 4 are removed is shown in Fig. 16(a). Assuming that we are using CNOT gates, we need to remove the output nodes of the actor network corresponding to the following actions: $\mathrm{CNOT}(1, 3), \mathrm{CNOT}(3, 1), \mathrm{CNOT}(2, 4)$, and $CNOT(4, 2)$, and keep the others. We then fine-tune this network to prepare the state for the new connectivity.

We compare the preparation of $|0\rangle_L$ of the $[[5, 1, 3]]$ code on IBMQ Lima [130] connectivity and $[[7, 1, 3]]$ code on IBMQ Jakarta [114] connectivity without and

with transfer learning. Without transfer learning means that the networks are randomly initialized at the start, while with transfer learning means that we reuse networks that were trained for all-to-all qubit connectivity. Fig. 16(b) and (c) shows the average return during the training of the $[[5, 1, 3]]$ perfect code and the $[[7, 1, 3]]$ Steane code with and without transfer learning. We see that with transfer learning, the average return initially starts a little bit higher, and more importantly, converges faster than without transfer learning. The agents without transfer learning need much more training time for convergence.

Appendix K: Varying the verification circuit synthesis task reward weights

Here, we vary the weights for the flag reward μ_f, the complementary distance reward μ_d, and the product state reward μ_p of the reward function that is defined in Eq. (4) for the verification circuit synthesis task. We then evaluate how this affects the acceptance and logical error rates.

Effectively, only the weight ratios matter, since scaling the reward function generally does not affect the performance of the reinforcement learning training. We vary μ_f/μ_d and μ_p/μ_d and synthesize the verification circuit at each point. We then compute the acceptance and logical error rates and fit them with the exponential and quadratic functions, respectively. We then compare the average coefficients over 10 different circuits.
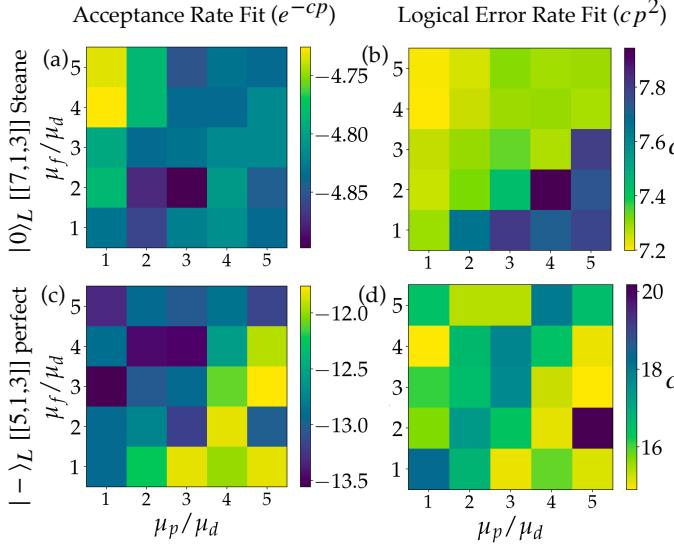

FIG. 17. Varying the weights of the reward function for verification circuit synthesis. We vary the μ_f/μ_d and μ_p/μ_d ranging from 1 to 5 with an interval of 1. The heatmap shows the average fitting coefficients for the acceptance ((a) and (c), the higher the better) and the logical error rate ((b) and (d), the lower the better). We evaluate for the verification circuit synthesis from the non-FT $|0\rangle_L$ of the $[[7,1,3]]$ Steane code taken from [32] ((a) and (b)) and $|-\rangle_L$ of the $[[5,1,3]]$ perfect code taken from [29] ((c) and (d)).
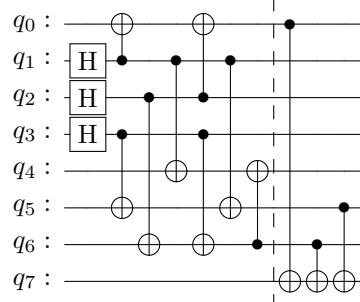
In Fig. 17(a) and (b), we see that the best strategy for the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code is to prioritize the weight of the flag reward μ_f. In contrast, one needs to prioritize the weight of the product state reward μ_p for $|-\rangle_L$ of the $[[5,1,3]]$ perfect code, as can be seen in Fig. 17(c) and (d).

Appendix L: Examples of RL-discovered circuits for the verification circuit synthesis task

Here we show more examples of RL-discovered circuits for the verification circuit synthesis task. We are particularly interested in showing the ability of RL to explore and present variants of circuits that minimize the fit coefficients of the acceptance or logical error rate. These circuits are also available online [103].

