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*

(Received 17 May 2024; revised 7 March 2025; accepted 11 September 2025; published 22 October 2025)

The realization of large-scale quantum computers requires not only quantum error correction but also fault-tolerant (FT) 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 FT quantum circuit with flag-based protocols, especially when we consider hardware constraints, such as the qubit connectivity and available gate set. This work presents a novel approach to automatically discover compact and hardware-adapted FT quantum circuits to make significant progress towards scalable FT quantum computing. We employ reinforcement learning (RL) as an enabling tool, leveraging a fast, parallelized stabilizer quantum circuit simulator and a nontrivial reward function specifically adapted to the problem. We show that, in the task of FT logical state preparation, RL discovers not only circuits with fewer gates and ancillary qubits than published results but also novel circuits without and with hardware constraints of up to distance-5 codes with 25 physical qubits, and they can be implemented directly in experiments. Furthermore, RL allows for straightforward exploration of different qubit connectivities and the use of transfer learning to accelerate the discovery. More generally, our work sets the framework towards the use of RL or other machine learning techniques for FT quantum circuit discovery with hardware constraints to make real progress towards the realization of large-scale quantum computers, addressing tasks beyond state preparation, including magic state preparation, logical gate synthesis, and syndrome measurement.

DOI: 10.1103/gqpr-dgz7 Subject Areas: Quantum Physics, Quantum Information

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 breakeven point of beneficial QEC [12–14].

QEC operations are often expressed using a sequence of quantum gates that form a quantum circuit. However, these gates are faulty, and multiqubit gates proliferate the errors, compromising the scalability of QEC. In general, the more gates, the more errors, making QEC less effective [15–17]. Therefore, we want to minimize the number of possible faulty operations that can lead to harmful errors: This goal 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 can we ensure systematic improvement in correction as the size of the code scales.

*Contact author: remmy.zen@mpl.mpg.de

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 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 and 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 this process 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 to automatically discover hardware-adapted FT quantum circuits for QEC. Hardware adapted means that we can constrain the qubit connectivity and the available gate set based on the quantum platforms of interest, such that the discovered circuits can be directly realized in experiments. Our method leverages RL 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, as illustrated in Fig. 1, our approach is based on the automatic discovery of quantum circuits that fault tolerantly prepare the target logical state of a given QEC code under given available gate set and qubit connectivity.

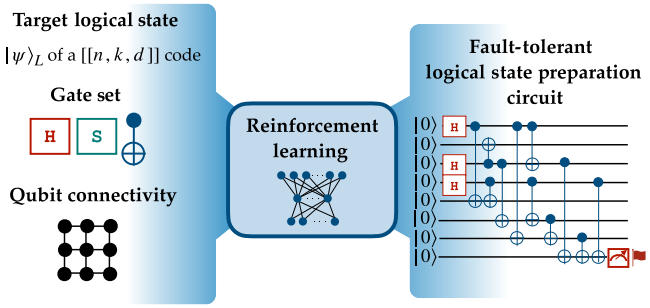


FIG. 1. Discovery of fault-tolerant logical state preparation circuit with reinforcement learning (RL). Given the target logical state $|\psi\rangle_L$ of a specified $[[n,k,d]]$ code, a gate set, and a qubit connectivity, we use RL to automatically discover circuits for preparing $|\psi\rangle_L$ fault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

Recently, 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. However, these works were more focused on proof-of-principle demonstrations that do not yet constitute experimentally relevant progress when judged on their own merits, i.e., when setting aside the novelty of using RL. In contrast, our work pioneers the use of RL for the automated discovery of FT quantum circuits that can be directly experimentally realized, thus making real progress towards FT scalable quantum computers.

FT quantum circuit design is also a more complex task than circuit compilation, circuit synthesis, or state preparation, requiring a more sophisticated approach to constructing an appropriate reward function. Unlike these tasks, where a fidelity reward is often sufficient [43,64,65], our task involves not only preparing the correct logical state but also tracking potentially harmful errors to be detected and/or corrected by a QEC code while considering hardware constraints to ensure experimental feasibility. In addition, by focusing on the preparation of a logical state for QEC, we can move to a larger number of qubits. The state-of-the-art results with RL on other tasks are up to 5 qubits [69] for general unitary synthesis and 11 qubits for general stabilizer states [70], while in our results we can go up to 25 qubits.

In our case, the RL agent needs to find an optimal strategy by applying a discrete gate at each step, guided by reward signals. 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. The construction of a nontrivial reward function specifically adapted to the problem is one of the key enablers of our approach. In addition, we find that RL is capable of efficiently navigating large and complex quantum circuit spaces. To achieve this goal, and another one of the key enablers of our approach, we introduce the development of our own fast, parallelized quantum circuit simulator. Finally, RL enables efficient and flexible automated discovery through transfer learning, i.e., reusing trained RL agents for similar but different quantum circuit problems. This result is not possible, for example, in the method proposed in Ref. [71], where finding a fault-tolerant quantum circuit is framed as a satisfiability modulo theory (SMT) problem.

We test our method on several QEC codes: the distance-3 codes, which include the 5-qubit perfect code, the 7-qubit Steane code, the 9-qubit Shor and surface codes, and the 15-qubit Reed-Muller code; and the distance-5 codes, which include the 17-qubit and 19-qubit color codes, as

well as the 25-qubit surface codes. 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 compared to existing circuits. More interestingly, 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. Then, in Sec. VII, we discuss how to scale our approach to higher distance codes. In Sec. VIII, we discuss how our approach is already useful for the current hardware context and discuss possibilities for future scaling. In Sec. IX, we summarize our work and discuss further extensions.

II. BACKGROUND

A. Quantum error correction

Here, we briefly review basic concepts from stabilizer QEC codes and introduce the notation that will be used in this paper. Readers familiar with these concepts can 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 code called a stabilizer code [72]. 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, code words can be transformed into one another using the logical operators Z_L^i, X_L^i, with $i = 1, \ldots, k$, 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 nonidentity 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 either 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 [73,74]. Because of 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 code [17,75] and the color code [76,77]. 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.

Because of 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 [78]. 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 [72] remains the same. A canonical tableau can be obtained by applying Gaussian elimination to the tableau [78]. Thus, different 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 [78]. Despite its simplicity, finding a compact circuit is still not trivial [79,80]. Several methods have been proposed to prepare arbitrary stabilizer states [78,81–84]. Some methods have also been developed specifically for the preparation of logical states of stabilizer QEC codes [85–89]. 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 three 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 [3,6,8,86].

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 [86]. 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 errors. 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 [20,21,28,29,71,90–93]. 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 $p_L\sim p^2$.

It is important to note that FT circuits are designed independently of specific information about the device's noise processes because FT circuits are designed to detect and correct even the most unlikely, yet potentially harmful, errors, regardless of their low probability of occurrence. In this way, the FT circuit ensures that any errors that occur, other than logical errors, are either detected or corrected in the next error correction cycle, thus guaranteeing the reliability of the FT circuit. The logical error rate p_L of an FT circuit for a code of distance d is guaranteed to scale as $p_L\sim cp^{(d+1)/2}$, for low enough physical error rate p. This scaling is a fundamental result of the threshold theorem in quantum computing [15]. We emphasize that achieving this scaling in the quantum circuit design is essential to have the guarantee that the logical error rate will be suppressed according to the correcting power of the QEC code. To be precise, according to the threshold theorem [15], as long as the physical components have a sufficiently low error rate p, this FT circuit will produce states with higher fidelity than the corresponding non-FT operation. In summary, an FT circuit can be used in any device with the same hardware constraints (connectivity and gate set), independently of the details of the noise actually present in the device.

It is true that the FT model of quantum computing does not account for all possible sources of error, one example being leakage, which takes the qubit out of computational space. A technique to overcome leakage errors has been proposed separately in Ref. [94]. Nevertheless, several FT implementations of QEC [3,95] have shown that, despite lacking full knowledge of the underlying error processes, the logical error rate improves compared to physical and non-FT implementations of the same operation.

There is no unique way to render a circuit FT [3,21,28,71,91,96,97]. 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 qubits always have the same measurement outcome. When faults that lead to harmful errors occur in any component of the circuit, the flag qubits are triggered, i.e., flip the measurement outcome of the flag ancilla qubits. 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. After successful preparation, the resulting logical state can be used directly in downstream QEC tasks such as logical gates, QEC cycles, and logical readout. One can also reset the flag qubits and reuse them for other tasks.

D. 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 parametrized by θ, which maps each state of the environment to a probability distribution over all possible actions, and $\pi_\theta(a_t|s_t)$ 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 [98] 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, with θ corresponding 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) [99]. 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

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.

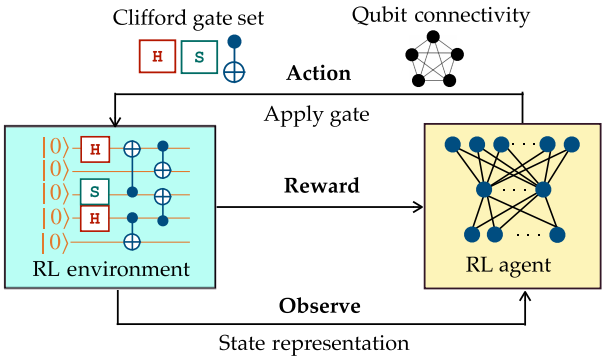


FIG. 2. General RL framework in this work. The circuit is the environment, where its state is represented by its stabilizer canonical tableau. At each step, the RL agent observes the environment and applies a discrete Clifford gate as an action from the specified available gate set (e.g., the Hadamard 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.

We must then choose a representation of the RL agent's observation. The most common representation is to directly observe the quantum circuit [100,101] or to observe the state vector of the state that the circuit represents [56,58,59,102]. 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 parametrized circuit and a variational approach as in Ref. [103]. 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 [104].

We use the PUREJAXRL library [105] for the implementation of the PPO algorithm, which is written with the JAX [106] library to allow very fast parallel training on a GPU. We then implement the environment for each task using JAX. Thus, 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 [107]. The details of the hyperparameters used and the training process are described in Appendix B. We discuss the scalability of our simulations in Appendix R.

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.

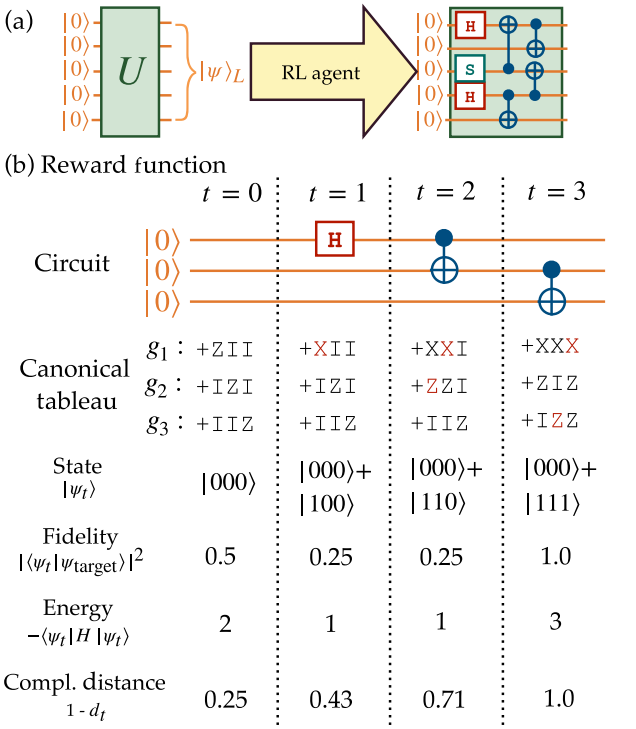


FIG. 3. Description and reward function for the logical state preparation task. (a) Logical state preparation task, which outputs a circuit U that prepares a target logical state $|\psi\rangle_L$ of a $[[n, k, d]]$ code. (b) Preparation of the state $|\psi_{\text{target}}\rangle = |000\rangle + |111\rangle$ (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 Ref. [103], 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.

A natural choice of the reward function is the fidelity of the state [56,57,59,102,108,109]. 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 $|\langle 000|111\rangle|^2 = |\langle 100|111\rangle|^2 = |\langle 110|111\rangle|^2 = 0$. Thus, the RL agent is harder to train because it does not receive 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. [103], 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. We can then 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 $E_0 = -n$. Reference [103] used E as a cost function for the variational optimization of a parametrized 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.

Figure 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 or energy functions, which is not good feedback to the RL agent. This is not the case for $1 - d_t$. Our empirical numerical experiments also show 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, in this case, 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, which is intrinsically embedded in the RL formulation, explained 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 [110], which has been realized experimentally, for example, in Refs. [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 [111], has been realized experimentally, for example, in Refs. [112,113]. We also consider 2D and 3D color codes [76,77]. The $[[7, 1, 3]]$ Steane code [73] is the smallest CSS and triangular 2D color code that has been realized experimentally, for example, in Refs. [3,6,37,114,115]. We also explore the distance-5 2D color code, which is the $[[17, 1, 5]]$ code [77]. Finally, we consider the smallest error-correcting 3D color code, the $[[15, 1, 3]]$ Reed-Muller code [76,116]. 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. Thus, 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].

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.* [81] and Aaronson-Gottesman [78]. We also compare with StabGraph [85], which works only for CSS codes and uses graph states, and QMAP [82], 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, using the depth as the optimization target,

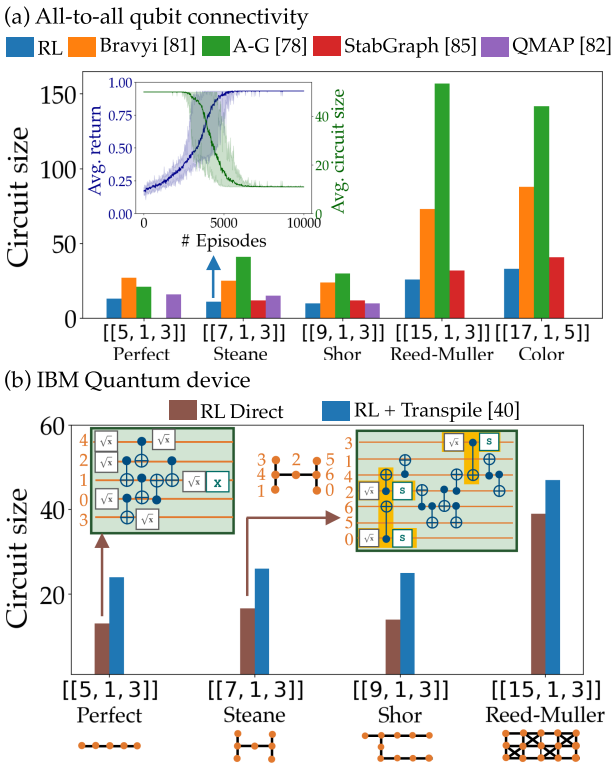


FIG. 4. Results for the logical state preparation task. (a) Minimum circuit size of different methods for logical state preparation of different QEC codes with all-to-all qubit connectivity and H, S, and CNOT gates. StabGraph [85] does not work for non-CSS codes such as the $[[5,1,3]]$ perfect code. QMAP [82] could not prepare the state of the $[[15,1,3]]$ and $[[17,1,5]]$ codes 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 are shown for various IBM Quantum device connectivities [119–122] 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 code and the $[[7,1,3]]$ Steane code.

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.

Figure 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 prepares a smaller circuit size compared to the other methods. StabGraph [85] is specialized in preparing logical states of CSS codes; therefore, it does not work for the $[[5,1,3]]$ perfect code. QMAP [82] 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 ten agents trained in parallel, where each agent sees 16 environments in parallel. The entire training takes approximately 100 seconds on a single NVIDIA Quadro RTX 6000 GPU and produces ten circuits. On average, the ten 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 find the correct tableau without worrying about the sign and then use 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, which is commonly referred to as transfer learning [123,124]. 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 (parametrized 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 [119] connectivity, the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code on the IBMQ Jakarta [120] connectivity, the $|+\rangle_L$ state of the $[[9,1,3]]$ Shor code on the IBMQ Guadalupe [121] connectivity, and the $|0\rangle_L$ state of the $[[15,1,3]]$ quantum Reed-Muller code on the IBMQ Tokyo [122] connectivity. These connectivities are shown at the bottom of Fig. 4(b). We train 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]. Figure 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. Thus, we see 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 performed 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 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* 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.

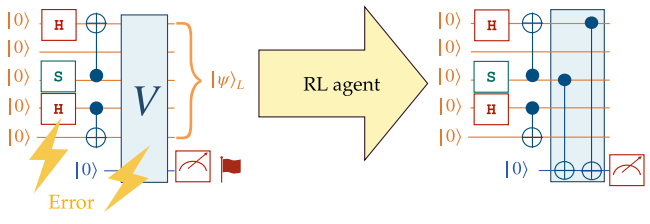


FIG. 5. Verification circuit synthesis task preparing a circuit V that uses flag qubits to flag harmful errors, thereby rendering a state preparation fault tolerant.

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 the data (nonancilla) qubits to be 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

$$f_t = \sum_{E\in\mathcal{E}} \begin{cases} 0 & \text{if } E \text{ is } I \text{ and a flag is triggered} \\ 1 & \text{if } E \text{ is tolerable} \\ 1 & \text{if } E \text{ is harmful and a flag is triggered} \\ 0 & \text{if } E \text{ is harmful and flags are not triggered.} \end{cases} \tag{3}$$

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. We discuss the scalability of this approach in Appendix R.

One might notice that if an error E is tolerable and the flag is triggered, the agent still receives a reward, which 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 approach does not compromise FT, of course. One could add additional terms in the reward function to minimize this result.