We first show the synthesis of the verification circuit for the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code by taking the non-FT logical state preparation circuit from Ref. [32]. First, we show that the RL method also rediscovers the
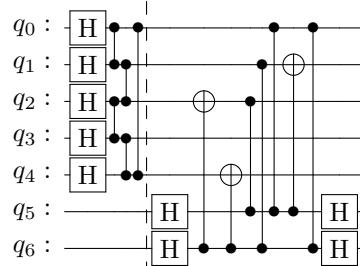
same circuit used in Ref. [32] shown below, which measures the stabilizer Z logical operator $ZIIIIZZ$:
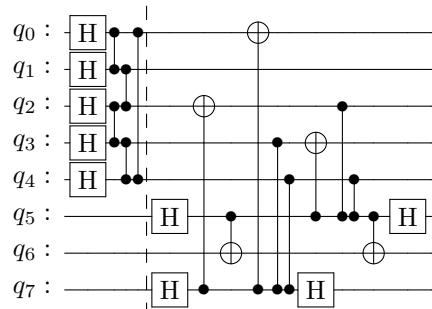
The circuit with the lowest fitting coefficients of the acceptance and logical error rate is the circuit shown in Fig. 6(a), which measures the stabilizer Z logical operator $IIZIZZI$. The other circuits are just permutations of the CNOT gates in the verification circuit.

We now show the synthesis of the verification circuit for the $|-\rangle_L$ state of the $[[5,1,3]]$ perfect code, using the non-FT logical state preparation circuit from Ref. [29].

The circuit with the lowest acceptance rate fitting coefficients is the circuit shown in Fig. 6(b). Below we show another circuit with two flag qubits and similar fitting coefficients.
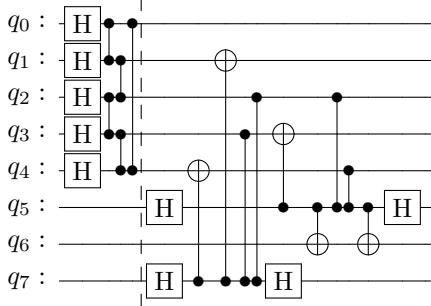
The circuit with the lowest fitting coefficients of the logical error rate is the circuit shown below. The circuit needs 3 flag qubits. The acceptance and logical error rate fitting coefficients are -14.1 and 12.1, respectively. One of the interesting properties that the RL agent learned is that it uses an extra flag qubit (second flag qubit) to fault-tolerantly measure stabilizer logical X operators $IIZXZ$ following the protocol in Ref. [29] in the first flag qubit and $XIXZZ$ in the third flag qubit.

We show another circuit with three flag qubits and similar fitting coefficients. The RL agent learned to measure
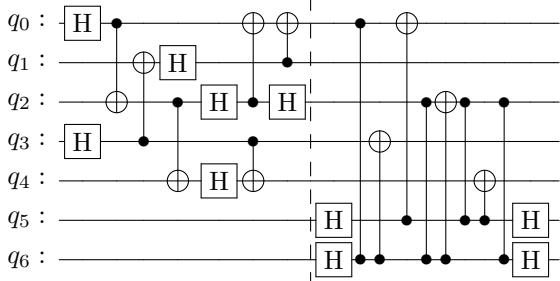
the stabilizer operators $IIZXZ$ in the first flag qubit and $IXZZX$ in the third flag qubit.
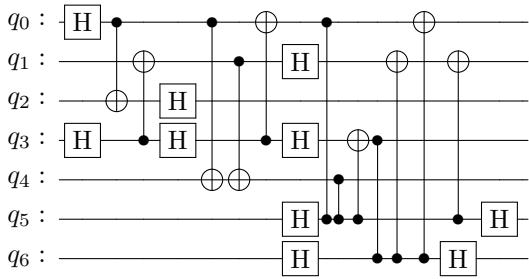
Appendix M: Examples of RL-discovered fault-tolerant logical state preparation circuits shown in Table I

Here, we show circuit examples from Table I with two RL approaches: logical state preparation followed by verification circuit synthesis (LSP + VCS) and integrated fault-tolerant logical state preparation (IFT-LSP). These circuits are also available online [103].
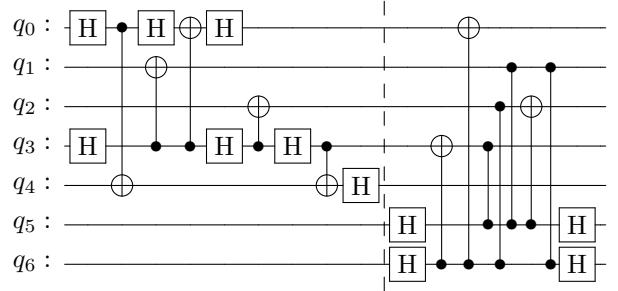
The $|1\rangle_{\mathbf{L}}$ state of the $[[5, 1, 3]]$ perfect code. The circuit obtained with the LSP + VCS approach has 14 two-qubit gates and 2 flag qubits:
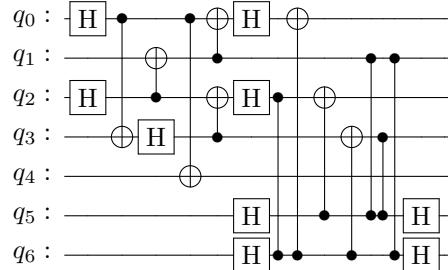
The circuit obtained using the IFT-LSP approach has a smaller number of only 12 two-qubit gates and 2 flag qubits:
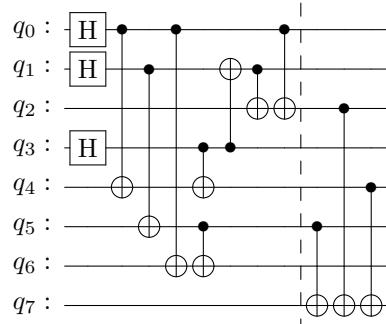
The $|-\rangle_{\mathbf{L}}$ state of the $[[5, 1, 3]]$ perfect code. The circuit with the LSP + VCS approach has 12 two-qubit gates and 2 flag qubits:
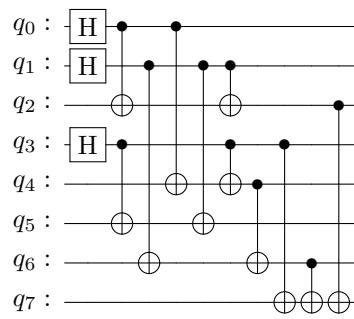
The circuit with the IFT-LSP approach has 12 two-qubit gates and 2 flag qubits:
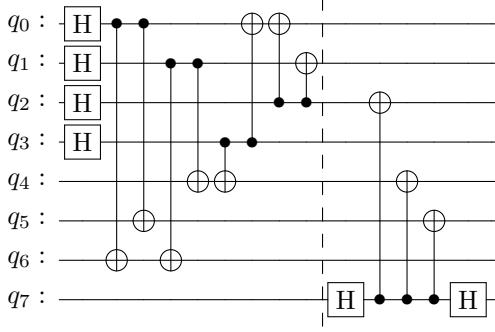
The $|0\rangle_{\mathbf{L}}$ state of the $[[7, 1, 3]]$ Steane code. The circuit found with the LSP + VCS approach has 11 two-qubit gates and 1 flag qubit:
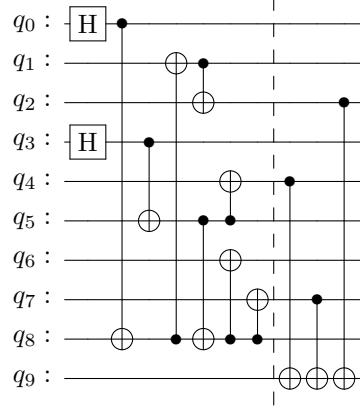
The circuit learned with the IFT-LSP approach has 11 two-qubit gates and 1 flag qubit, while using a smaller number of single-qubit gates:
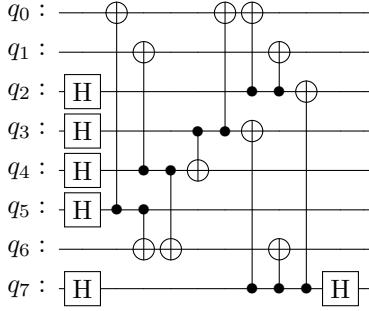
The $|+\rangle_{\mathbf{L}}$ state of the $[[7, 1, 3]]$ Steane code. The circuit learned with the LSP + VCS approach has 11 two-qubit gates and 1 flag qubit:
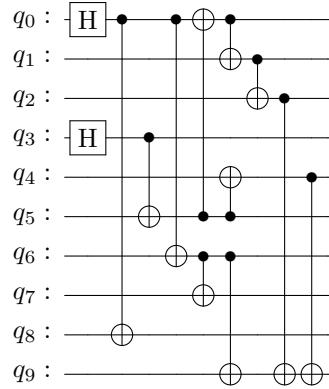
The circuit found with the IFT-LSP approach has 11 two-qubit gates and 1 flag qubit:
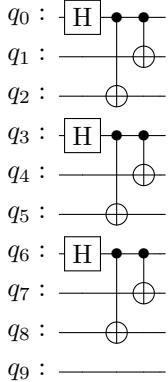