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 process 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 the 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

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

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 [125] library, and we 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. Thus, the execution of the syndrome extraction circuit is noiseless and is followed by decoding and correction. Throughout this paper, we use a lookup table decoder. This decoder assigns a fixed correction to each possible syndrome. Therefore, it does not account for the differing error probabilities introduced by each encoding circuit. Using a more sophisticated decoder, such as minimum-weight perfect matching (MWPM) in the case of surface codes, could improve the logical error rate of a given encoding circuit. However, we do not expect it to significantly alter the relative logical error rates between different encoding circuits. 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 proportionally 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. We illustrate this process 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 Ref. [32] [part of the circuit in Fig. 6(a) shaded in green] and experimentally realized in Refs. [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 Ref. [32]. Part of the circuit in Fig. 6(a), 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 Figs. 6(c) and 6(d) that the acceptance rate and the logical error rate of the RL-discovered circuit are marginally better than the verification circuit proposed in Ref. [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 Ref. [29] and experimentally realized in Refs. [6,7]. The blue-shaded part of the circuit in Fig. 6(b) 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 Ref. [29] due to the smaller

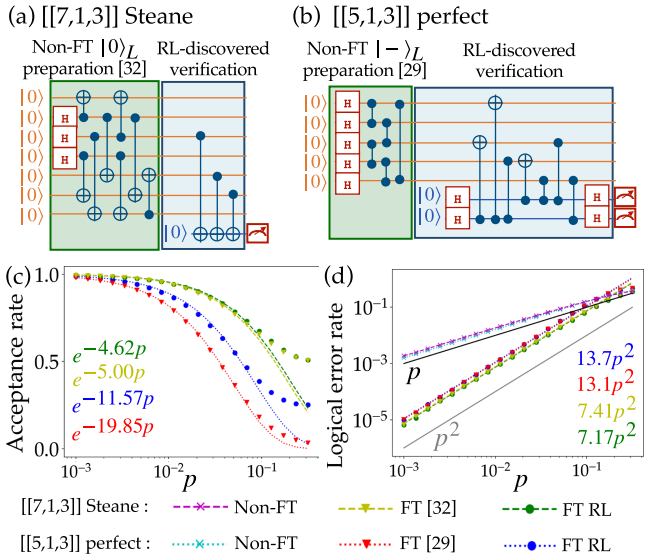


FIG. 6. Results for the verification circuit synthesis task. Examples of RL-discovered verification circuits (shaded in blue) for a given non-FT preparation (shaded in green) of (a) the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code from Ref. [32] and (b) the $|-\rangle_L$ state of the $[[5,1,3]]$ perfect code from Ref. [29]. In Ref. [32], the verification circuit uses one flag qubit and three two-qubit gates, which is the same as the circuit discovered by the RL agent. In Ref. [29], the verification circuit uses six flag qubits (or two flag qubits with two qubit resets) and 15 two-qubit gates, while the RL-discovered circuit in panel (b) uses only two flag qubits and seven 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 panels (a) and (b) compared to non-FT circuits and circuits in Refs. [29,32].

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 Ref. [29]. However, the RL agent also discovers circuits with a lower logical error rate than the circuit in Ref. [29], at the expense of requiring three flag qubits. We show this circuit in Appendix L.

One might argue that the RL results do not significantly outperform existing handcrafted circuits, which are close to optimal or already optimal. However, the discovery of optimized versions of some known FT circuits is only one part of our results. A more significant result that we would like to highlight is the discovery of novel, previously unknown circuits, especially those with limited connectivity that can be directly implemented in real devices, which we will present in the next section.

So far, we discuss the discovery of verification circuits for codes with $d = 3$. The discovered circuits, in some cases, can be used directly to scale to larger codes by means of encoding concatenation. We discuss the discovery of verification circuits for higher distance codes in more detail in Sec. VII and Appendix R.

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 trade-offs 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

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, which 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.

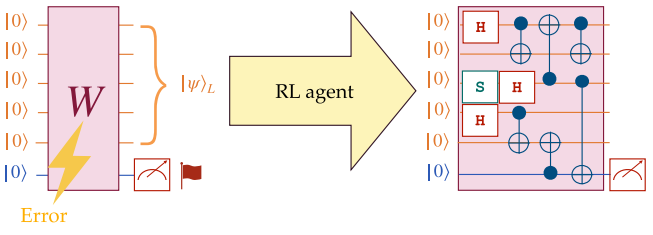


FIG. 7. Integrated fault-tolerant logical state preparation task, which outputs a circuit W that directly prepares $|\psi\rangle_L$ of a $[[n,k,d]]$ code in a fault-tolerant way.

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, which directly prepares the fault-tolerant logical state and which we refer to as IFT-LSP.

We discuss the preparation of the following logical states. For the CSS codes considered (i.e., the $[[7,1,3]]$ Steane code, the $[[9,1,3]]$ Shor code, and the $[[15,1,3]]$ 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 Refs. [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 two states: $|1\rangle_L$ of the $[[5,1,3]]$ perfect code and $|0\rangle_L$ of the $[[15,1,3]]$ Reed-Muller code. This finding 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 the $[[5,1,3]]$ perfect code and $|+\rangle_L$ of the $[[15,1,3]]$ 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.

Figures 8(a) and 8(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 operator $ZXIXZ$ via the first ancilla and $XXIZI$ via the second ancilla.

TABLE I. 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 LSP (in Sec. IV) followed by the VCS (in Sec. V). The second RL approach is our main approach, which is the 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]

[a]Reference [35] shows the FT preparation of the $|0\rangle_L$ state of the $[[9,1,3]]$ Surface-17 code with eight two-qubit gates and zero flag qubits. However, the connectivity of qubits is different, namely, a 3×3 grid without ancilla qubits.

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 process is not obvious, for example, in the $[[5,1,3]]$ perfect code. In Ref. [6], the $|-\rangle_L$ is always prepared fault tolerantly first; then, the logical basis state is rotated to another state. With our RL approach, instead, we can automatically discover fault-tolerant preparation circuits for other states.

2. Restricted qubit connectivity

While it is true that any optimization or discovery achieved through RL could, in principle, be performed manually by an expert with sufficient time and patience, we would like to emphasize the important point of scalability, not only in terms of qubits but also in terms of hardware constraints. We argue that the design and optimization of FT quantum circuits is even more nontrivial when hardware constraints, such as the available gate set or connectivity, are considered. Manual exploration of these configurations is very challenging, especially as the number of qubits grows. We will show that, even with relatively few qubits, our RL approach is able to discover novel FT logical state preparation with connectivity based on real devices.

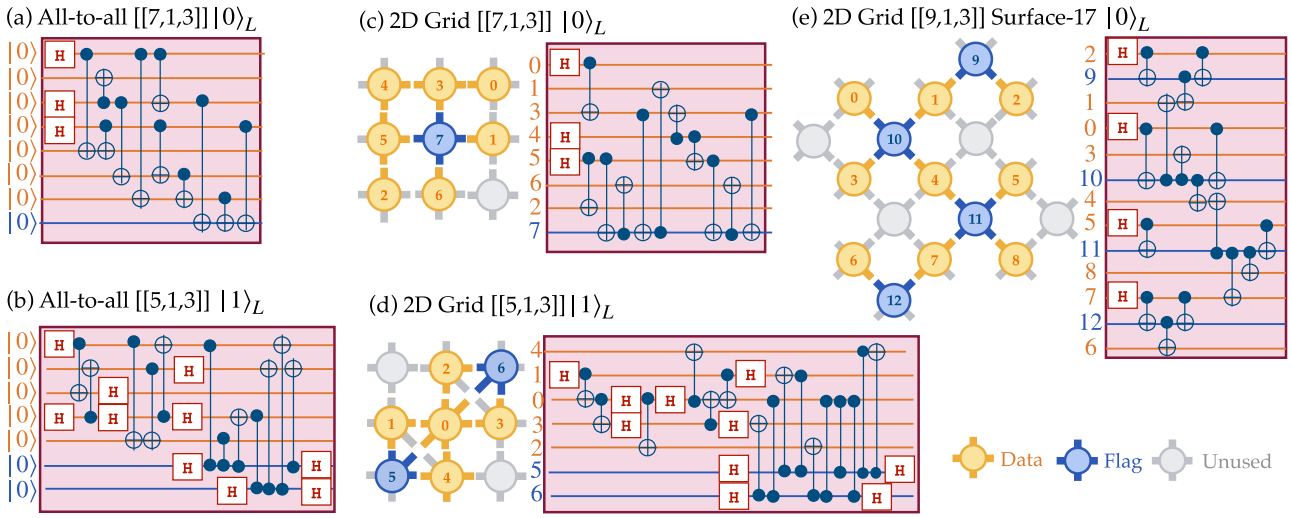


FIG. 8. Results for the RL-based integrated fault-tolerant logical state preparation (IFT-LSP). We show an example of a fault-tolerant circuit prepared by an RL agent for (a) the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code and (b) the $|1\rangle_L$ state of the $[[5,1,3]]$ perfect code in all-to-all 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 [126] (c) and IBMQ Tokyo [122] (d) devices. In panel (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 Ref. [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 they are not used in the solution found by the agent.

Therefore, 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 [127]. 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 under 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 [126], IBM Quantum devices, and 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. Figure 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). Figure 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 [122] device. Compared to the circuit on all-to-all qubit connectivity, it has the same number of flag qubits and requires only four 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 $|0\rangle_L$ ($|+\rangle_L$) of a surface code is commonly prepared fault tolerantly by using stabilizer measurements [128,129]—initializing the physical qubits in the product state $|0\rangle^{\otimes n}$ ($|+\rangle^{\otimes n}$) and measuring the X (Z) stabilizers, which projects the entire many-body state into a surface code state. The outcome of the stabilizer measurements is random even in the absence of error; thus, they cannot be used to flag harmful errors. On the other hand, flag-based FT preparation uses specially designed circuits with flag qubits to detect and address errors in a single round of measurements. Thus, when a qubit is flagged, we can either discard and reset the state or use the flag as additional syndrome information, providing a more versatile error management strategy [130].

Here, for the first time, we show RL-discovered flag-based fault-tolerant logical state preparation of a surface code in the standard 2D grid connectivity: We illustrate this process by preparing the $|0\rangle_L$ of the $[[9,1,3]]$ Surface-17 code with the connectivity and qubit placement from Ref. [5] in Fig. 8(e). The RL-discovered circuit, although given eight flag qubits to use, only needs four flag qubits and uses 16 two-qubit gates. It is only five two-qubit gates

more than preparing it in an all-to-all qubit connectivity as shown in Table I (the circuit is shown in Appendix M). The circuit was discovered in approximately 2000 seconds. The discovered circuit shows a novel approach to fault tolerantly prepare a logical Pauli state of the surface code.

Regarding the interpretation of the discovered FT circuit and its possible relation to the stabilizer measurement initialization scheme, we note that the four ancilla qubits used in the FT circuit found by the agent in Fig. 8(e) are, in fact, not measuring stabilizers. Instead, they are used as bridges to connect data qubits. To further clarify, since there are gates connecting ancilla and data qubits in the middle of the circuit (i.e., before the logical state preparation is completed), measuring these ancilla qubits is not equivalent to measuring the stabilizer operators of the given code. This finding can also be seen in the circuit shown in Fig. 8(c), where there is a CNOT with control and target in the flag qubit. Surprisingly, the RL agent can come up with an intelligent design that flags harmful gate errors, even though none of the surface code stabilizers are directly measured. The latter makes the circuit novel and different from the stabilizer measurement initialization scheme.

We now compare the flag-based scheme with the stabilizer measurement initialization scheme. The circuit for preparing the $|0\rangle_L$ of the $[[9,1,3]]$ Surface-17 code with the stabilizer measurement scheme is shown in Fig. 9(a). It uses 14 CNOTs, while the circuit discovered by RL uses 16 CNOTs. The CNOT gates in Fig. 9(a) are ordered to ensure FT, such that the hook X and Z errors are placed orthogonally to the X_L and Z_L operators, respectively, following Ref. [131]. However, we still see in Fig. 9(b) that the flag-based scheme performs better in terms of the logical error rate compared to the stabilizer measurement scheme. We hypothesize that this has to do with the postselection that the flag-based scheme performs to filter out preparations with harmful errors, while the stabilizer measurement scheme always accepts the state (100% acceptance rate). Nevertheless, the acceptance rate of the flag-based scheme is also good, as it is above 90% for $p < 3 \times 10^{-2}$, as shown in Fig. 21 in Appendix O.

In Appendix O, we compare the logical error and acceptance rates of the circuits shown in Fig. 8. In all cases, the logical error rate scales as p^2, confirming that the circuits are fault tolerant, as desired. In Appendix Q, we show how different values of the weight μ in the reward affect the acceptance and logical error rates.

Unlike manual design, RL can adapt to different hardware constraints without requiring significant re-engineering or manual intervention. We illustrate this case by showing that the RL approach allows a straightforward exploration of different qubit connectivity and placements, i.e., assignments of data and flag qubits to physical qubits of the underlying device, by training different RL agents. We illustrate this approach by preparing the $|0\rangle_L$ state for the $[[7,1,3]]$ Steane code. On a 3×3 grid based on the

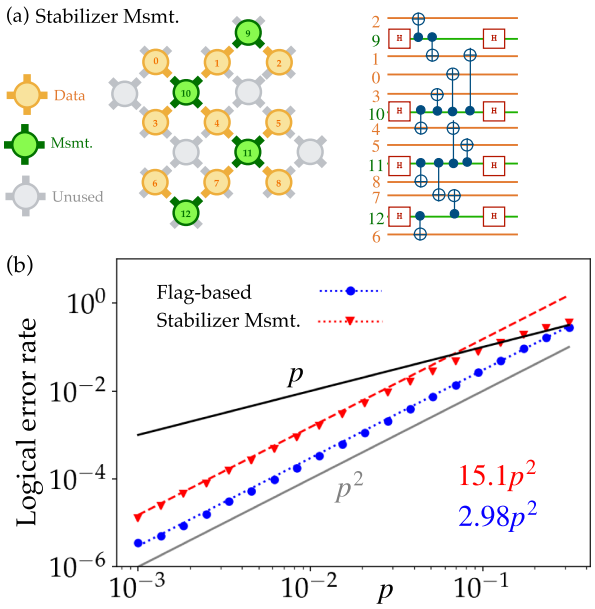


FIG. 9. Comparison between flag-based and stabilizer measurement initialization schemes for fault-tolerant preparation of the $|0\rangle_L$ of the $[[9,1,3]]$ Surface-17 code. (a) Circuit for stabilizer measurement scheme with 12 CNOTs to measure all of the X stabilizers. Note that the qubits in green are measurement qubits and not flag qubits. (b) Comparison of the logical error rate of the circuit shown in panel (a) and the flag-based circuit shown in Fig. 8(e). We see that the flag-based FT circuit has a lower logical error rate with a factor of 5 improvement compared to the stabilizer measurement scheme.

Google Sycamore device, there are $\binom{9}{7} = 36$ possible data and flag qubit placements. We train on all possible configurations, and in Fig. 10(a), we show five qubit placements where the circuit discovered by the RL agent has the lowest number of two-qubit gates. In this case, a new RL agent is trained from scratch for different qubit placements. A more efficient approach is to use transfer learning, which involves reusing a trained network for different placements. We will demonstrate this approach more generally later (see Fig. 18). Preparing the $|0\rangle_L$ of the $[[7,1,3]]$ Steane code on a 2D grid takes approximately 200 seconds, which is roughly 50 seconds slower than preparing the same state with all-to-all connectivity. Therefore, in about 2 hours (200 seconds times 36 possible qubit placements), our approach can discover the most efficient fault-tolerant preparation of the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code on a 2D grid. This process could be even faster if we train the agent in parallel or use a transfer learning approach. We argue that this is much faster than the time human experts (scientists or engineers) would require to perform a similar task. Training a single RL agent for one qubit placement takes about 200 seconds.

Next, we illustrate the same preparation for heavy-hex connectivity based on the IBMQ Guadalupe device. We show three data and flag qubit placements where the circuit

has the lowest number of two-qubit gates in Fig. 10(b). Training one agent takes approximately 1000 seconds. We have also tried different flag qubit placements, but sometimes the agent does not find an encoding circuit, especially when the flag qubit is not located in the crossing (i.e., the flag qubit is connected to three data qubits). This case may indicate that the best flag qubit placement in the heavy-hex connectivity is in the crossings. We provide the circuits in Appendix P.

One might expect that when the RL agent directly prepares a fault-tolerant logical state, it would try to detect some harmful errors in the middle of the preparation. This is indeed the case, for example, in the circuit shown in Fig. 8(c). However, we observe that most of the discovered circuits can be decomposed into a state preparation circuit U, followed by a verification circuit V [e.g., the circuits shown in Figs. 8(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, which we illustrate in Fig. 11. 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 [85]. 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. 11.

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 [123].

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 make the training converge faster. Additionally, we see that the transferred agent retains the strategy from the previous training. We illustrate this finding by comparing the integrated fault-tolerant preparation of the $|0\rangle_L$ state of

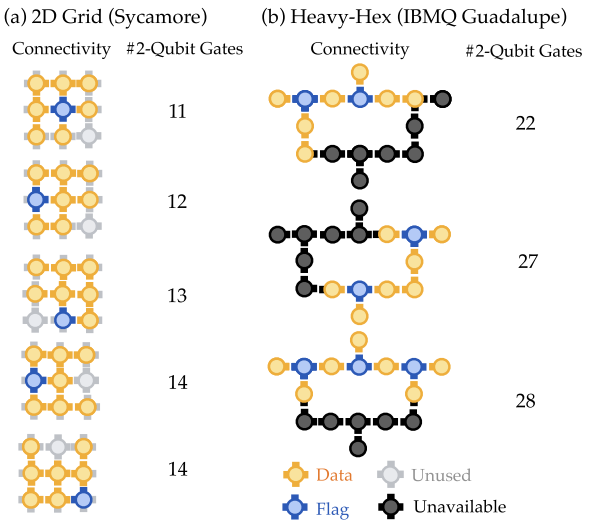


FIG. 10. Exploration of different qubit connectivity and placement for the integrated fault-tolerant $|0\rangle_L$ state preparation of the $[[7,1,3]]$ Steane code. We show the number of two-qubit gates in some of the RL-discovered circuits with (a) 2D grid (based on the Google Sycamore device) and (b) heavy-hex layout (based on the IBMQ Guadalupe [121] device). Unused qubits and connectivities (in gray) mean that the qubits were given to the RL agent to be used as flag qubits, but they were not used. Unavailable qubits and connectivities (in black) mean that the qubits are not set as available to the RL agent.

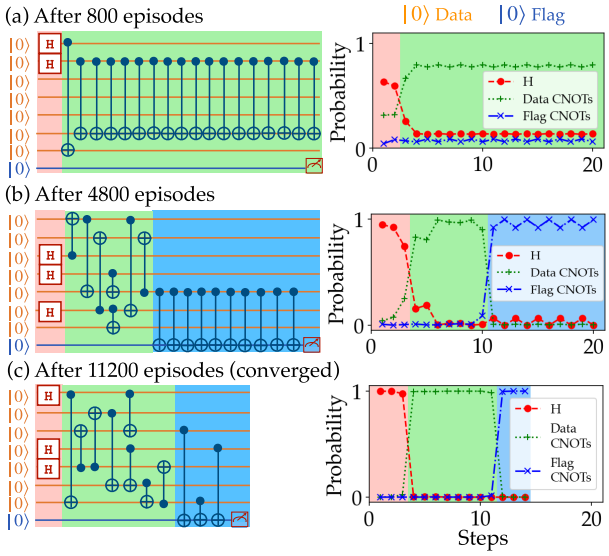


FIG. 11. Evolution of the learned strategy during training. We train an RL agent for the integrated fault-tolerant $|0\rangle_L$ state preparation of the $[[7,1,3]]$ Steane code, assuming all-to-all qubit connectivity. The left part of the figure shows the circuit prepared by the agent. The right part shows the probability of actions for each step. We group the actions into three main groups: applying Hadamard gates on some qubits (red), applying CNOTs between data qubits (green), and applying CNOTs between data qubits and flag qubits (blue). The background color indicates the most probable group of actions at that step. We can see the progression of the RL agent's learning process, starting from applying mostly Hadamard gates in the first few steps in panel (a), followed by learning how to prepare the logical state in panel (b), and finally learning how to prepare the state and flag the harmful errors after convergence in panel (c). We hypothesize that the agent applies long sequences of self-canceling CNOTs in panels (a) and (b) because it has not yet learned what to do in the later time steps. The agent then chooses a "safe" strategy by applying multiple CNOTs several times, which does not change the reward.

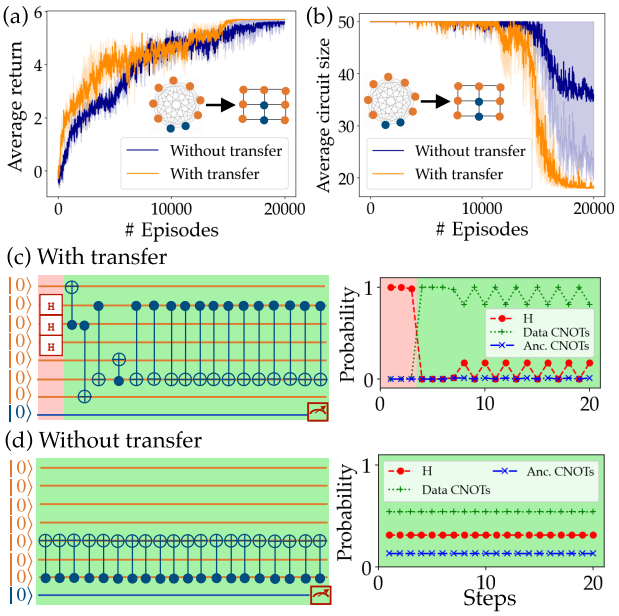


FIG. 12. Transfer learning for integrated fault-tolerant logical state preparation from an all-to-all qubit connectivity to a 2D grid connectivity. Panel (a) shows the average return, and panel (b) shows the circuit size during training for the fault-tolerant $|0\rangle_L$ preparation of the $[[7,1,3]]$ Steane code with and without transfer learning. The training without transfer learning also converges, but it requires more training. 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. 11. This finding is in contrast to the case without transfer learning shown in part (d), where the agent applies only a long sequence of self-canceling CNOTs. In this case, the agent has not learned anything, so the probability of each gate is still uniform. The CNOT gate between qubits 7 and 5 is just a random gate chosen by the agent.

the $[[7,1,3]]$ Steane code on a 2D grid without and with transfer learning from all-to-all qubit connectivity in Fig. 12.

In summary, we have shown that the 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. SCALING TO HIGHER DISTANCE CSS CODES

So far, we have shown results for $d = 3$ codes. In some cases, the discovered quantum circuits can already be scaled to higher distance codes by code concatenation. We discuss this approach in more detail in Appendix R 2; here, we focus on how to scale up beyond concatenation. One of the main obstacles in scaling up to higher distances in the approach described in Secs. IV–VI is to store a rapidly increasing number of errors. Here, we implement ways to mitigate this and other problems and, thereby, scale our approach to find circuits for higher distance codes from scratch.

In general, designing a FT circuit requires consideration of every type of error that can occur in the circuit. However, the important class of CSS codes provides a further simplification in the design of fault-tolerant schemes. Because of their construction, CSS codes split the correction of Pauli X and Z errors into independent processes, which implies that Y errors can also be treated as both X and Z errors happening in the same location, as well as treated independently. Furthermore, 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 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 errors are not harmful by construction. Note that the same would not be true for non-CSS codes because the stabilizers involve both the X and Z Pauli operators, so the logical operators consist of both X and Z operators.

Thus, to prepare the $|0\rangle_L$ or $|+\rangle_L$ states of CSS codes fault tolerantly, we only need to consider errors of type X or Z, respectively. Furthermore, some two-qubit gates do not convert errors into different types; for example, a CNOT gate propagates X errors into other X errors. Therefore, by avoiding the use of any single-qubit gate in the middle of the circuit, we can only consider one type of error generator. In fact, as shown in Fig. 11, this is the general strategy learned by the RL agent when trained from scratch. Specifically, one only needs to initialize some of the initial qubits to the $|+\rangle$ state using a Hadamard gate and then work exclusively with CNOT gates to prepare the logical state and synthesize the verification circuit. By restricting ourselves to this simple, yet generic strategy, the number of errors to be handled can be drastically reduced.

As mentioned above, one of the problems in scaling to higher distance codes is the memory scaling to store the errors $\mathcal{E}$. In Appendix R 3, we discuss in detail how the memory scales with distance. We show that, with this new strategy, we can fit errors for distance-9 CSS codes in a single GPU, whereas for the general strategy, it can only fit errors for distance-5 codes. We also show, in Appendix R 1, that the simulation time of $d = 5$ codes is only about 3 times slower than the one for $d = 3$ codes, and it still scales polynomially with n. These results are very promising, and they illustrate the scalability of our approach.

So far, our assumption is that one prepares the $|0\rangle_L$ and $|+\rangle_L$ states of CSS codes fault tolerantly using only CNOT or CZ gates. This approach already works for both our LSP + VCS and IFTLSP tasks with any qubit connectivities. However, one can make it even more efficient if one restricts oneself to LSP + VCS tasks. There are underlying principles for designing FT flag verification circuits, for instance, by measuring some logical or stabilizer operators [20,28,32]. However, there is no known recipe for which or how many stabilizer operators need to be measured. We can then design an RL agent for the VCS task where the action is to apply a sequence of gates that measure stabilizer operators fault tolerantly according to the protocol in Ref. [20] instead of applying a single gate. In this case, one only needs to use the f_t term in the reward in Eq. (4) since the measurement of an operator is guaranteed to preserve the state and to produce a product state between the data and ancilla qubits. Note that, in the worst case, the number of actions of the RL agent grows exponentially as 2^n. However, this number can be reduced because, in CSS codes, we only need to consider operators of type X or Z. Our strategy is to incrementally train the RL agents by including operators of increasing weights. For example, we first start the training by only including operators up to weight 5, and then 6, and so on.

By incorporating these additional insights into the training of the RL agent, we can scale and discover FT circuits for codes of higher distance. We show in Fig. 13(a) the number of two-qubit gates and flag qubits needed for fault tolerantly preparing different $d = 5$ codes. In Fig. 13(b), we plot the logical error rate of the discovered non-FT and FT circuits. In this case, we simulate 3×10^{10} noisy circuit trajectories for each varying error probability p to compute the logical error rate. While the logical error rates of the non-FT circuits scale as p or p^2, the fault-tolerant circuits scale as p^3, confirming that the circuits are fault tolerant, as desired. It takes approximately 4000 seconds to discover the circuit for the $[[17, 1, 5]]$ color code, 4500 seconds for the $[[19, 1, 5]]$ color code, and 8000 seconds for the $[[25, 1, 5]]$ surface code.

(a)

Code	[[17,1,5]] Color	[[19,1,5]] Color	[[25,1,5]] Surface
# Gates (Flag)	38 (3)	52 (5)	55 (4)

(b)

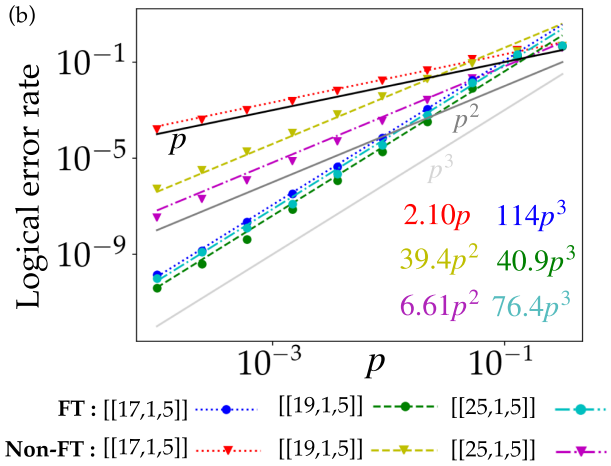


FIG. 13. Results for scaling to distance-5 CSS codes. (a) Number of two-qubit gates and flag qubits in the RL-discovered circuits that prepare the $|0\rangle_L$ state of different $d = 5$ codes fault tolerantly. (b) Logical error rate with different simulated error probabilities p for the RL-discovered distance-5 non-FT and FT circuits. The logical error rates of the FT circuits scale as p^3, confirming that the circuits are FT. The circuits are available in Appendix S.

VIII. CURRENT HARDWARE CONTEXT AND OPPORTUNITIES FOR FUTURE SCALING

We have shown the RL-discovered fault-tolerant quantum circuits for logical state preparation for several known codes. In Sec. VII, we have also shown how we can scale our approach to higher distance CSS codes. We would like to emphasize that our method is already useful and can be used directly in current quantum hardware, especially since our RL method takes the hardware constraints into account. While the total number of physical qubits in a given experiment can already be on the order of hundreds, it is important to realize that none of these recent experiments operate on a QEC code composed of hundreds of physical qubits but rather in a highly parallel manner on a number of logical qubits, each of them encoded in small or medium distance codes. To provide context, we summarize recent experimental results on multiqubit codes in Table II.

We can see in Table II that our RL method already reaches the same level and can be applied to the number of physical qubits comprising individual logical qubits as the current experiments. It is also important to recognize that the results in Table II mostly represent a very specific handcrafted circuit construction of the code, which benefits from a highly regular structure and well-studied techniques. In contrast, our approach addresses the broader challenge of automatically discovering and optimizing fault-tolerant circuits for more general QEC codes, where the structure and connectivity may be less regular and manual optimization becomes significantly more difficult. These and other future experiments can thus clearly benefit directly from our RL method in optimizing the quantum circuits used in the experiment.

In principle, there is even more room to further extend and scale our method to codes with hundreds of qubits. We briefly discuss some of the opportunities and possibilities here. Our RL approach can be further improved by restricting or modifying the action space as in Sec. VII or by designing a better reward function. One could consider developing modular formulations of the FT compilation tasks, e.g., by including more complex building blocks, such as subcircuits for specific tasks like

TABLE II. Recent experimental results using multiqubit codes. The $l \times [[n, k, d]]$ means that $l \times k$ logical qubits are realized with the $[[n, k, d]]$ code: if not specified, then $l = 1$.

Reference	Codes used
[132]	$[[9, 1, 3]]$, $[[25, 1, 5]]$, and $[[49, 1, 7]]$ surface codes
[10]	$40 \times [[7, 1, 3]]$ Steane code, $2 \times [[49, 1, 7]]$ surface code, and $16 \times [[8, 3, 2]]$ 3D color code
[133]	$[[24, 1, 4]]$ 4D surface code
[134]	$3 \times [[16, 4, 4]]$ tesseract code
[135]	$2 \times [[16, 1, 4]]$ 3CX code
[136]	$12 \times [[4, 2, 2]]$ Toric code, $28 \times [[4, 1, 2]]$ Toric code, and $[[9, 1, 3]]$ Bacon-Shor code

stabilizer readout, in the set of actions available to the RL agent, or by exploiting the symmetry (e.g., translational invariance) of the code construction. One can also consider a hierarchical approach to circuit design. For example, instead of directly targeting the circuit with large distance d, one can build the solution by targeting smaller distances first. In other words, we start by considering errors with probability p until a solution is found, then include p^2, and so on until we reach the target distance. In Appendix R 4, we also discuss the possibility of exploring the potential of advanced collaborative multiagent RL scenarios, which may allow one to apply the techniques proposed in this work to larger distances and concatenated error correction codes.

IX. CONCLUSIONS AND OUTLOOK

We have presented novel approaches to automatically discover compact and hardware-adapted FT quantum circuits based on flag qubit protocols relying on RL as an enabling tool. We illustrate our approaches for FT logical state preparation of QEC codes. Compared to other circuit synthesis tasks, our approach requires not only the preparation of the correct state but also the tracking of harmful errors to ensure FT, all while adhering to hardware constraints—a highly nontrivial challenge that would be difficult to achieve through manual handcrafted design. 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 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. We have also investigated and shown that transfer learning can help speed up the training process of the RL agent. Finally, by incorporating additional structural insights about CSS codes into the simulation of the environment and introducing innovations such as incremental training for the RL agent, we have scaled our approach to CSS codes with higher distance. All in all, we have demonstrated a scalable method for automatic discovery of novel, hardware-specific, and FT circuits with RL that can be directly implemented in experiments, bringing us closer to the realization of large-scale FT quantum computers.

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 discovery of quantum circuits for fault-tolerant magic state preparation [36,71], syndrome measurement [137,138], 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 verificationlike 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.

ACKNOWLEDGMENTS

We thank Sangkha Borah, Maximilian Nägele, Oleg Yevtushenko, Josias Old, Julio Carlos Magdalena de la Fuente, and Manuel Rispler for fruitful discussions. The research is part of the Munich Quantum Valley (K-4 and K-8), which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus. L. C. and M. M. further acknowledge support by the U.S. Army Research Office through Grant No. W911NF-21-1-0007. M. M. also acknowledges support by the European Union's Horizon Europe research and innovation program under Grant Agreement No. 101114305 ("MILLENION-SGA1"

EU Project), the ERC Starting Grant QNets through Grant No. 804247, and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy "Cluster of Excellence Matter and Light for Quantum Computing (ML4Q) EXC 2004/1," No. 390534769.

DATA AVAILABILITY

The data that support the findings of this article are openly available [107].

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 Ref. [78] since they are not useful for the task at hand.

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

The tableau for $|0\rangle_L$ of the $[[7,1,3]]$ Steane code [73] is a matrix of binary numbers of size 7×15 that contains $7-1=6$ stabilizer generators and one logical operator Z_L. Table III shows the generators of the $[[7,1,3]]$ Steane code. Equation (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.

$$\left(\begin{array}{ccccccc|ccccccc|c} 0&0&0&0&0&0&0&1&1&1&1&1&1&1&0\\ 0&0&0&0&0&0&0&1&0&1&0&1&0&1&0\\ 1&0&1&0&1&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&1&0&0&1&1&0\\ 0&1&1&0&0&1&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&0\\ 0&0&0&1&1&1&1&0&0&0&0&0&0&0&0 \end{array}\right). \quad \text{(A1)}$$

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 [78], 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 obtain the canonical tableau in Eq. (A2). The Pauli string for this tableau is as follows: $+XIXIXIX$, $+ZIIIIZZ$, $+IXXIIXX$, $+IZIIZIZ$, $+IIZIZZI$, $+IIIXXXX$, and $+IIIZZZZ$. We can reorder the rows or change a row by multiplying other rows in Eq. (A1) and still find the same canonical tableau.

$$\left(\begin{array}{ccccccc|ccccccc|c} 1&0&1&0&1&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0&1&1&0\\ 0&1&1&0&0&1&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&1&0&1&0\\ 0&0&0&0&0&0&0&0&0&1&0&1&1&0&0\\ 0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&0 \end{array}\right). \quad \text{(A2)}$$

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 [98]. 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 [98]. We use the Adam optimizer with a learning rate of 0.001 with an annealing learning rate. We train ten 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 four 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 found using NVIDIA Quadro RTX 6000 GPU. The training is performed with the same seed value to ensure the same randomness.

For all experiments, we set the stopping threshold at $\epsilon = 0.9999$ and the maximum steps in the trajectory as $L = 50$ and, in harder cases, $L = 100$. 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 Appendixes K and 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 [139] 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. 14, we see that the 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, we do not take into account when the Pauli identity I matches in both target and current tableau. Thus, 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 case in the inset of Fig. 14, 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.

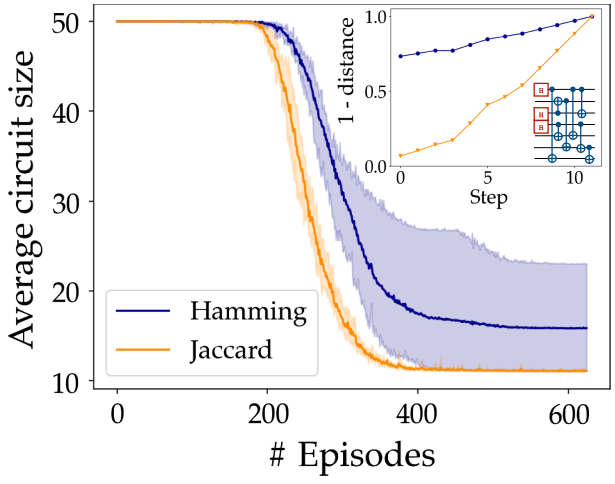


FIG. 14. Comparison of Hamming and Jaccard distance for the reward function. We show the average circuit size of training an RL agent to prepare the $|0\rangle_L$ state of the $[[7, 1, 3]]$ Steane code using the Hamming and Jaccard distance for the complementary tableau distance reward $1 - d_t$. The color shade shows the standard deviation over five different trainings. The inset shows how the inverse distance value evolves for each step of the gate application for the circuit shown.