The circuit learned using the IFT-LSP approach has again 11 two-qubit gates and 1 flag qubit:
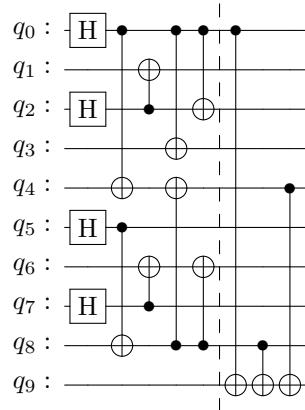
The $|0\rangle_{\mathbf{L}}$ state of the $[[\mathbf{9, 1, 3}]]$ Shor code. Both approaches discover the known fault-tolerant state preparation circuit that does not require any flag qubits at all, shown below.
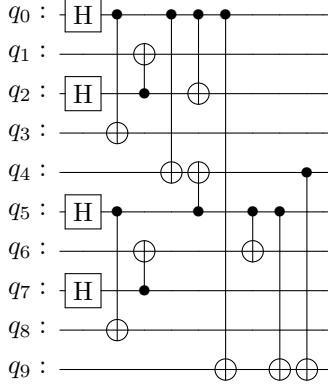
The $|0\rangle_{\mathbf{L}}$ state of the $[[\mathbf{9, 1, 3}]]$ Surface-17 code. The circuit learned with the LSP + VCS approach has 11 two-qubit gates and 1 flag qubit:
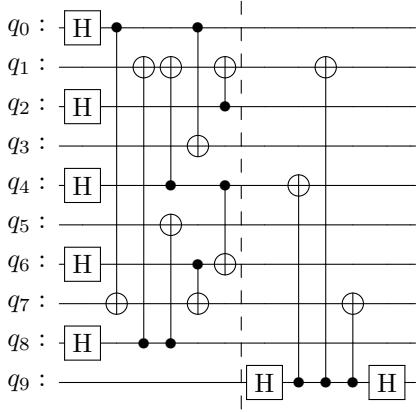
The $|+\rangle_{\mathbf{L}}$ state of the $[[\mathbf{9, 1, 3}]]$ Shor code. The circuit learned with the LSP + VCS approach has 11 two-qubit gates and 1 flag qubit:
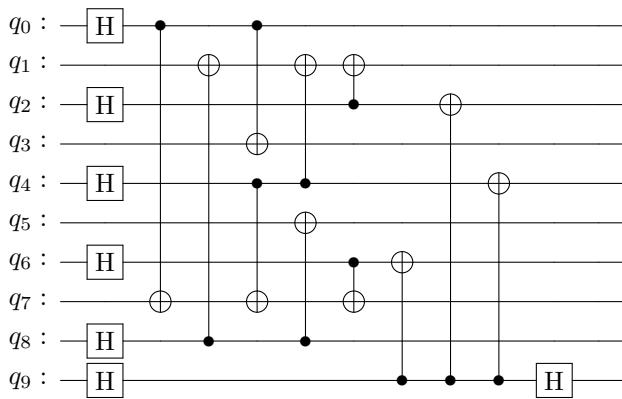
The circuit learned using the IFT-LSP approach has again 11 two-qubit gates and 1 flag qubit:
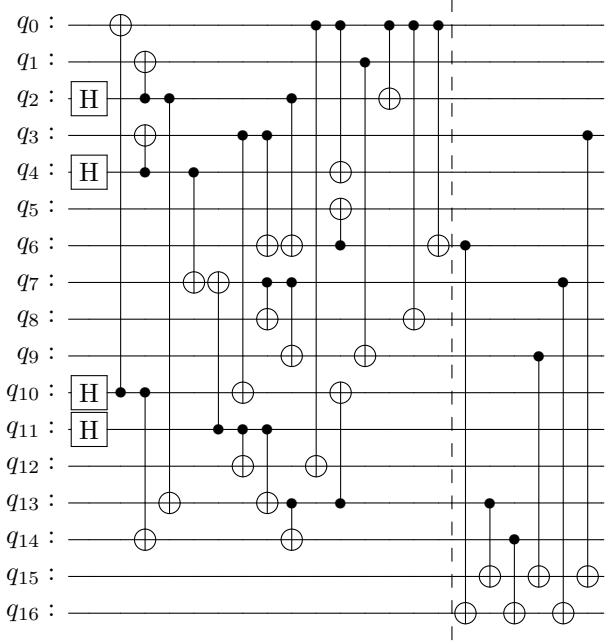
The $|+\rangle_{\mathbf{L}}$ state of the $[[\mathbf{9}, \mathbf{1}, \mathbf{3}]]$ Surface-17 code. The circuit learned with the LSP + VCS approach has 11 two-qubit gates and 1 flag qubit:
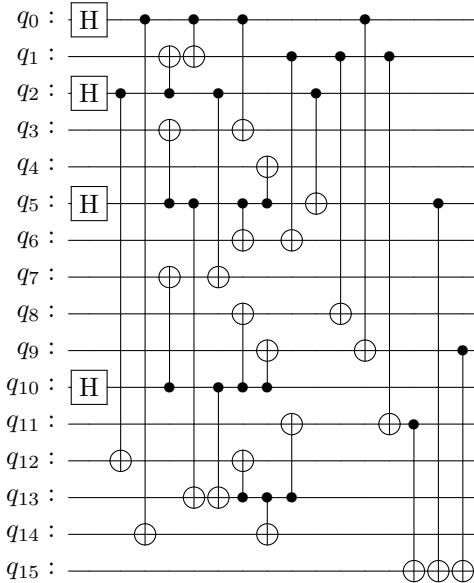
The circuit learned using the IFT-LSP approach has 11 two-qubit gates and 1 flag qubit:
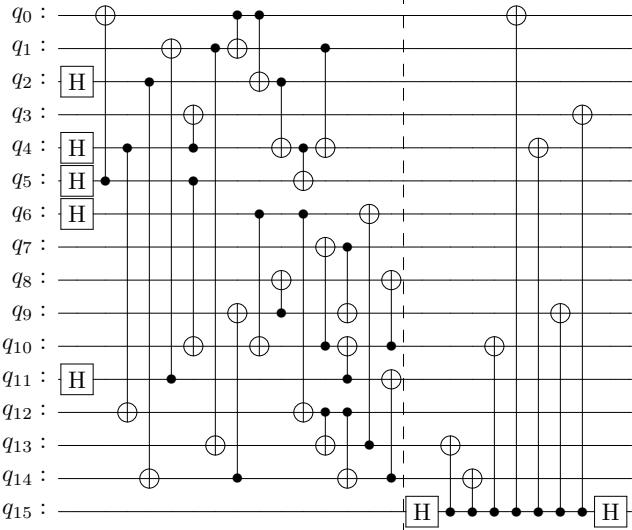
The $|\mathbf{0}\rangle_{\mathbf{L}}$ state of the $[[\mathbf{15}, \mathbf{1}, \mathbf{3}]]$ Reed-Muller code. The circuit found based on the LSP + VCS approach has 29 two-qubit gates and 2 flag qubits:
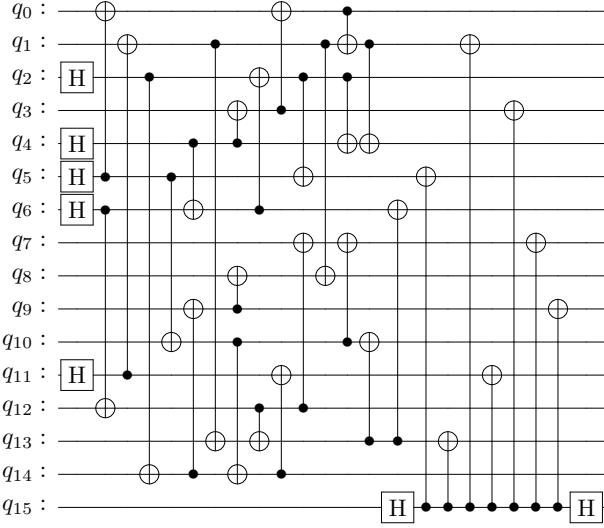
In contrast, the circuit obtained with the IFT-LSP approach has 25 two-qubit gates (instead of 29) and only requires 1 flag qubit (instead of 2), illustrating the superior performance of the IFT-LSP over the LSP + VCS approach:

The $|+\rangle_{\mathbf{L}}$ state of the $[[\mathbf{15}, \mathbf{1}, \mathbf{3}]]$ Reed-Muller code. The circuit found with the LSP + VCS approach has 31 two-qubit gates and 1 flag qubit:
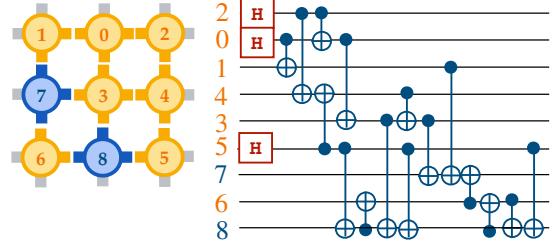
The circuit found with the IFT-LSP approach requires also 31 two-qubit gates and 1 flag qubit:

Appendix N: Examples of when LSP+VCS fails with restricted connectivity

Here, we discuss two cases where logical state preparation (LSP) followed by verification circuit synthesis (VCS) to prepare a fault-tolerant logical state would fail in restricted connectivity settings. Fig. 18(a) shows a case where a data qubit (qubit 6 in Fig. 18(a)) is only connected to the flag qubits, and the circuit on the right is the output of the integrated fault-tolerant logical state preparation (IFT-LSP) task. If we separate the task, the logical state preparation task fails to prepare the state in this case. Fig. 18(b) shows a case where if we separate the task, then the verification circuit synthesis fails, while the circuit on the right is the RL-prepared circuit with the fault-tolerant logical state preparation task. If
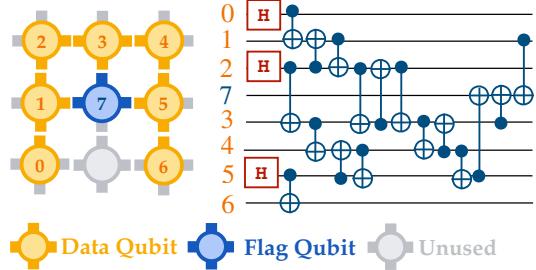
FIG. 18. Two cases where the LSP+VCS approach fails, but the IFT-LSP approach succeeds with restricted connectivity. We use the IFT-LSP approach to fault-tolerantly prepare the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code on a 2D grid. (a) A case where the data qubit is only connected to the flag qubits. (b) A case where VCS does not find a verification circuit.

we separate the task, then the state preparation does not know where the ancilla is. Therefore, the verification circuit synthesis fails to flag all of the harmful errors.

Appendix O: Logical error and acceptance rate of circuits found by integrated fault-tolerant logical state preparation
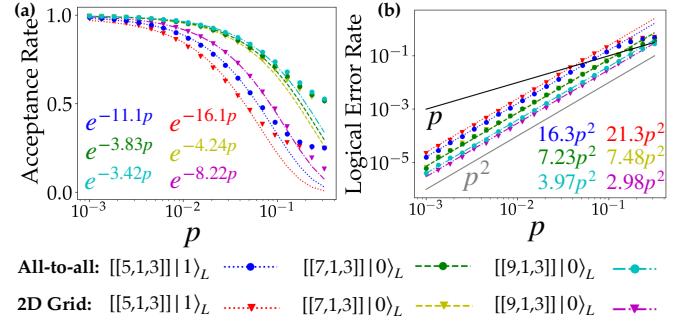

FIG. 19. Logical error rate and acceptance rate of circuits found with integrated fault-tolerant logical state preparation. We take the circuits shown in Fig. 8. The circuits for $[[9,1,3]]$ Surface-17 in all-to-all qubit connectivity is shown in Appendix M.