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

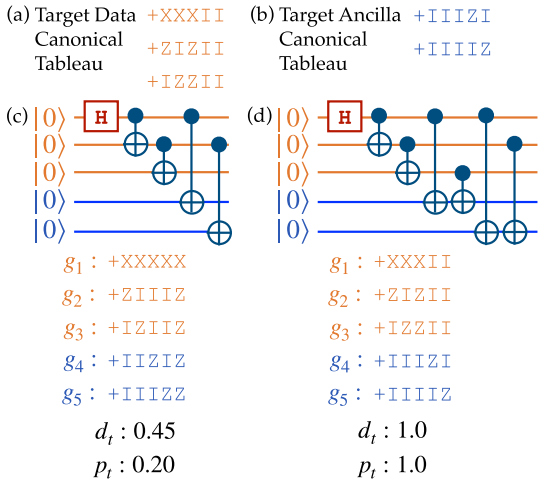


FIG. 15. Illustration of the complementary tableau distance (d_t) and product state (p_t) calculation. We want to prepare the state $|000\rangle + |111\rangle$ with two ancillas ($n_A = 2$). The target canonical tableau can be separated according to the data (a) and ancilla (b) qubits. (c) Example of a circuit that almost prepares the target state, but the data qubits are entangled with the ancilla. (d) Example of a circuit that prepares the target state and is a product state of the data qubits and the ancillas.

canonical tableau with $I^{\otimes n_A}$, so the target canonical tableau for the data qubits is now $+XXXII$, $+ZIZII$, and $+IZZII$, as shown in Fig. 15(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. 15(b).

Figures 15(c) and 15(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. 15(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. However, the circuit in Fig. 15(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

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.

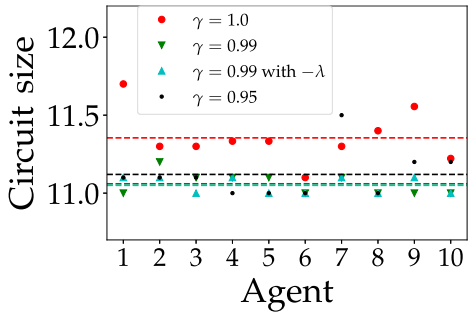


FIG. 16. Illustration of how the discount factor γ affects the circuit size. We train ten different agents for preparing the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code with $\gamma = \{0.95, 0.99, 1\}$. Additionally, we also try to set γ to 0.99 and add the $\lambda = -1/50$ (referred to in the figure as $\gamma = 0.99$ with $-\lambda$) in the reward function to penalize longer trajectories. The dashed line shows the average circuit size.

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. 16. 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 III shows the stabilizer generators for the codes used in this paper.

TABLE III. Stabilizer generators for the $[[5,1,3]]$ perfect code [110], the $[[7,1,3]]$ Steane code [73], the $[[9,1,3]]$ Shor code [111], the $[[9,1,3]]$ Surface-17 code [17], the $[[15,1,3]]$ quantum Reed-Muller code or 3D color code [116], the $[[17,1,5]]$ 2D color code [76], the $[[19,1,5]]$ 2D color code [76], and the $[[25,1,5]]$ surface code [17]. 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
		IIIIIIZZI	IIIIXXIXX
		IIIIIIZIZ	IIIIIIXXI

$[[15,1,3]]$ Reed-Muller	$[[17,1,5]]$ 2D Color
ZIZIZIZIZIZIZIZ	XXXXIIIIIIIIIIIII
XIXIXIXIXIXIXIX	ZZZZIIIIIIIIIIIII
IZZIIZZIIZZIIZZ	XIXIXXIIIIIIIIIII
IXXIIXXIIXXIIXX	ZIZIZZIIIIIIIIIII
IIIZZZZIIIIZZZZ	IIIIXXIIXXIIIIIII
IIIXXXXIIIIXXXX	IIIIZZIIZZIIIIIII
IIIIIIIZZZZZZZZ	IIIIIIXXIIXXIIIII
IIIIIIIXXXXXXXX	IIIIIIZZIIZZIIIII
IIZIIIZIIIZIIIZ	IIIIIIIIXXIIXXIII
IIIIZIZIIIIIZIZ	IIIIIIIIZZIIZZIII
IIIIIZZIIIIIIZZ	IIIIIIIIIIXXIIXXI
IIIIIIIIIZZIIZZ	IIIIIIIIIIZZIIZZI

(Table continued)

TABLE III. *(Continued)*

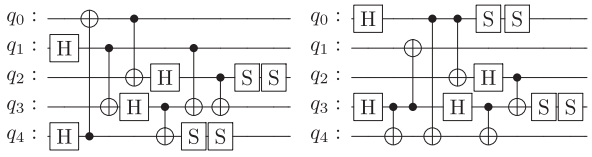
[[15, 1, 3]] Reed-Muller	[[17, 1, 5]] 2D Color
IIIIIIIIIIIZZZZ	IIIIIIIXIIIXIIIXX
IIIIIIIIZIZIZIZ	IIIIIIIZIIIZIIIZZ
	IIXXIXXIIXXIIXXII
	IIZZIZZIIZZIIZZII

[[19, 1, 5]] Color	[[25, 1, 5]] Surface
IIXIIXIIXXIIIIIIIII	XXIIIIIIIIIIIIIIIIIIIIIII
IIZIIZIIZZIIIIIIIII	IIXXIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIXIXIIIIXX	ZZIIIZZIIIIIIIIIIIIIIIII
IIIIIIIIIZIZIIIIZZ	IXXIIIXXIIIIIIIIIIIIIIII
IIIIIIIIIXXIXIIIIX	IIZZIIIZZIIIIIIIIIIIIIII
IIIIIIIIIZZIZIIIIZ	IIIXXIIIXXIIIIIIIIIIIIII
XXIIIIIIIIIIIXXIII	IIIIZIIIIZIIIIIIIIIIIIII
ZZIIIIIIIIIIIZZIII	IIIIIZIIIIZIIIIIIIIIIIII
XXIXXIXIIIIIIIIIXII	IIIIIXXIIIXXIIIIIIIIIIII
ZZIZZIZIIIIIIIIIZII	IIIIIIZZIIIZZIIIIIIIIIII
IIXIXIIIXIIIIIIIXII	IIIIIIIXXIIIXXIIIIIIIIII
IIZIZIIIZIIIIIIIZII	IIIIIIIIZZIIIZZIIIIIIIII
IXIIIIXXIIIIIIIXIII	IIIIIIIIIZZIIIZZIIIIIIII
IZIIIIZZIIIIIIIZIII	IIIIIIIIIIXXIIIXXIIIIIII
IIIXIIXXIIXXXIIIIII	IIIIIIIIIIIZZIIIZZIIIIII
IIIZIIZZIIZZZIIIIII	IIIIIIIIIIIIXXIIIXXIIIII
IIIXXIIIXXIXIXIIIII	IIIIIIIIIIIIIZIIIIZIIIII
IIIZZIIIZZIZIZIIIII	IIIIIIIIIIIIIIZIIIIZIIII
	IIIIIIIIIIIIIIIXXIIIXXIII
	IIIIIIIIIIIIIIIZZIIIZZII
	IIIIIIIIIIIIIIIIXXIIIXXI
	IIIIIIIIIIIIIIIIIZZIIIZZ
	IIIIIIIIIIIIIIIIIIIXXII
	IIIIIIIIIIIIIIIIIIIIIXX

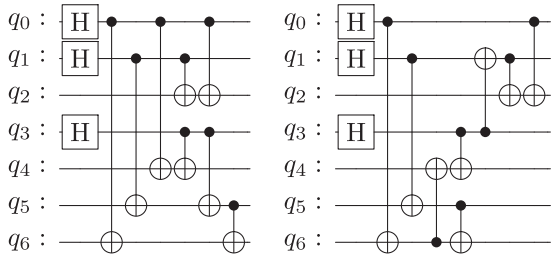
APPENDIX G: LOGICAL STATE PREPARATION CIRCUITS FOR ALL-TO-ALL QUBIT CONNECTIVITY WITH STANDARD GATE SETS

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 [107].

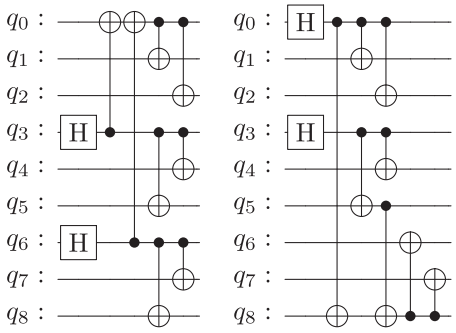
We study the $|\mathbf{0}\rangle_\mathbf{L}$ state of the $[[\mathbf{5}, \mathbf{1}, \mathbf{3}]]$ perfect code. We see that the prepared circuit has a pattern of two S gates at the end of the circuit because the RL agent would first aim to obtain the correct stabilizer generators without worrying about the sign and then apply Z gates (which can be decomposed into two S gates) to fix the sign.

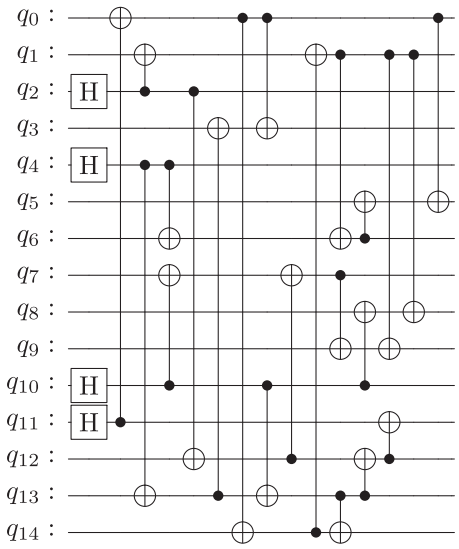


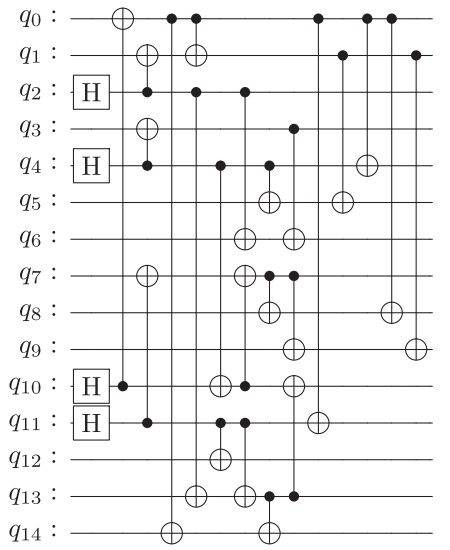
Next, we look at the $|\mathbf{0}\rangle_\mathbf{L}$ state of the $[[\mathbf{7}, \mathbf{1}, \mathbf{3}]]$ Steane code. We see that the RL agent learns from scratch to prepare part of the initial state of the physical qubits to $|+\rangle$ by applying an H gate, similar to the strategy in Ref. [85]. 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.

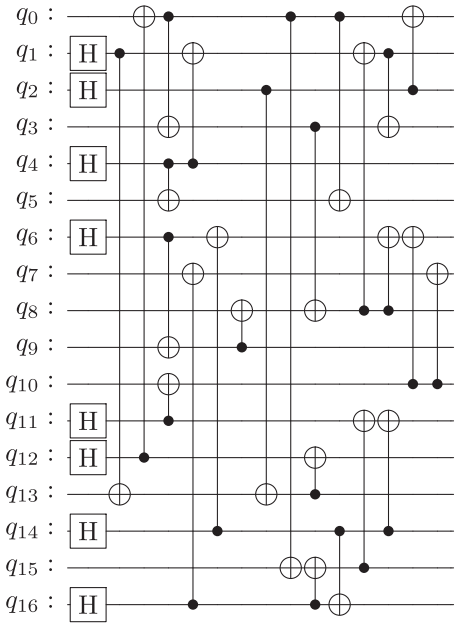


We now look at the $|+\rangle_\mathbf{L}$ state of the $[[\mathbf{9}, \mathbf{1}, \mathbf{3}]]$ Shor code, followed by the $|\mathbf{0}\rangle_\mathbf{L}$ state of the $[[\mathbf{17}, \mathbf{1}, \mathbf{5}]]$ 2D color code, and the $|\mathbf{0}\rangle_\mathbf{L}$ state of the $[[\mathbf{15}, \mathbf{1}, \mathbf{3}]]$ Reed-Muller and distance-3 3D color codes.









APPENDIX H: TRANSFER LEARNING FOR DIFFERENT LOGICAL STATES

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 case 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 [124]. 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. 17. 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 $|0\rangle_L$, then retrain it to prepare $|+\rangle_L$, and consecutively retrain it to prepare $|+i\rangle_L$.

Figures 17(a) and 17(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. Figures 17(c) and 17(d) show the same but for the preparation of $|+i\rangle_L$. Here, we qualitatively compare the performance based on the metrics used in Ref. [124] 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 the value without transfer learning. Therefore, we have shown that the proposed transfer learning technique is useful to efficiently prepare different logical states.

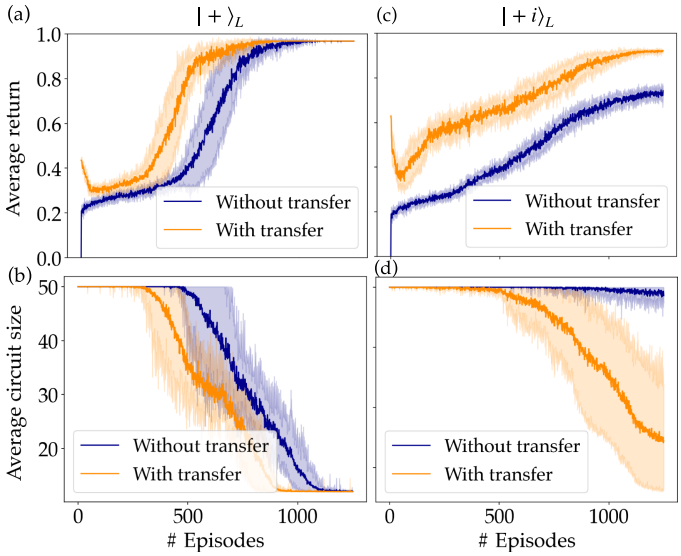
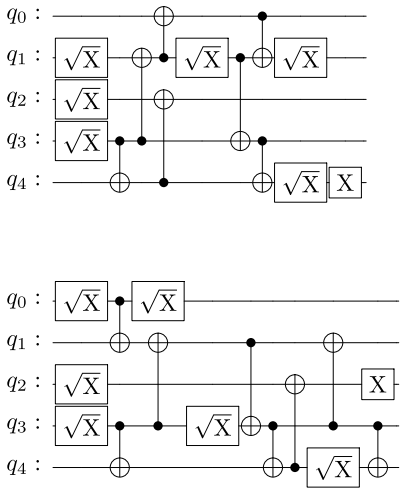


FIG. 17. Transfer learning results for different logical state preparation tasks. We first train the agent to prepare the $|0\rangle_L$ for the $[[7,1,3]]$ Steane code with $Z_L = Z^{\otimes 7}$ and then reuse and retrain the agent to prepare $|+\rangle_L$ with $X_L = X^{\otimes 7}$ and then successively $|+i\rangle_L$ with $Y_L = -Y^{\otimes 7}$. (a),(b) Average return and circuit size of training without and with transfer learning to prepare $|+\rangle_L$. (c),(d) Same as panels (a) and (b), but for the preparation of $|+i\rangle_L$. The shaded area shows the standard deviation of training ten different agents.

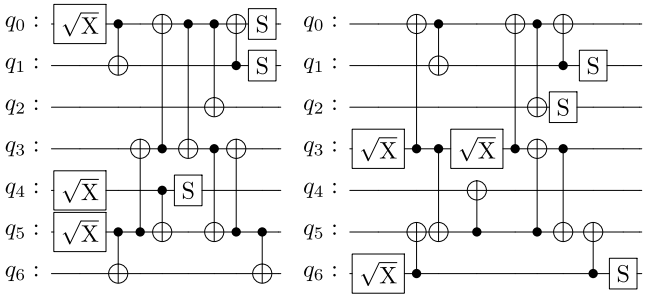
APPENDIX I: LOGICAL STATE PREPARATION CIRCUITS FOR IBM QUANTUM DEVICES

Here, we show 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 [107].

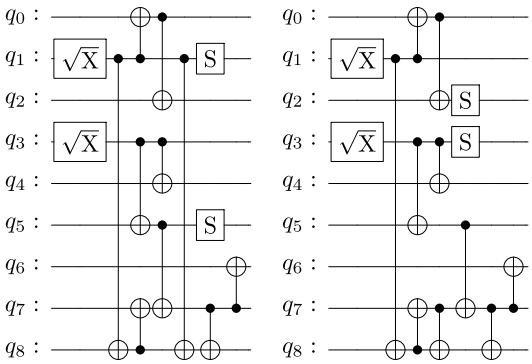
We consider the $|\mathbf{0}\rangle_\mathbf{L}$ state of the $[[\mathbf{5},\mathbf{1},\mathbf{3}]]$ perfect code on the IBMQ Manila [119] connectivity. The qubit placement is 4,3,0,2,1.



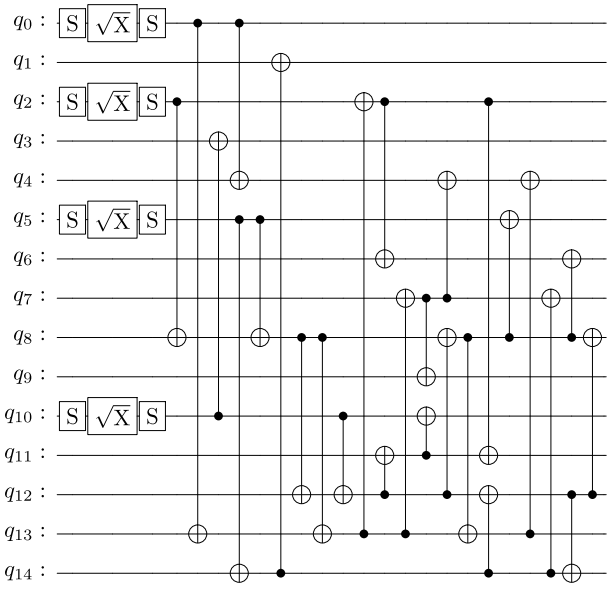
Next, we look at the $|\mathbf{0}\rangle_\mathbf{L}$ state of the $[[\mathbf{7},\mathbf{1},\mathbf{3}]]$ Steane code on the IBMQ Jakarta [120] connectivity. The qubit placement is 5,6,4,3,0,1,2.