When using a restricted connectivity, we would expect a trade-off in the state acceptance and logical error

rate compared to using all-to-all qubit connectivity. In Fig. 19(a) and (b), we quantify and compare these two rates of circuits shown in Fig. 8. We see that the state acceptance rate of the circuit to prepare the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code on a 2D grid is only marginally lower than the one for the circuits with all-to-all qubit connectivity, but both are higher than the acceptance rate of the verification circuit synthesis task in Fig. 6(c). Similarly, the logical error rate is only marginally higher. In the case of the $|1\rangle_L$ for the $[[5,1,3]]$ perfect code, the circuit with restricted connectivity requires 5 two-qubit gates more than the circuit shown with full connectivity. Nevertheless, the logical error rate is larger only by approximately 25%. Interestingly, for the $|0\rangle_L$ of the $[[9,1,3]]$ Surface-17 code, the logical error rate is higher in the 2D grid connectivity than in the all-to-all qubit connectivity, but the acceptance rate is lower. We suggest that this has to do with the number of flag qubits. In the 2D grid, we need 4 flag qubits, while we only need 1 flag qubits in the all-to-all qubit connectivity case. We have also observed similar phenomenon in Appendix L, where using more flag qubits decrease the acceptance rate but increase the logical error rate. The relationship between the number of flag qubits, number of gates, acceptance, and logical error rate of different circuits is an interesting avenue for further study, but beyond the scope of this paper. Nevertheless, in all cases the logical error rate scales as p^2, confirming that the circuits are fault-tolerant, as desired.

Appendix P: Circuits for fault-tolerant logical state preparation in restricted connectivity
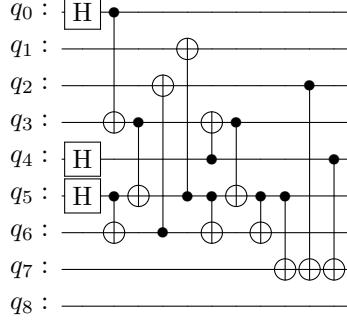
Here, we show examples of fault-tolerant logical state preparation circuits with restricted connectivity as shown in Fig. 9. If the qubit placement is given as $a, b, c, \ldots$, then it means that qubit 0 (q_0) in the circuit is placed in qubit a on the device, qubit 1 (q_1) is placed in qubit b on the device, and so on. These circuits are also available online [103].

1. The $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code on 2D grid connectivity (Google Sycamore)
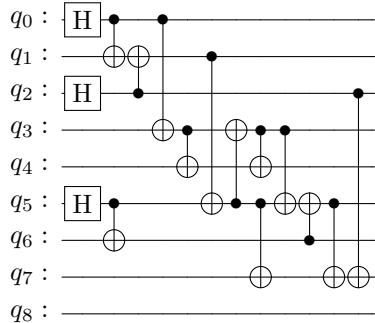
For the qubit placement, we follow the row-major order starting with 0 at the top left of the qubit and 9 at the bottom right of the qubit. The last two qubits are always given as ancilla qubits. The RL agent can decide whether to use them or not.

The circuit for the first qubit placement shown in Fig. 9(a) with 11 two-qubit gates is already shown in Fig. 8(c).
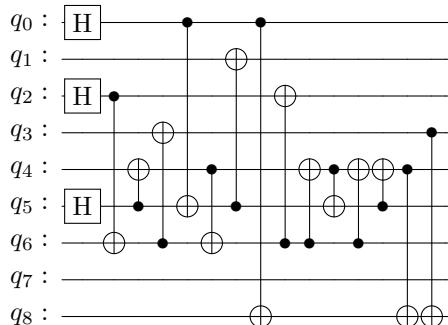
The circuit for the second qubit placement shown in Fig. 9(a) with 12 two-qubit gates. The qubit placement is: $2, 5, 6, 1, 0, 4, 7, 3, 8$.
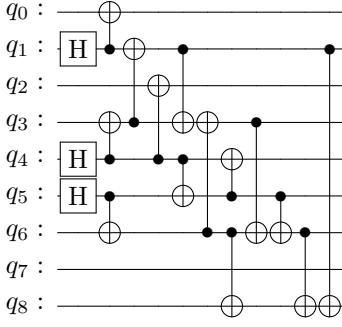

The circuit for the third qubit placement shown in Fig. 9(a) with 13 two-qubit gates. The qubit placement is: $2, 5, 8, 1, 0, 4, 3, 7, 6$.

The circuit for the fourth qubit placement shown in Fig. 9(a) with 14 two-qubit gates. The qubit placement is: $0, 2, 8, 6, 4, 1, 7, 5, 3$.
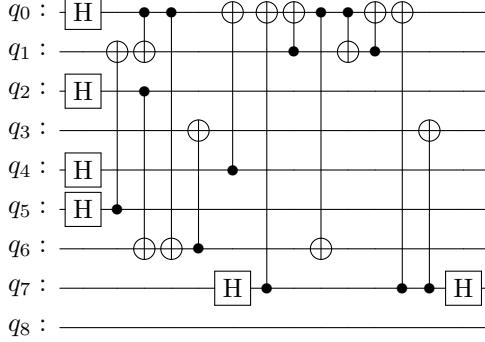