Now, we consider the $|+\rangle_\mathbf{L}$ state of the $[[\mathbf{9},\mathbf{1},\mathbf{3}]]$ Shor code on the IBMQ Guadalupe [121] connectivity. The qubit placement is 5,3,8,7,6,4,0,1,2.



Finally, we look at the $|\mathbf{0}\rangle_{\mathbf{L}}$ state of the $[[\mathbf{15},\mathbf{1},\mathbf{3}]]$ Reed-Muller code on the IBMQ Tokyo [122] connectivity. The qubit placement is 5,17,2,9,10,13,1,11,7,16,4,3,8,6,12.



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 approach 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 approach 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 qubits 1 and 3 and the connections between qubits 2 and 4 are removed is shown in Fig. 18(a). Assuming that we are using CNOT gates, we need to remove the output nodes of the actor network corresponding to the following actions: $\text{CNOT}(1,3)$, $\text{CNOT}(3,1)$, $\text{CNOT}(2,4)$, and $\text{CNOT}(4,2)$, and keep the others. We then fine-tune this network to prepare the state for the new connectivity.

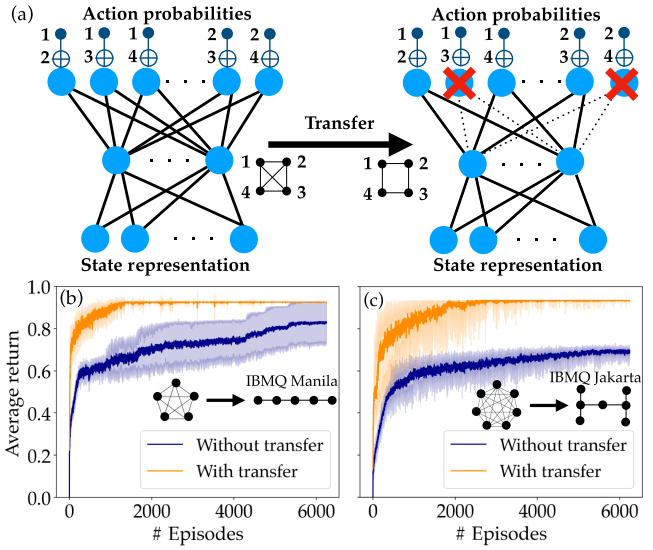


FIG. 18. Results of transfer learning for logical state preparation on IBM Quantum devices. We first train the agent to prepare $|0\rangle_L$ in a setting of all-to-all qubit connectivity and then reuse and retrain it to prepare the same $|0\rangle_L$ under a different, restricted qubit connectivity. (a) 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) 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 [140] connectivity and without transfer. (c) Training evolution of preparing $|0\rangle_L$ of the $[[7,1,3]]$ Steane code with $Z_L = Z^{\otimes 7}$ on all-to-all qubit connectivity and transferring it to IBMQ Jakarta [120] connectivity and without transfer.

We compare the preparation of $|0\rangle_L$ of the $[[5,1,3]]$ code on IBMQ Lima [140] connectivity and the $[[7,1,3]]$ code on IBMQ Jakarta [120] 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. Figures 18(b) and 18(c) show 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.

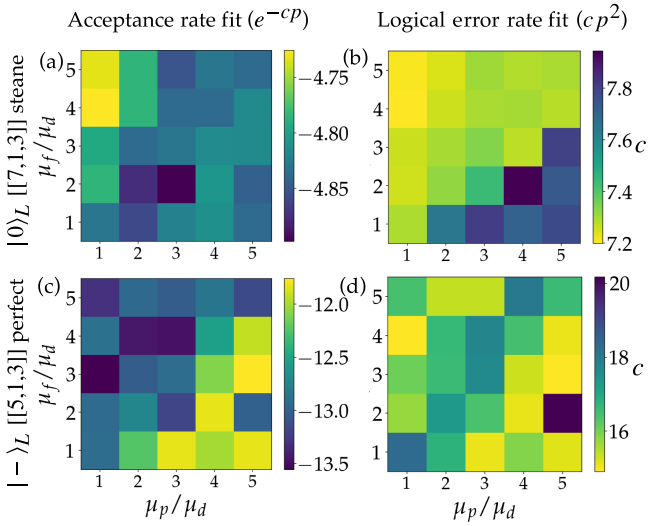


FIG. 19. Varying the weights of the reward function for verification circuit synthesis. We vary the μ_f/μ_d and μ_p/μ_d ranging from 1 to 5 with an interval of 1. The heatmap shows the average fitting coefficients for the acceptance [panels (a) and (c), the higher the better] and the logical error rate [panels (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 Ref. [32] [panels (a) and (b)] and $|-\rangle_L$ of the $[[5,1,3]]$ perfect code taken from Ref. [29] [panels (c) and (d)].

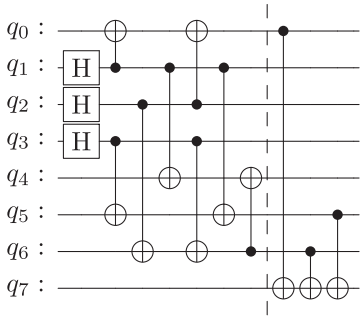
Effectively, only the weight ratios matter since scaling the reward function generally does not affect the performance of the reinforcement learning training. We vary μ_f/μ_d and μ_p/μ_d and synthesize the verification circuit at each point. We then compute the acceptance and logical error rates and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

In Figs. 19(a) and 19(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 Figs. 19(c) and 19(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 [107].

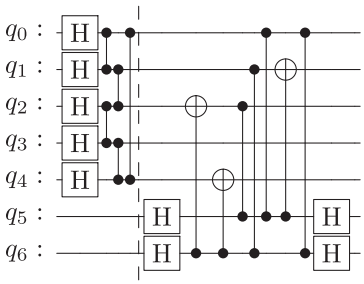
We first show the synthesis of the verification circuit for the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code by taking the non-FT logical state preparation circuit from Ref. [32]. We show that the RL method also rediscovers the same circuit used in Ref. [32] shown below, which measures the stabilizer Z logical operator $IIIIIZZ$:



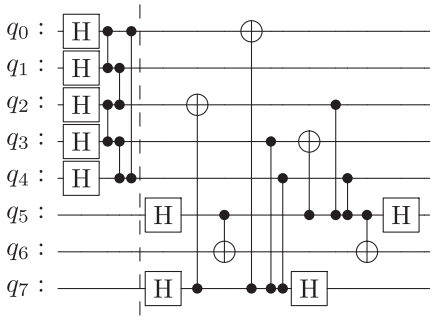
The circuit with the lowest fitting coefficients of the acceptance and logical error rate is the circuit shown in Fig. 6(a), which measures the stabilizer Z logical operator $IIZIZZI$. The other circuits are just permutations of the CNOT gates in the verification circuit.

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.

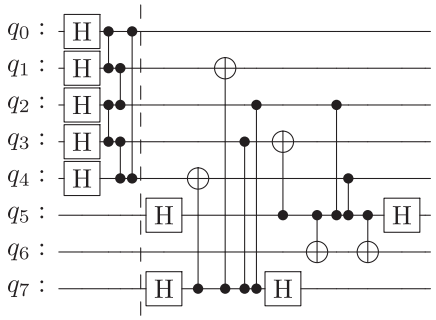


The circuit with the lowest fitting coefficients of the logical error rate is the circuit shown below. The circuit needs three flag qubits. The acceptance and logical error rate fitting coefficients are −14.1 and 12.1, respectively. One of the interesting properties that the RL agent learned is that 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

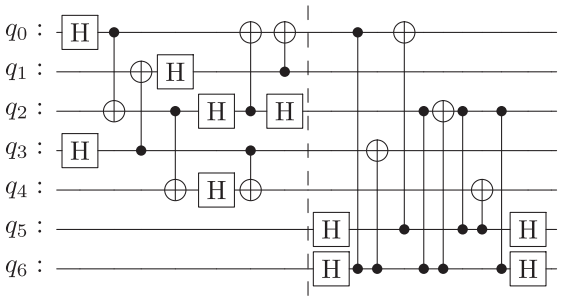
measure the stabilizer operators $IIZXZ$ in the first flag qubit and $IXZZX$ in the third flag qubit.



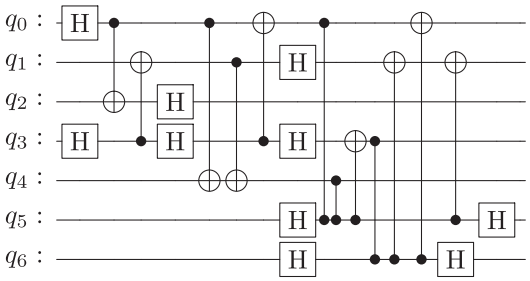
APPENDIX M: EXAMPLES OF RL-DISCOVERED FAULT-TOLERANT LOGICAL STATE PREPARATION CIRCUITS SHOWN IN TABLE I

Here, we show circuit examples from Table I with two RL approaches: LSP + VCS and integrated fault-tolerant IFT-LSP. These circuits are also available online [107].

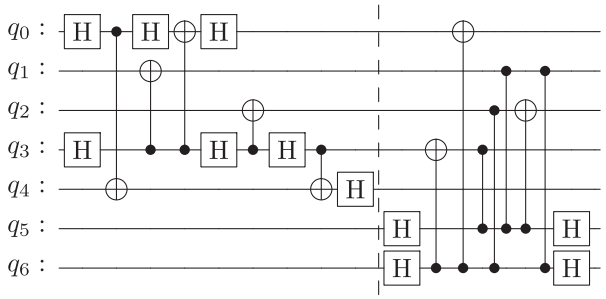
We consider the $|\mathbf{1}\rangle_{\mathbf{L}}$ state of the $[[\mathbf{5},\mathbf{1},\mathbf{3}]]$ perfect code. The circuit obtained with the LSP + VCS approach has 14 two-qubit gates and two flag qubits:



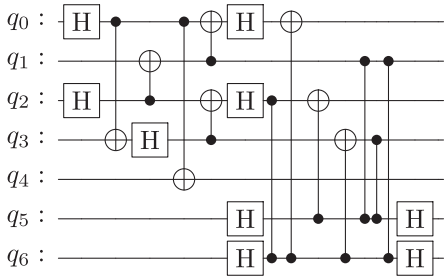
The circuit obtained using the IFT-LSP approach has a smaller number of only 12 two-qubit gates and two flag qubits:



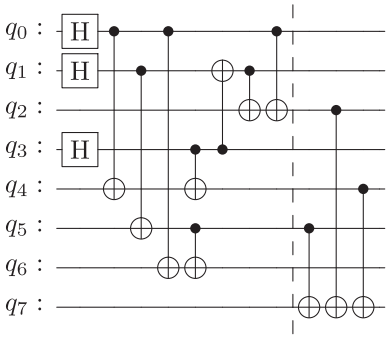
Next, we look at the $|-\rangle_{\mathbf{L}}$ state of the $[[\mathbf{5},\mathbf{1},\mathbf{3}]]$ perfect code. The circuit with the LSP + VCS approach has 12 two-qubit gates and two flag qubits:



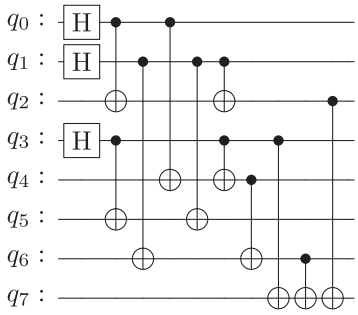
The circuit with the IFT-LSP approach has 12 two-qubit gates and two flag qubits:



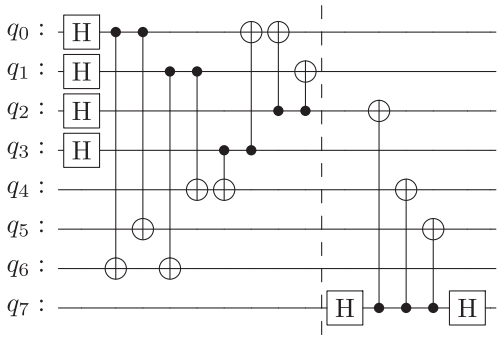
Now, we consider the $|\mathbf{0}\rangle_{\mathbf{L}}$ state of the $[[\mathbf{7},\mathbf{1},\mathbf{3}]]$ Steane code. The circuit found with the LSP + VCS approach has 11 two-qubit gates and one flag qubit:



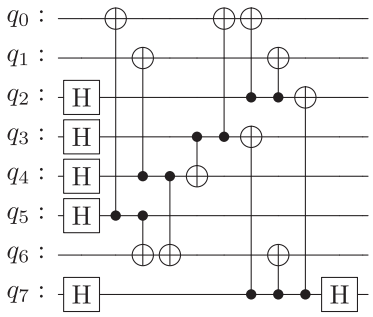
The circuit learned with the IFT-LSP approach has 11 two-qubit gates and one flag qubit, while using a smaller number of single-qubit gates:



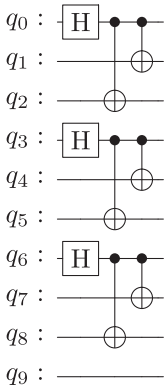
Next, we consider the $|+\rangle_{\mathbf{L}}$ state of the $[[\mathbf{7},\mathbf{1},\mathbf{3}]]$ Steane code. The circuit learned with the LSP + VCS approach has 11 two-qubit gates and one flag qubit:



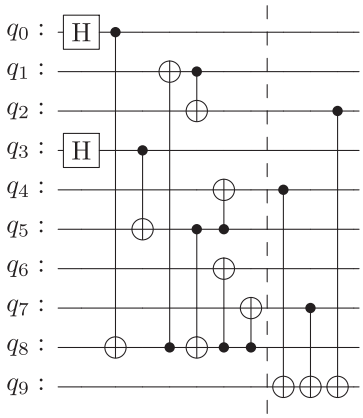
The circuit found with the IFT-LSP approach has 11 two-qubit gates and one flag qubit:



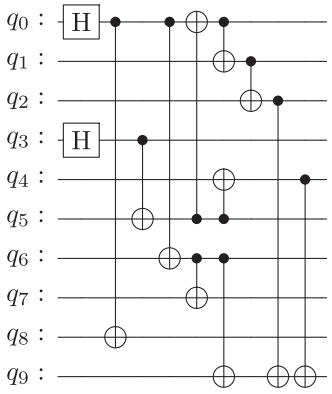
For the $|\mathbf{0}\rangle_{\mathbf{L}}$ state of the $[[\mathbf{9},\mathbf{1},\mathbf{3}]]$ Shor code, both approaches discover the known fault-tolerant state preparation circuit that does not require any flag qubits at all, shown below.



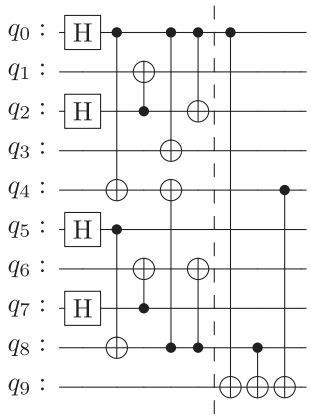
For the $|+\rangle_{\mathbf{L}}$ state of the $[[\mathbf{9},\mathbf{1},\mathbf{3}]]$ Shor code, the circuit learned with the LSP + VCS approach has 11 two-qubit gates and one flag qubit:



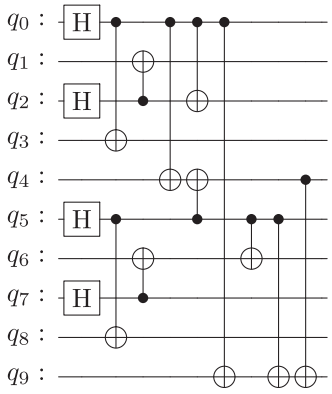
The circuit learned using the IFT-LSP approach has, again, 11 two-qubit gates and one flag qubit:



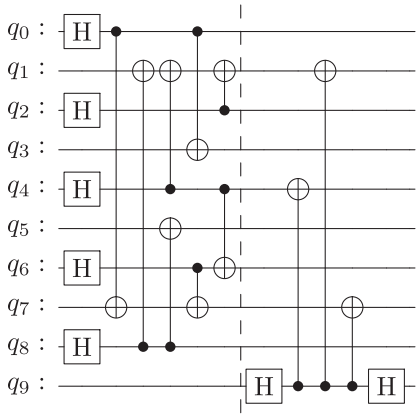
For the $|\mathbf{0}\rangle_{\mathbf{L}}$ state of the $[[\mathbf{9},\mathbf{1},\mathbf{3}]]$ Surface-17 code, the circuit learned with the LSP + VCS approach has 11 two-qubit gates and one flag qubit:



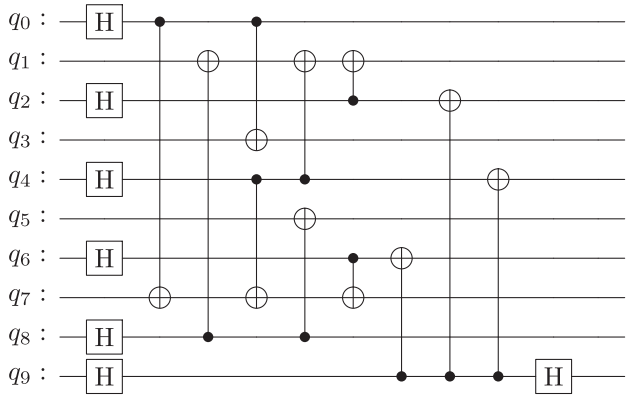
The circuit learned using the IFT-LSP approach has, again, 11 two-qubit gates and one flag qubit:



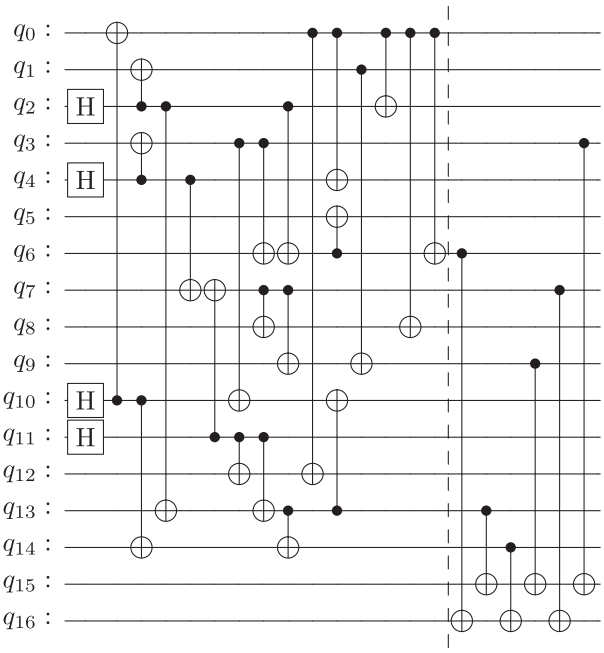
For the $|+\rangle_{\mathbf{L}}$ state of the $[[\mathbf{9},\mathbf{1},\mathbf{3}]]$ Surface-17 code, the circuit learned with the LSP + VCS approach has 11 two-qubit gates and one flag qubit:



The circuit learned using the IFT-LSP approach has 11 two-qubit gates and one flag qubit:

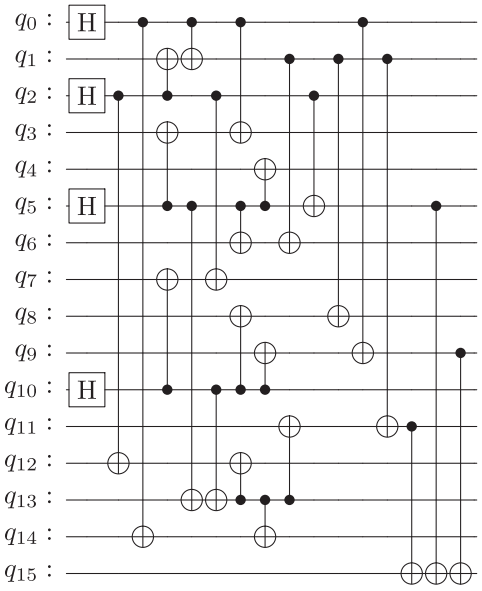


For the $|\mathbf{0}\rangle_{\mathbf{L}}$ state of the $[[\mathbf{15},\mathbf{1},\mathbf{3}]]$ Reed-Muller code, the circuit found based on the LSP + VCS approach has 29 two-qubit gates and two flag qubits:

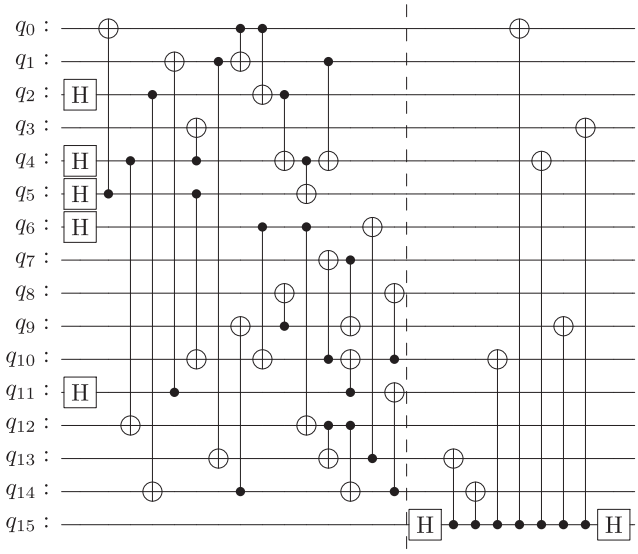


In contrast, the circuit obtained with the IFT-LSP approach has 25 two-qubit gates (instead of 29) and only requires one flag qubit (instead of two), illustrating the superior performance of the IFT-LSP

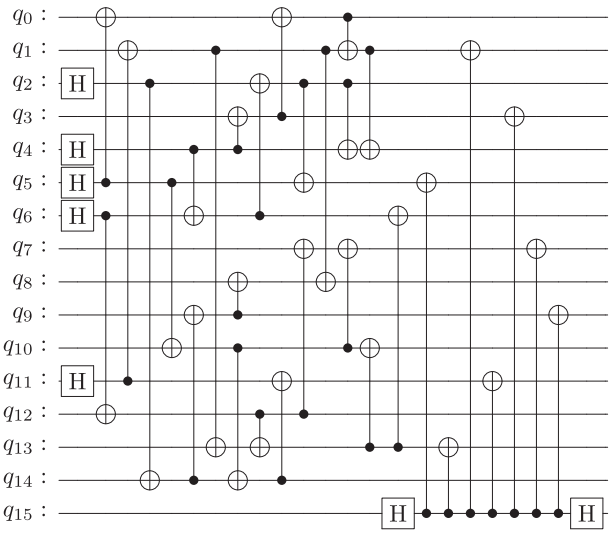
over the LSP + VCS approach:



For 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 one flag qubit:



The circuit found with the IFT-LSP approach requires also 31 two-qubit gates and one flag qubit:



APPENDIX N: EXAMPLES OF WHEN LSP + VCS FAILS WITH RESTRICTED CONNECTIVITY

Here, we discuss two cases where LSP followed by VCS to prepare a fault-tolerant logical state would fail in restricted connectivity settings. Figure 20(a) shows a case

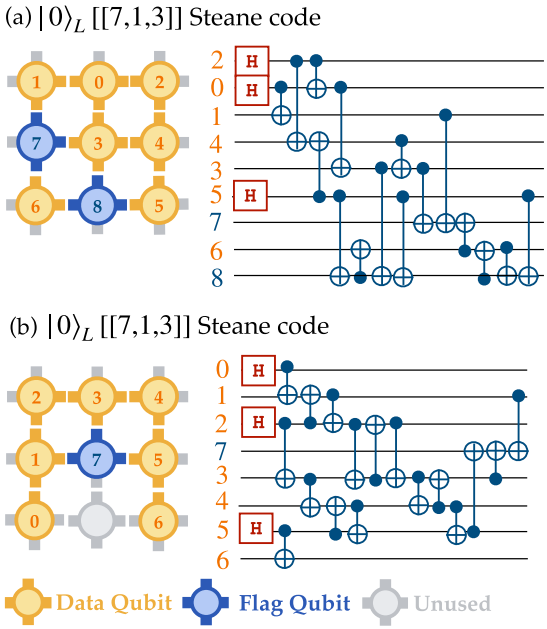


FIG. 20. Two cases where the LSP + VCS approach fails but the IFT-LSP approach succeeds with restricted connectivity. We use the IFT-LSP approach to fault tolerantly prepare the $|0\rangle_L$ state of the $[[7,1,3]]$ Steane code on a 2D grid. (a) Case where the data qubit is only connected to the flag qubits. (b) Case where VCS does not find a verification circuit.

where a data qubit [qubit 6 in Fig. 20(a)] is only connected to the flag qubits, and the circuit on the right is the output of the IFT-LSP task. If we separate the task, the logical state preparation task fails to prepare the state in this case. Figure 20(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 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

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 Figs. 21(a) and 21(b), we quantify and compare the 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 five two-qubit gates more than the circuit shown with full connectivity. Nevertheless, the logical error rate is larger by only 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 four flag qubits, while we only need one flag qubit in the all-to-all qubit connectivity case. We have also observed a similar phenomenon in Appendix L, where using more flag qubits decreases the acceptance rate but increases 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 it is 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.

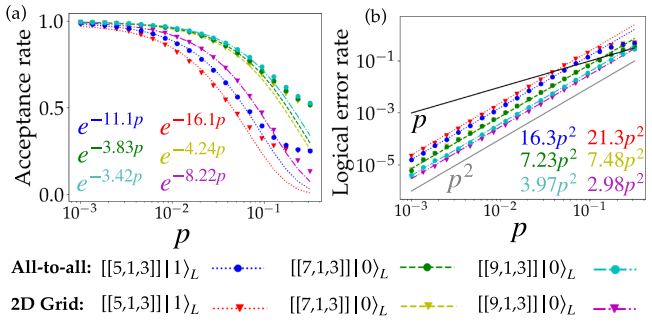


FIG. 21. Logical error rate and acceptance rate of circuits found with integrated fault-tolerant logical state preparation. We take the circuits shown in Fig. 8. The circuits for the $[[9,1,3]]$ Surface-17 in all-to-all qubit connectivity are shown in Appendix M.

APPENDIX P: CIRCUITS FOR FAULT-TOLERANT LOGICAL STATE PREPARATION IN RESTRICTED CONNECTIVITY

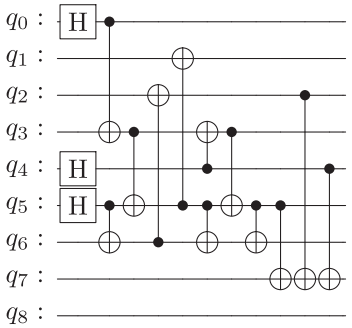
Here, we show examples of fault-tolerant logical state preparation circuits with restricted connectivity as shown in Fig. 10. If the qubit placement is given as $a, b, c, \ldots$, then the qubit 0 (q_0) in the circuit is placed in qubit a on the device, qubit 1 (q_1) is placed in qubit b on the device, and so on. These circuits are also available online [107].

1. The $|0\rangle_L$ state of the [[7,1,3]] Steane code on 2D grid connectivity (Google Sycamore)

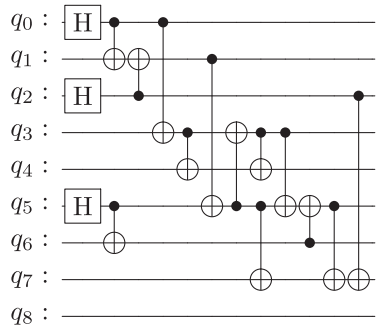
For the qubit placement, we follow the row-major order starting with 0 at the top left of the qubit and 9 at the bottom right of the qubit. The last two qubits are always given as ancilla qubits. The RL agent can decide whether to use them or not.

The circuit for the first qubit placement shown in Fig. 10(a) with 11 two-qubit gates is already shown in Fig. 8(c).

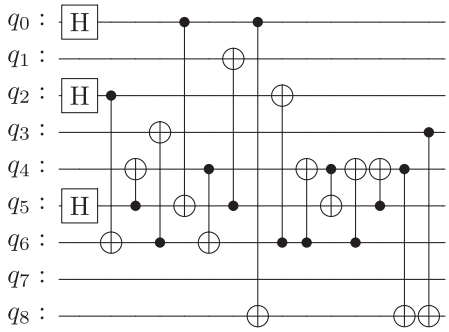
For the circuit for the second qubit placement shown in Fig. 10(a) with 12 two-qubit gates, the qubit placement is 2,5,6,1,0,4,7,3,8.



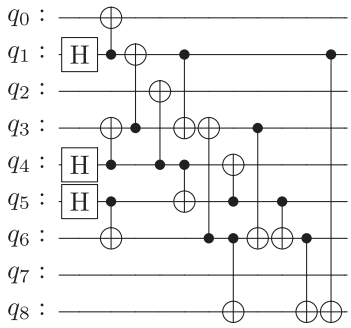
For the circuit for the third qubit placement shown in Fig. 10(a) with 13 two-qubit gates, the qubit placement is 2,5,8,1,0,4,3,7,6.



For the circuit for the fourth qubit placement shown in Fig. 10(a) with 14 two-qubit gates, the qubit placement is 0,2,8,6,4,1,7,5,3.

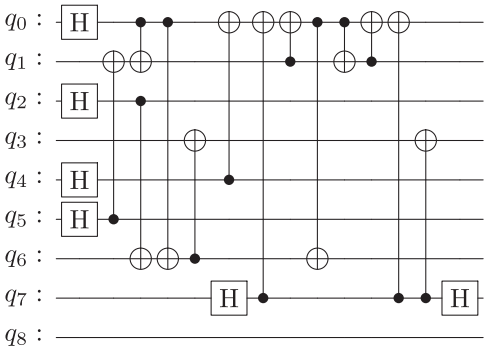


For the circuit for the fifth qubit placement shown in Fig. 10(a) with 14 two-qubit gates, the qubit placement is 2,5,0,4,3,6,7,1,8.

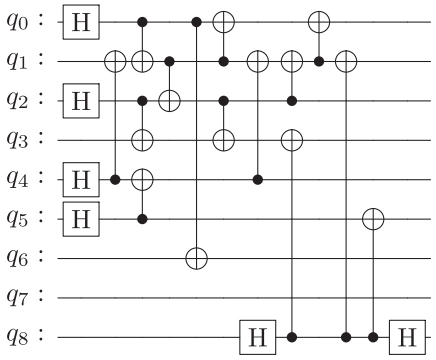


2. The $|+\rangle_L$ state of the [[7,1,3]] Steane code on 2D grid connectivity (Google Sycamore)

The qubit placement for the circuit below is 4,3,8,2,7,0,5,1,6.

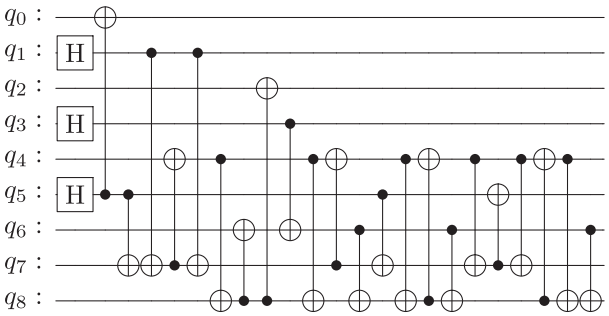


The qubit placement for the circuit below is 5,4,7,6,1,0,8,2,3.



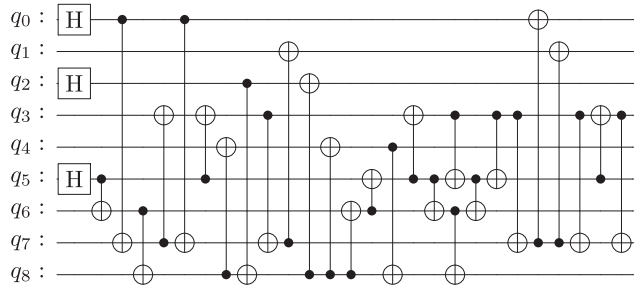
3. The $|0\rangle_L$ state of the [[7,1,3]] Steane code on heavy-hex connectivity (IBMQ Guadalupe)

The circuit for the first qubit placements shown in Fig. 10(b) has 22 two-qubit gates. The qubit placement in the IBMQ Guadalupe [121] device is 3,0,6,12,4,2,10,1,7. The last two qubits are given as ancilla qubits.

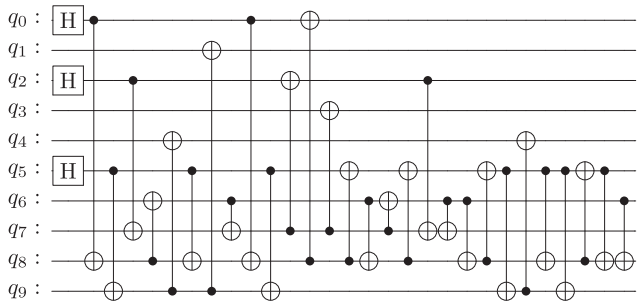


Now, we consider the circuit for the second qubit placement shown in Fig. 10(b) with 27 two-qubit gates. The qubit placement in the IBMQ Guadalupe [121] device

is 9,5,15,11,10,14,13,8,12. The last two qubits are given as ancilla qubits.



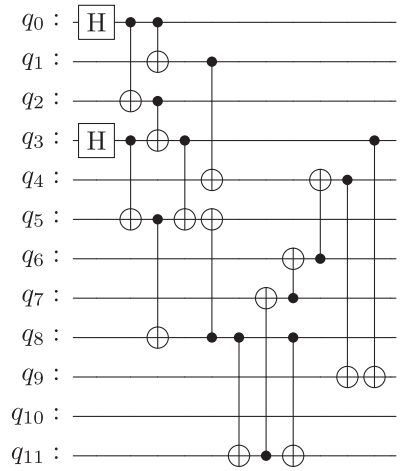
Next, we look at the circuit for the third qubit placement shown in Fig. 10(b) with 28 two-qubit gates. The qubit placement in the IBMQ Guadalupe [121] device is 6,15,0,2,13, 10,4,1,7,12. The last three qubits are given as ancilla qubits.



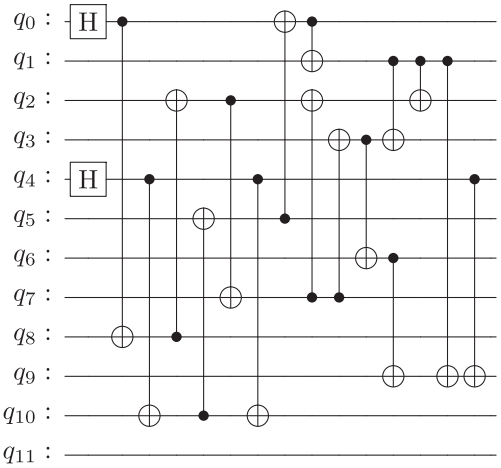
4. The $|+\rangle_L$ state of the [[9,1,3]] Shor code on 2D grid connectivity (Google Sycamore)

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.

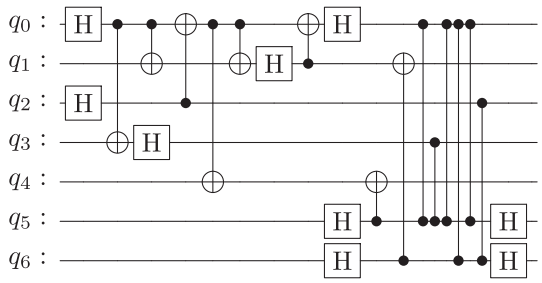


The qubit placement for the circuit below is 6,7,10,8,1,3,5,11,9,4,0,2. One flag qubit is not used. In this case, one data qubit has the flag qubit as its neighbor, so the agent needs to use the flag qubit as a bridge to that data qubit.

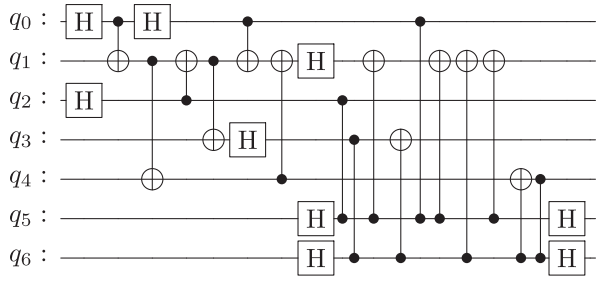


5. The $|-\rangle_L$ state of the [[5,1,3]] perfect code on IBMQ Tokyo

The qubit placement for the circuit below on the IBMQ Tokyo [122] connectivity is 6,1,7,5,11,2,10.



The qubit placement for the circuit below on the IBMQ Tokyo [122] connectivity is 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 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 ten different circuits.

In Fig. 22, 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 finding is expected since the RL training starts from scratch, so μ_d must be prioritized.

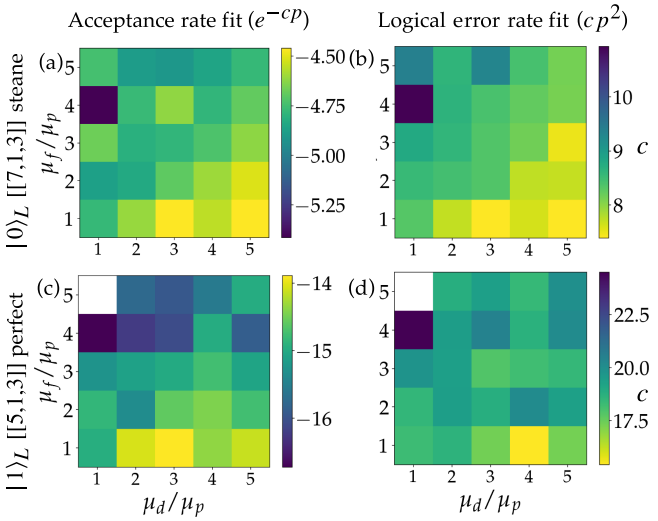


FIG. 22. Varying the weights of the reward function for IFT-LSP. We vary μ_f/μ_p and μ_d/μ_p ranging from 1 to 5 with an interval of 1. The heatmap shows the average fitting coefficients for the acceptance rate [in panels (a) and (c), the higher the better] and the logical error rate [in panels (b) and (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 panels (a) and (b)] and the $|1\rangle_L$ state of the $[[5,1,3]]$ perfect code [in panels (c) and (d)]. The white color means that no agent has converged on that parameter.

APPENDIX R: DISCUSSIONS ON SCALABILITY

Here, we discuss several aspects of scalability and various ideas for scaling our approach to larger system sizes or code distances. In addition, we would like to emphasize that the results presented here are not only proofs of principle, but they can be directly implemented in experiments.

1. Scalability of the simulator

We first discuss the scalability of our parallel simulator shown in Fig. 23. As we can see, the simulation time scales polynomially with the number of qubits due to the efficient simulation using the stabilizer formalism. The simulation of trajectories with error propagation is only about 3 times slower when we increase the distance from 3 to 5. Furthermore, our efficient simulation leverages parallelization on GPUs, which significantly reduces the time required to train the RL agent.

As discussed in Sec. V, in general, one needs to compute the minimum weight of each error by multiplying it by all members of the stabilizer group and logical operator when computing the flag reward f_t. Generating all members of the stabilizer group scales exponentially with the number of qubits. This strategy ensures that the minimum weight error is always found. In some cases, however, more sophisticated strategies that avoid brute-force checking of the entire stabilizer set are also effective. For instance, for the small CSS codes we study, multiplying each error only with the stabilizer generators and the logical operators is sufficient to find the minimum weight error. Therefore, it scales only linearly with the number of qubits. The time shown in Fig. 23 for the simulation with error propagation multiplies each error only with the stabilizer generators and the logical operators. It is

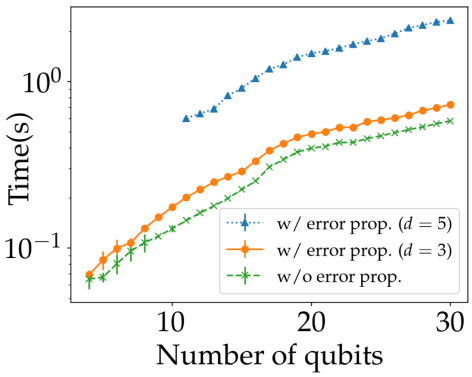


FIG. 23. Simulation time of ten trajectories with 50 steps in parallel with error propagation for $d = 3$ and 5 codes for the VCS and IFT-LSP tasks or without error propagation for the LSP task. In the LSP task, d has no effect on simulation performance.

important to note that not finding the absolute minimum weight error does not imply that the circuit cannot be FT, but it leads to a misjudgment of the tolerable errors, making the task of finding a FT encoding circuit even more difficult.

2. Encoding concatenation

Our existing RL results, in some cases, can be scaled directly to higher numbers of qubits and higher distance codes with encoding concatenation. Code concatenation has been studied extensively before [141,142]. To make a concatenated encoding, we take an encoding circuit from an inner code C_1 to encode some logical state. The encoded logical state now acts as physical qubits, and we encode it again using an encoding circuit from an outer code C_2. It has been proven that concatenating two codes $C_1 = [[n_1, k, d_1]]$ and $C_2 = [[n_2, k, d_2]]$ yields a $C = [[n_1 n_2, k, d \geq d_1 d_2]]$ code [143]. Note that scaling to higher distance codes with this technique would not work without FT circuits. Any gain in simpler FT circuits that the RL discovered translates directly into an amplified gain in concatenation. This process is also consistent with an approach often used in QEC, where one is dealing with subroutines (logical gadgets or building blocks) for which it is worth optimizing a subpart and thereby optimizing the whole.

We illustrate this approach by taking C_1 and C_2 as the $[[7, 1, 3]]$ Steane code to fault tolerantly prepare the $|0\rangle_L$ of the $C = [[49, 1, 9]]$ code in Fig. 24. We take the circuit for the FT preparation of $|0\rangle_L$ from Fig. 8(a) and copy it 8 times as our new physical qubits. Since the Hadamard and CNOT gates are transversal in the $[[7, 1, 3]]$ Steane code, we can then apply the same gate sequence to prepare the $|0\rangle_L$ of the $[[49, 1, 9]]$ code and ensure fault tolerance.

As we can see, the trick here is that the code should have transversal gates to ensure fault tolerance. However, other techniques such as piecewise FT [27] can also be used. More interestingly, adapting our RL approach to discover fault-tolerant logical gates is also an interesting avenue for future work.

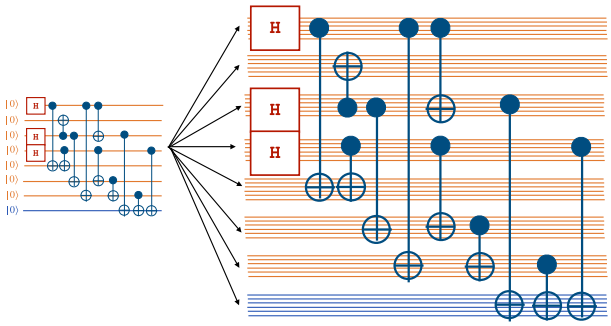


FIG. 24. Encoding concatenation to scale to higher distance codes. The RL-discovered circuits can be concatenated to achieve higher distance codes in some cases where transversal gates are possible. In this case, we concatenate the FT $|0\rangle_L$ state preparation of the $[[7, 1, 3]]$ Steane code to prepare the FT $|0\rangle_L$ of the $[[49, 1, 9]]$ code.

3. Memory scaling

As discussed in Sec. II C, in a code of distance d, all error events with probability p^i, $i < (d+1)/2$, should be tolerable. Thus, we must store and propagate all error events with probability p^i. We now discuss how the memory scales with d to store the error operators $\mathcal{E}$.

For general stabilizer codes, given a circuit with G_1 one-qubit gates and G_2 two-qubit gates, the number of errors to store and propagate is given as

$$|\mathcal{E}| = \sum_{i=1}^{(d-1)/2} \sum_{j=0}^{i} 3^{i-j} \binom{G_1}{i-j} 15^j \binom{G_2}{j}, \tag{R1}$$

where the 3 and 15 come from the number of all one-qubit and two-qubit Pauli errors to propagate except the identity for the one-qubit gates and two-qubit gates, respectively. Note that i counts the number of possible errors with probability p^i, and j chooses the number of two-qubit gates. Here, we assume that $(i-j) \leq G_1$ and $i \leq G_2$, which is realistic for circuits preparing distance d codes. Note that this assumption is not important because the formula can be modified in a straightforward way otherwise.

As discussed in Sec. VII, if we restrict ourselves to CSS codes and use only gates that do not change the error type, then we can consider only one Pauli error for one-qubit gates (i.e., X for preparing $|0\rangle_L$ or Z for preparing $|+\rangle_L$) and three Pauli errors for two-qubit gates (i.e., XI, IX, XX for preparing $|0\rangle_L$ or ZI, IZ, ZZ for preparing $|+\rangle_L$). We can then change the corresponding coefficient in the calculation of $|\mathcal{E}|$ into

$$|\mathcal{E}|_{\mathrm{CSS}} = \sum_{i=1}^{(d-1)/2} \sum_{j=0}^{i} \binom{G_1}{i-j} 3^j \binom{G_2}{j}. \tag{R2}$$

An error is a Pauli string that can be represented by binary arrays of size $2n$. Assuming that a binary digit is stored in 1 byte, Fig. 25 shows how the memory scales with n and d. We can see that we can store $\mathcal{E}$ in a single A100 GPU up to $d = 9$ for general stabilizer codes and up to $d = 11$ for CSS codes, which can be further reduced by using 1 bit to store a binary or by using a sparse representation.

4. Potential avenues for advanced RL techniques

Here, we outline some potential avenues for future work to scale our approach by using more advanced RL techniques.

In our RL algorithm, we use the tableau representation as the agent's observation and a fully connected neural network as our policy and value networks, which does not scale well with the size of the observation. A more advanced neural network architecture—such as convolutional, recurrent, or graph neural networks—could be used to replace the policy and value networks to scale our method. However, one may need a better and more efficient representation of the observation that fits this architecture.

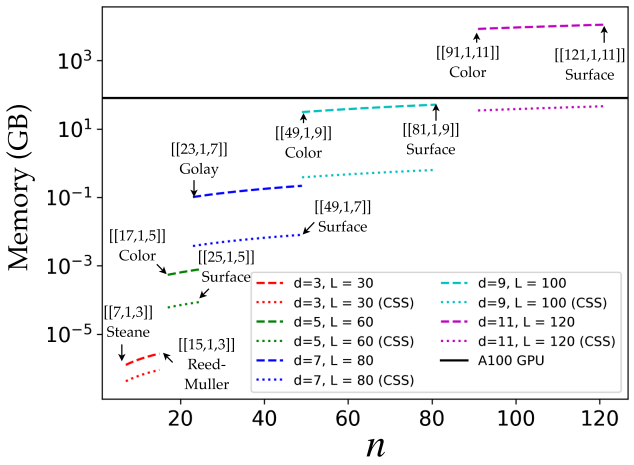


FIG. 25. Memory needed to store errors with varying distances and number of physical qubits (n) for general stabilizer codes and for CSS codes. Here, $L = G_1 + G_2$, where we assume that the number of one-qubit gates (G_1) is always $n/2$ and the rest are the two-qubit gates (G_2). The typical 80-GB memory of an A100 GPU is indicated just for reference.

Another potential candidate is the use of collaborative or multi-agent RL techniques, such as hierarchical structures of agents working on partial encodings, which also offer promising directions for tackling larger codes and more complex protocols. Multi-agent RL is an active area of research in the machine learning community [144,145], and to the best of our knowledge, it has not been applied to quantum problems.

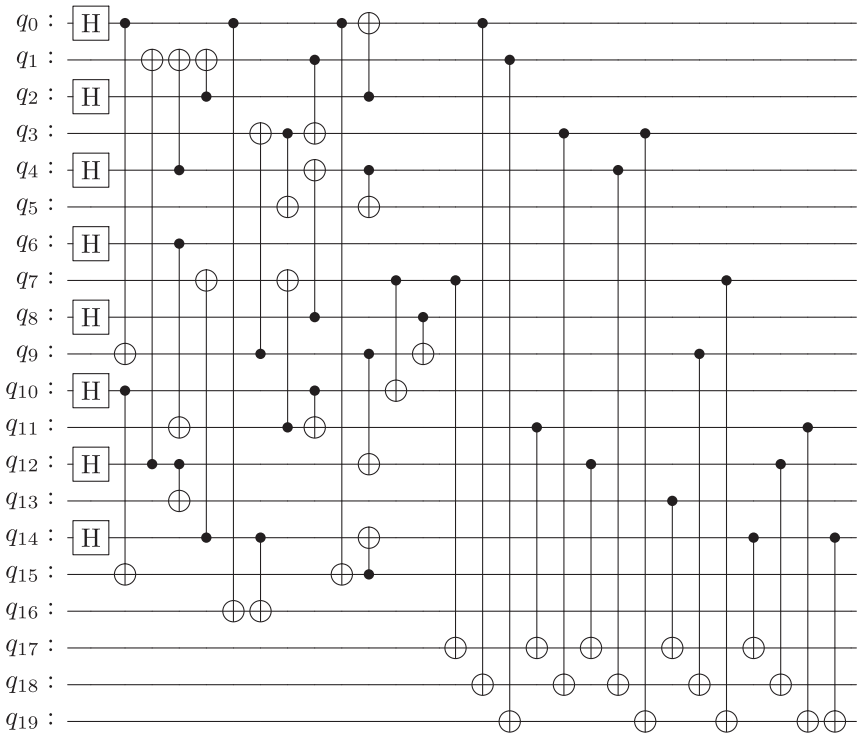
The main idea is to have multiple agents that take care of finding partial encodings on parts of the code and then communicate and coordinate to come up with a valid overall encoding. The idea of transfer learning also works here since one can train agents to solve individual tasks first and then reuse them to solve the combined task together. It would be an interesting avenue for future work to see and study the dynamics of the agents.

We outline how to potentially tackle the challenge of finding encodings for the surface code. One could divide the $[[25, 1, 5]]$ surface code into several patches, which are handled by several collaborative agents. These agents will share some qubits at the boundaries of the patches. Another technique is to use a transfer learning protocol, where one first trains an agent to find an encoding for the $[[9, 1, 3]]$ surface code and then reuses and copies that agent into four agents that take the four different patches of the $[[25, 1, 5]]$ surface code and fine-tune them in a multi-agent setting.

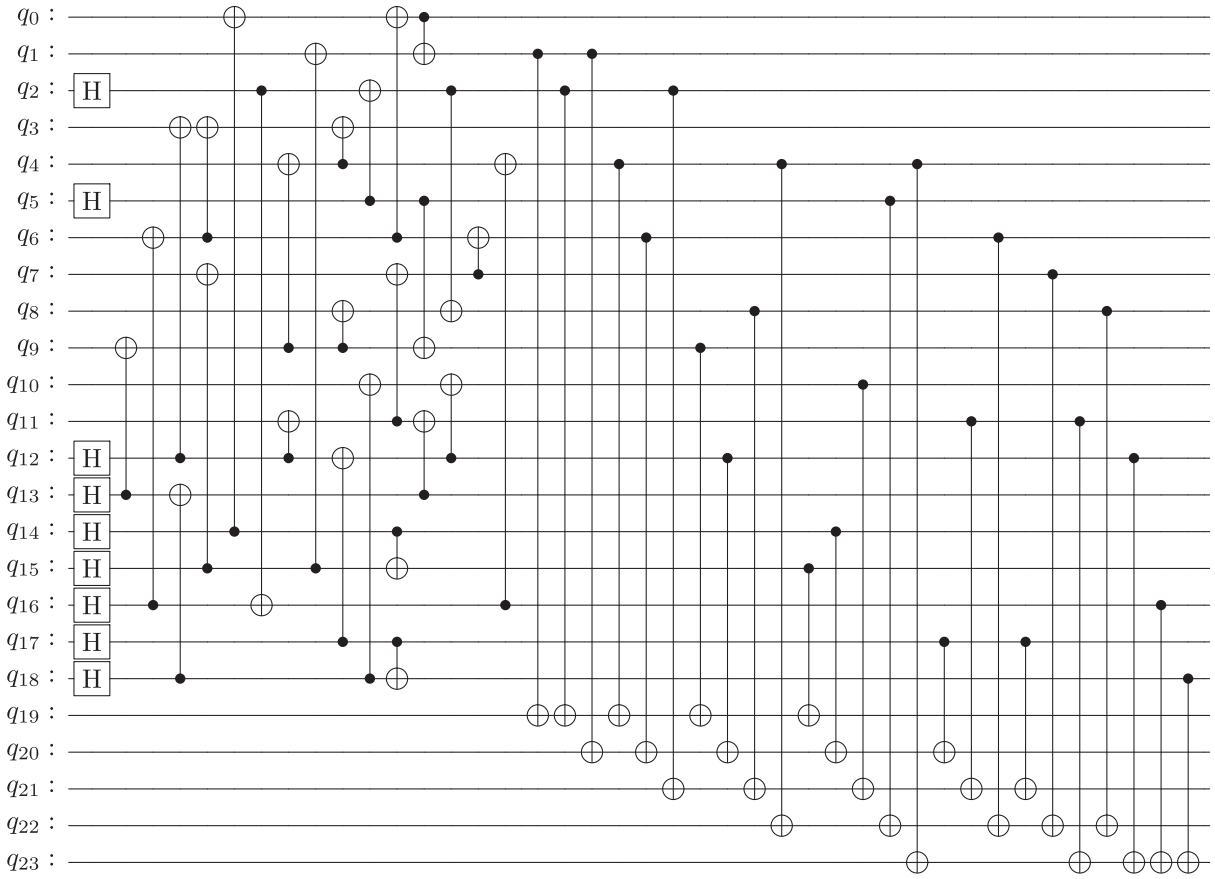
APPENDIX S: DISTANCE-5 CIRCUITS

Here, we show more examples of RL-discovered circuits for the distance-5 CSS codes shown in Fig. 13. These circuits are also available online [107]. The flag qubits are always the last qubits in the quantum circuit.