The circuit for the fifth qubit placement shown in Fig. 9(a) with 14 two-qubit gates. The qubit placement is: $2, 5, 0, 4, 3, 6, 7, 1, 8$.
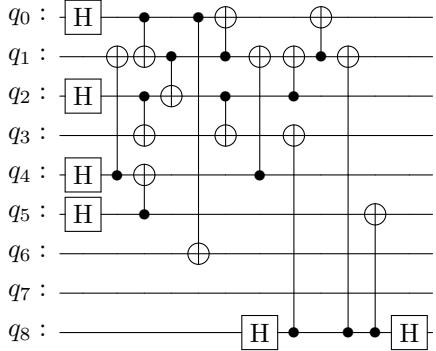
2. The $|+\rangle_L$ state of the $[[7,1,3]]$ Steane code on 2D grid connectivity (Google Sycamore)

The qubit placement for the circuit below is: 4,3,8,2,7,0,5,1,6.
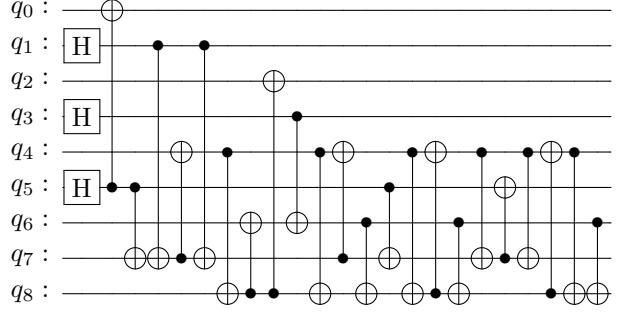
The qubit placement for the circuit below is: 5,4,7,6,1,0,8,2,3.
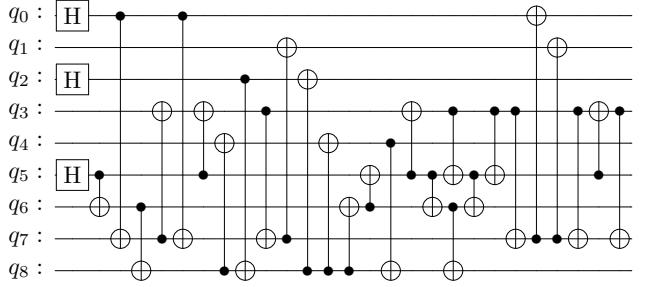
3. The $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code on heavy-hex connectivity (IBMQ Guadalupe)

The circuit for the first qubit placements shown in Fig. 9(b) has 22 two-qubit gates. The qubit placement in the IBMQ Guadalupe [115] device is: 3, 0, 6, 12, 4, 2, 10, 1, 7. The last two qubits are given as ancilla qubits.
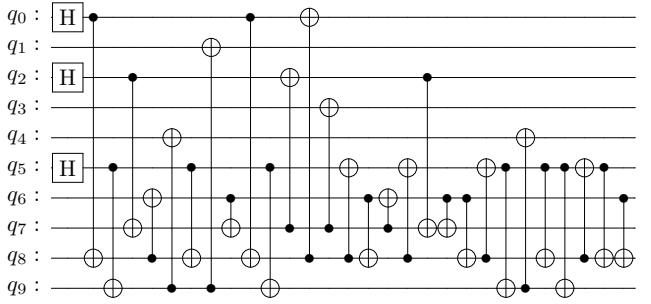
The circuit for the second qubit placement shown in Fig. 9(b) with 27 two-qubit gates. The qubit placement in the IBMQ Guadalupe [115] device is: 9, 5, 15, 11, 10, 14, 13, 8, 12. The last two qubits are given as ancilla qubits.

The circuit for the third qubit placement shown in Fig. 9(b) with 28 two-qubit gates. The qubit placement in the IBMQ Guadalupe [115] device is: 6, 15, 0, 2, 13, 10, 4, 1, 7, 12. The last three qubits are given as ancilla qubits.
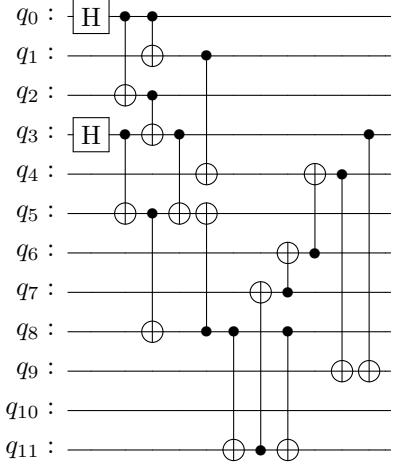

4. The $|+\rangle_L$ state of the $[[9,1,3]]$ Shor code on 2D grid connectivity (Google Sycamore)
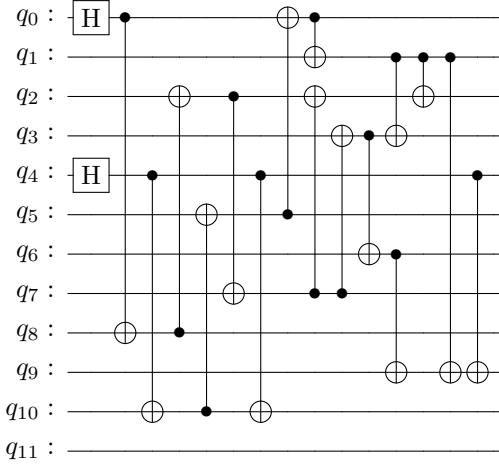
Here, we show the result for the fault-tolerant preparation of the $|+\rangle_L$ state for the $[[9,1,3]]$ Shor code on a 4×3 grid. For the qubit placement, we follow the row-major order starting with 0 at the top left of the qubit and 11 at the bottom right of the qubit. The last three qubits are always given as flag qubits. The RL agent can

decide whether to use them or not.