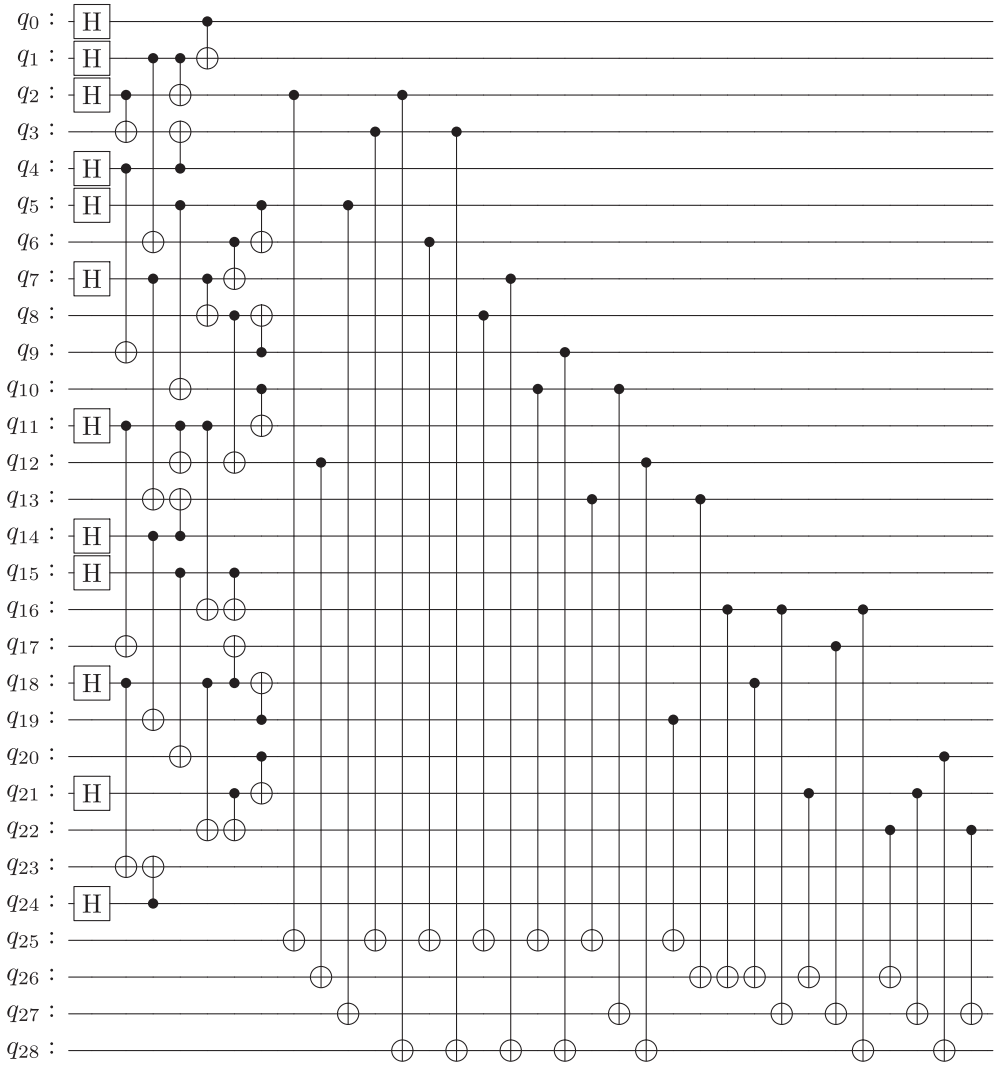
We consider the $|\mathbf{0}\rangle_{\mathbf{L}}$ state of the $[[\mathbf{17}, \mathbf{1}, \mathbf{5}]]$ color code with three flag qubits.



Next, we consider the $|\mathbf{0}\rangle_{\mathbf{L}}$ state of the $[[\mathbf{19}, \mathbf{1}, \mathbf{5}]]$ color code with five flag qubits.



Finally, we look at the $|\mathbf{0}\rangle_{\mathbf{L}}$ state of the $[[\mathbf{25}, \mathbf{1}, \mathbf{5}]]$ surface code with four flag qubits.



[1] J. Preskill, *Quantum computing in the NISQ era and beyond*, Quantum **2**, 79 (2018).

[2] D. Gottesman, *An introduction to quantum error correction and fault-tolerant quantum computation*, in *Proceedings of Symposia in Applied Mathematics*, edited by S. Lomonaco (American Mathematical Society, Providence, Rhode Island, 2010), Vol. 68, 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 (London) **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 (London) **605**, 669 (2022).

[6] C. Ryan-Anderson *et al.*, *Implementing fault-tolerant entangling gates on the five-qubit code and the color code*, arXiv:2208.01863.

[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 (London) **606**, 884 (2022).

[8] Y. Zhao *et al.*, *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 color code*, Sci. Adv. **10**, eado9024 (2024).
[10] D. Bluvstein *et al.*, *Logical quantum processor based on reconfigurable atom arrays*, Nature (London) **626**, 58 (2024).
[11] K. Mayer *et al.*, *Benchmarking logical three-qubit quantum Fourier transform encoded in the Steane code on a trapped-ion quantum computer*, arXiv:2404.08616.
[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 (London) **616**, 50 (2023).
[13] R. Acharya, I. Aleiner, R. Allen *et al.*, *Suppressing quantum errors by scaling a surface code logical qubit*, Nature (London) **614**, 676 (2023).
[14] M. P. da Silva *et al.*, *Demonstration of logical qubits and repeated error correction with better-than-physical error rates*, arXiv:2404.02280.
[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*, Proc. R. Soc. A **454**, 365 (1998).
[17] A. Kitaev, *Fault-tolerant quantum computation by anyons*, Ann. Phys. (Amsterdam) **303**, 2 (2003).
[18] E. T. Campbell, B. M. Terhal, and C. Vuillot, *Roads towards fault-tolerant universal quantum computation*, Nature (London) **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*, PRX Quantum **5**, 010333 (2024).
[22] J. Preskill, *Reliable quantum computers*, Proc. R. Soc. A **454**, 385 (1998).
[23] P. Shor, *Fault-tolerant quantum computation*, in *Proceedings of 37th Conference on Foundations of Computer Science* (1996), pp. 56–65, 10.1109/SFCS.1996.548464.
[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*, Sci. Adv. **10**, eadp2008 (2024).
[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*, PRX Quantum **5**, 030326 (2024).
[27] T. J. Yoder and I. H. Kim, *The surface code with a twist*, Quantum **1**, 2 (2017).
[28] C. Chamberland and M. E. Beverland, *Flag fault-tolerant error correction with arbitrary distance codes*, Quantum **2**, 53 (2018).
[29] R. Chao and B. W. Reichardt, *Quantum error correction with only two extra qubits*, Phys. Rev. Lett. **121**, 050502 (2018).
[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*, Quantum Inf. Comput. **12**, 1034 (2012).
[34] F. Butt, S. Heußen, M. Rispler, and M. Müller, *Fault-tolerant code-switching protocols for near-term quantum processors*, PRX Quantum **5**, 020345 (2024).
[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).
[37] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, T. M. Gatterman, S. K. Halit, K. Gilmore, J. A. Gerber, B. Neyenhuis, D. Hayes, and R. P. 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. G. 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*, arXiv:2403.13732.
[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. S. Humble, and A. Li, *QASMTrans: A QASM based quantum transpiler framework for NISQ devices*, arXiv:2308.07581.
[42] E. Younis and C. Iancu, *Quantum circuit optimization and transpilation via parameterized circuit instantiation*, in *2022 IEEE International Conference on Quantum 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, Quantum Sci. Technol. **9**, 033001 (2024).
[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).
[46] N. Paraskevopoulos, F. Sebastiano, C. G. Almudever, and S. Feld, *SpinQ: Compilation strategies for scalable spin-qubit architectures*, arXiv:2301.13241.
[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).
[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).
[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).
[53] R. Sweke, M. S. Kesselring, E. P. L. Van Nieuwenburg, and J. Eisert, *Reinforcement learning decoders for fault-tolerant quantum computation*, Mach. Learn. **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*, arXiv:2311.04750.
[55] M. Puviani, S. Borah, R. Zen, J. Olle, and F. Marquardt, *Boosting the Gottesman-Kitaev-Preskill quantum error correction with non-Markovian feedback*, arXiv:2312.07391.
[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. R. 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).
[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. **2**, 01LT02 (2021).
[60] V. V. Sivak, A. Eickbusch, H. Liu, B. Royer, I. Tsioutsios, and M. H. 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. Res. **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).
[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*, arXiv:2204.06904.
[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. Res. **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*, arXiv:2311.04285.
[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*, arXiv:2307.05744.
[69] S. Rietsch, A. Y. Dubey, C. Ufrecht, M. Periyasamy, A. Plinge, C. Mutschler, and D. D. Scherer, *Unitary synthesis of* Clifford + T *circuits with reinforcement learning*, in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)* (2024), pp. 824–835, arXiv:2404.14865.
[70] D. Kremer, V. Villar, H. Paik, I. Duran, I. Faro, and J. Cruz-Benito, *Practical and efficient quantum circuit synthesis and transpiling with reinforcement learning*, arXiv:2405.13196.
[71] N. Shutty and C. Chamberland, *Decoding merged color-surface codes and finding fault-tolerant Clifford circuits using solvers for satisfiability modulo theories*, Phys. Rev. Appl. **18**, 014072 (2022).
[72] D. Gottesman, *Stabilizer codes and quantum error correction*, arXiv:quant-ph/9705052.
[73] A. Steane, *Multiple-particle interference and quantum error correction*, Proc. R. Soc. A **452**, 2551 (1996).
[74] A. R. Calderbank and P. W. Shor, *Good quantum error-correcting codes exist*, Phys. Rev. A **54**, 1098 (1996).
[75] A. Y. Kitaev, *Quantum computations: Algorithms and error correction*, Russ. Math. Surv. **52**, 1191 (1997).
[76] H. Bombín, *Gauge color codes: Optimal transversal gates and gauge fixing in topological stabilizer codes*, New J. Phys. **17**, 083002 (2015).
[77] H. Bombin and M. A. Martin-Delgado, *Topological quantum distillation*, Phys. Rev. Lett. **97**, 180501 (2006).
[78] S. Aaronson and D. Gottesman, *Improved simulation of stabilizer circuits*, Phys. Rev. A **70**, 052328 (2004).
[79] S. Bravyi, J. A. Latone, and D. Maslov, *6-qubit optimal Clifford circuits*, npj Quantum Inf. **8**, 79 (2022).
[80] V. Kliuchnikov and D. Maslov, *Optimization of Clifford circuits*, Phys. Rev. A **88**, 052307 (2013).
[81] S. Bravyi, R. Shaydulin, S. Hu, and D. Maslov, *Clifford circuit optimization with templates and symbolic Pauli gates*, Quantum **5**, 580 (2021).

[82] 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.
[83] D. Winder, Q. Huang, A. M.-v. de Griend, and R. Yeung, *Architecture-aware synthesis of stabilizer circuits from Clifford tableaus*, arXiv:2309.08972.
[84] 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.
[85] D. Amaro, M. Müller, and A. K. Pal, *Scalable characterization of localizable entanglement in noisy topological quantum codes*, New J. Phys. **22**, 053038 (2020).
[86] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10th ed. (Cambridge University Press, Cambridge; New York, 2010).
[87] N. Rengaswamy, R. Calderbank, S. Kadhe, and H. D. Pfister, *Logical Clifford synthesis for stabilizer codes*, IEEE Trans. Quantum Eng. **1**, 1 (2020).
[88] A. Mondal and K. K. Parhi, *Optimization of quantum circuits for stabilizer codes*, IEEE Trans. Circuits Syst. I **71**, 3647 (2024).
[89] D. Tandeitnik and T. Guerreiro, *Evolving quantum circuits*, Quantum Inf. Process. **23**, 109 (2024).
[90] A. M. Steane, *Overhead and noise threshold of fault-tolerant quantum error correction*, Phys. Rev. A **68**, 042322 (2003).
[91] 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).
[92] E. Knill, *Quantum computing with realistically noisy devices*, Nature (London) **434**, 39 (2005).
[93] T. Tansuwannont and D. Leung, *Achieving fault tolerance on capped color codes with few ancillas*, PRX Quantum **3**, 030322 (2022).
[94] K. C. Miao *et al.*, *Overcoming leakage in quantum error correction*, Nat. Phys. **19**, 1780 (2023).
[95] C. Ryan-Anderson *et al.*, *High-fidelity and fault-tolerant teleportation of a logical qubit using transversal gates and lattice surgery on a trapped-ion quantum computer*, arXiv:2404.16728.
[96] H. Bombín, M. Pant, S. Roberts, and K. I. Seetharam, *Fault-tolerant post-selection for low overhead magic state preparation* (2022).
[97] 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*, arXiv:2009.11482.
[98] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, *Policy gradient methods for reinforcement learning with function approximation*, arXiv:1706.06643.
[99] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, arXiv:1707.06347.
[100] T. Fösel, M. Y. Niu, F. Marquardt, and L. Li, *Quantum circuit optimization with deep reinforcement learning*, arXiv:2103.07585.
[101] M. Ostaszewski, L. M. Trenkwalder, W. Masarczyk, E. Scerri, and V. Dunjko, *Reinforcement learning for optimization of variational quantum circuit architectures*, Adv. Neural Inf. Process. Syst. **34**, 18182 (2021).
[102] 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.
[103] X. Xu, S. C. Benjamin, and X. Yuan, *Variational circuit compiler for quantum error correction*, Phys. Rev. Appl. **15**, 034068 (2021).
[104] 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).
[105] C. Lu, J. G. Kuba, A. Letcher, L. Metz, C. S. de Witt, and J. Foerster, *Discovered policy optimisation*, Adv. Neural Inf. Process. Syst. **35**, 16455 (2022).
[106] 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).
[107] https://github.com/remmyzen/rlftqc.
[108] S. Z. Baba, N. Yoshioka, Y. Ashida, and T. Sagawa, *Deep reinforcement learning for preparation of thermal and prethermal quantum states*, Phys. Rev. Appl. **19**, 014068 (2023).
[109] J. Mackeprang, D. B. R. Dasari, and J. Wrachtrup, *A reinforcement learning approach for quantum state engineering*, Quantum Mach. Intell. **2**, 5 (2020).
[110] R. Laflamme, C. Miquel, J. P. Paz, and W. H. Zurek, *Perfect quantum error correcting code*, Phys. Rev. Lett. **77**, 198 (1996).
[111] P. W. Shor, *Scheme for reducing decoherence in quantum computer memory*, Phys. Rev. A **52**, R2493 (1995).
[112] 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. Appl. **16**, 024057 (2021).
[113] 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).
[114] 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).
[115] 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 (London) **604**, 451 (2022).
[116] 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).
[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)*, 10.5281/zenodo.2573505.

[119] FakeManilaV2, https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.fake_provider.FakeManilaV2 (accessed: 2024 − 02 − 20).
[120] FakeJakartaV2, https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.fake_provider.FakeJakartaV2 (accessed: 2024-02-20).
[121] FakeGuadalupeV2, https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.fake_provider.FakeGuadalupeV2 (accessed: 2024-02-20).
[122] FakeTokyo, https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.fake_provider.FakeTokyo (accessed: 2024-02-20).
[123] M. E. Taylor and P. Stone, *Transfer learning for reinforcement learning domains: A survey*, J. Mach. Learn. Res. **10** (2009).
[124] 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).
[125] C. Gidney, *Stim: A fast stabilizer circuit simulator*, Quantum **5**, 497 (2021).
[126] F. Arute, K. Arya, R. Babbush *et al.*, *Quantum supremacy using a programmable superconducting processor*, Nature (London) **574**, 505 (2019).
[127] 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 (London) **625**, 259 (2024).
[128] 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).
[129] Y. Ye *et al.*, *Logical magic state preparation with fidelity beyond the distillation threshold on a superconducting quantum processor*, Phys. Rev. Lett. **131**, 210603 (2023).
[130] S. Heußen, L. Postler, M. Rispler, I. Pogorelov, C. D. Marciniak, T. Monz, P. Schindler, and M. Müller, *Strategies for a practical advantage of fault-tolerant circuit design in noisy trapped-ion quantum computers*, Phys. Rev. A **107**, 042422 (2023).
[131] Y. Tomita and K. M. Svore, *Low-distance surface codes under realistic quantum noise*, Phys. Rev. A **90**, 062320 (2014).
[132] R. Acharya *et al.*, *Quantum error correction below the surface code threshold*, arXiv:2408.13687.
[133] N. Berthusen, J. Dreiling, C. Foltz, J. P. Gaebler, T. M. Gatterman, D. Gresh, N. Hewitt, M. Mills, S. A. Moses, B. Neyenhuis, P. Siegfried, and D. Hayes, *Experiments with the 4D surface code on a QCCD quantum computer*, Phys. Rev. A **110**, 062413 (2024).
[134] B. W. Reichardt, D. Aasen, R. Chao, A. Chernoguzov, W. v. Dam, J. P. Gaebler, D. Gresh, D. Lucchetti, M. Mills, S. A. Moses, B. Neyenhuis, A. Paetznick, A. Paz, P. E. Siegfried, M. P. d. Silva, K. M. Svore, Z. Wang, and M. Zanner, *Demonstration of quantum computation and error correction with a tesseract code*, arXiv:2409.04628.
[135] B. Heténi and J. R. Wootton, *Creating entangled logical qubits in the heavy-hex lattice with topological codes*, PRX Quantum **5**, 040334 (2024).
[136] B. W. Reichardt *et al.*, *Logical computation demonstrated with a neutral atom quantum processor*, arXiv:2411.11822.
[137] N. Delfosse and B. W. Reichardt, *Short Shor-style syndrome sequences*, arXiv:2008.05051.
[138] 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)* (IEEE, Bellevue, WA, 2023), pp. 63–69.
[139] S. Kosub, *A note on the triangle inequality for the Jaccard distance*, arXiv:1612.02696.
[140] FakeLimaV2, https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.fake_provider.FakeLimaV2 (accessed: 2024-02-20).
[141] E. Knill and R. Laflamme, *Concatenated quantum codes*, Technical Report LA-UR–96-2808, 369608, 1996.
[142] M. Grassl, P. Shor, G. Smith, J. Smolin, and B. Zeng, *Generalized concatenated quantum codes*, Phys. Rev. A **79**, 050306(R) (2009).
[143] D. Gottesman, Surviving as a quantum computer in a classical world (2024), preprint on webpage at https://www.cs.umd.edu/class/spring2024/cmsc858G.
[144] S. Gronauer and K. Diepold, *Multi-agent deep reinforcement learning: A survey*, Artif. Intell. Rev. **55**, 895 (2022).
[145] S. V. Albrecht, F. Christianos, and L. Schäfer, *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches* (MIT Press, Cambridge, MA, 2024).