The qubit placement for the circuit below is: 1,0,2,5,3,8,6,9,11,4,7,10. One flag qubit is not used.
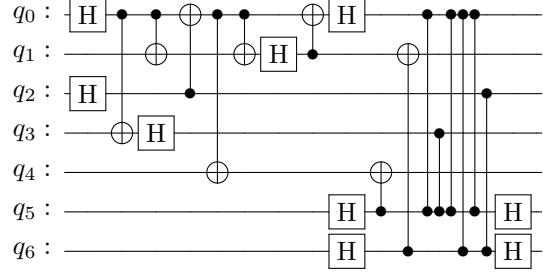
The qubit placement for the circuit below is: 6,7,10,8,1,3,5,11,9,4,0,2. One flag qubit is not used. In this case, one data qubit has the flag qubit as its neighbor, so the agent needs to use the flag qubit as a bridge to that data qubit.
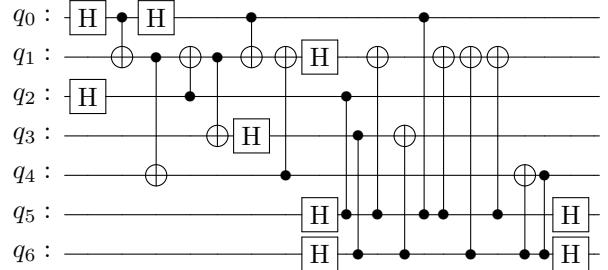
5. The $|-\rangle_L$ state of the $[[5,1,3]]$ perfect code on IBMQ Tokyo

The qubit placement for the circuit below on the IBMQ Tokyo [116] connectivity: 6,1,7,5,11,2,10.

The qubit placement for the circuit below on the IBMQ Tokyo [116] connectivity: 2,7,1,12,13,6,8.

Appendix Q: Varying integrated fault-tolerant logical state preparation task weight rewards

Here, we vary the weights for the flag reward μ_f, the complementary distance reward μ_d, and the product state reward μ_p of the reward function that is defined in Eq. (4) for the integrated fault-tolerant logical state preparation task (IFT-LSP). We then evaluate how it affects the acceptance and the logical error rates.

Effectively, only the weight ratios matter, since scaling the reward function generally does not affect the performance of the reinforcement learning training. We vary the ratios μ_f/μ_p and μ_d/μ_p and prepare the fault-tolerant logical state at each point. We then compute the acceptance and logical error rates, and fit them with exponential and quadratic functions, respectively. We then compare the average coefficients over 10 different circuits.

In Fig. 20, we see that the best strategy for the integrated fault-tolerant logical state preparation task is to prioritize the weight of the complementary distance reward μ_d. This is expected since the RL training starts from scratch, so μ_d must be prioritized.
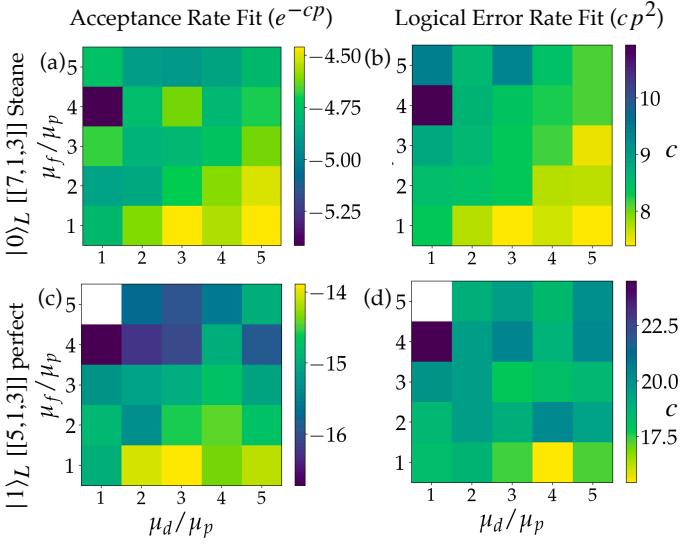
FIG. 20. Varying the weights of the reward function for integrated fault-tolerant logical state preparation (IFT-LSP). We vary μ_f/μ_p and μ_d/μ_p ranging from 1 to 5 with an interval of 1. The heatmap shows the average fitting coefficients for the acceptance rate (in (a) and (c), the higher the better) and the logical error rate (in (b) and (d), the lower the better). We evaluate for integrated fault-tolerant logical state preparation of the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code (in (a) and (b)) and the $|1\rangle_L$ state of the $[[5,1,3]]$ perfect code (in (c) and (d)). The white color means that no agent has converged on that parameter.