

Efficient Magic State Cultivation on $\mathbb{RP}^2$ Zi-Han Chen^{1,2,3,*}, Ming-Cheng Chen^{1,2,3,†}, Chao-Yang Lu^{1,2,3,‡} and Jian-Wei Pan^{1,2,3,§}¹Hefei National Research Center for Physical Sciences at the Microscale and School of Physical Sciences, University of Science and Technology of China, Hefei 230026, China²Shanghai Research Center for Quantum Science and CAS Center for Excellence in Quantum Information and Quantum Physics, University of Science and Technology of China, Shanghai 201315, China³Hefei National Laboratory, University of Science and Technology of China, Hefei 230088, China

(Received 4 August 2025; revised 15 October 2025; accepted 19 November 2025; published 22 January 2026)

Preparation of high-fidelity logical magic states is crucial for fault-tolerant quantum computation. Among previous attempts to reduce the substantial cost of magic state preparation, magic state cultivation (MSC) [Gidney *et al.*, Magic state cultivation: growing T states as cheap as CNOT gates, [arXiv:2409.17595](#)], a recently proposed protocol for logical T state preparation without magic state distillation, achieves state-of-the-art efficiency. Inspired by this work [Gidney *et al.*, Magic state cultivation: growing T states as cheap as CNOT gates, [arXiv:2409.17595](#)], we propose a MSC procedure that can produce logical T states on the rotated surface code at a further reduced cost. To maintain high efficiency throughout our protocol, we design structured codes along with compact circuits bridging between them. More specifically, we construct a code family, the RP code, by putting the rotated surface code on $\mathbb{RP}^2$ (a two-dimensional manifold), as well as two self-dual Calderbank-Shor-Steane codes, named SRP-3 and SRP-5, respectively. In our MSC protocol, we start with a cultivation process, in which a high-fidelity T state is prepared on a small RP code with distance 3 or 5. Then, to preserve the logical T state, we use an efficient and easy-to-decode expansion stage to grow a small RP code to a larger rotated surface code in one syndrome extraction (SE) round. The RP code serves as an efficient transfer station with efficient SE circuits and compact interfaces between the SRP-3 (or SRP-5) code—used in the cultivation process to efficiently verify the correctness of the logical T state—and larger rotated surface codes for preserving the prepared logical T state. Our MSC protocol utilizes nonlocal connectivity, available on both neutral atom array and ion trap platforms. According to our Monte Carlo sampling results, our MSC protocol requires about an order of magnitude smaller space-time volume to reach a target logical error rate of around 10^{-9} compared to the original MSC protocol.

DOI: [10.1103/9kys-3whh](#)

Fault-tolerant quantum computation (FTQC) is necessary, given noisy physical operations, to implement quantum algorithms with arbitrarily high precision. The rotated surface code [1–3], with high circuit noise threshold $\sim 1\%$, is promising for practical implementations of FTQC and admits experimental demonstration definitively below threshold [4]. Tremendous theoretical progress has been

made in surface code-based logical processing over the last two decades, including the construction and optimization of logical Clifford operations [3,5–13] and magic state preparations [5,6,14–22] (magic states are used to implement non-Clifford gates via gate teleportation [23,24]). There has been extensive efforts [5,6,15,18–22,25–28] to lower the substantial cost of preparing magic states (with low logical error rates under a practically relevant physical noise rate $\sim 10^{-3}$). Recently, Gidney *et al.* proposed a new T state preparation protocol, magic state cultivation (MSC) [29], which can produce high-fidelity T states with roughly the same space-time cost as performing lattice surgery CNOT gates of the same fidelity.

For physical systems such as neutral atom arrays [30] and ion traps [31] that support nonlocal connectivity, logical Clifford gates on the rotated surface code can be implemented transversally and have considerably smaller space-time cost compared to those implemented via lattice

*Contact author: czh007@mail.ustc.edu.cn†Contact author: cmc@ustc.edu.cn‡Contact author: cylu@ustc.edu.cn§Contact author: pan@ustc.edu.cn

Published by the American Physical Society under the terms of the [Creative Commons Attribution 4.0 International](#) license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

surgery, since transversal Clifford gates only require $O(1)$ SE rounds per gate with no additional spatial overhead [32–35]. Therefore, the cost of logical T states produced via the MSC protocol in Ref. [29] is still considerably higher than that of transversal logical Clifford gates. This strongly motivates a T state preparation protocol with a further reduced cost for platforms with nonlocal connectivity.

Based on the observation that certain magic states (including T states and CCZ states) are stabilized by Clifford operators, there is a long line of works that enhance the fidelity of logical magic states by measuring their corresponding Clifford stabilizers [21–24,26,29,36,37]. More recently, this technique has been used in a few protocols [21,22,26,29] (including the MSC protocol [29]) to prepare a high-fidelity T state on a small code, which should then grow to a larger code to protect and preserve the logical T state. Thus, a high-performance protocol based on this technique should simultaneously achieve high efficiency in both the logical T state preparation process on the small code and the code growth process. The MSC procedure in Ref. [29] cultivates a high-fidelity T state—by verifying the correctness of the injected logical T state through an implementation (called logical double-checking in Ref. [29]) of measuring and resetting the logical Clifford stabilizer $H_{XY} = \frac{1}{\sqrt{2}}(X + Y)$ —on a small color code, which is then “grafted” to a larger matchable code—essentially a rotated surface code deformed at a corner. While the cultivation process (with a small color code) is highly efficient, the “grafting” process, used to bypass the color code growth decoding problem, incurs additional postselection and syndrome extraction (SE) rounds [29]. Furthermore, it would be more convenient to prepare T states directly on regular rotated surface codes, since these T states can be teleported [23,24] more efficiently via transversal CNOT gates (instead of lattice surgery CNOT gates) to apply logical T gates on other regular rotated surface codes. Recently, another MSC procedure [38] is proposed and has a more efficient expansion stage, which grows a smaller surface code to a larger one. However, this procedure lacks an efficient cultivation process, unlike the original MSC procedure, and is overall less efficient than the original MSC procedure. Thus, until this work, it remains an open challenge to design a (Clifford measurement-based) T state preparation protocol for the rotated surface code that maintains high efficiency throughout the protocol.

In this work, we propose a MSC protocol that simultaneously has an efficient cultivation process and an efficient and easy-to-decode expansion and stabilization (ES) process. Moreover, our MSC protocol uses nonlocal connectivity and produces T states on regular rotated surface codes. Our protocol features a global functionality-oriented design, in which codes with special structures for specific purposes are *codesigned* with

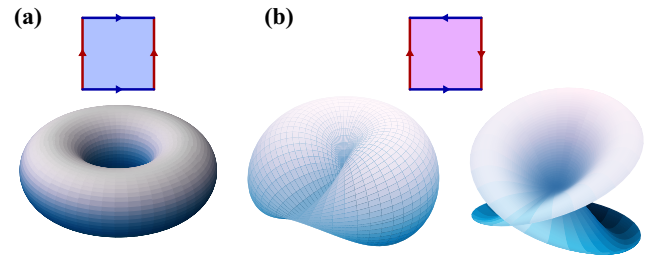


FIG. 1. Torus and $\mathbb{RP}^2$ obtained by identifying the sides of a square. Red (blue) arrows on the sides of a square patch are identified. (a) Illustration of the torus. (b) Illustration of the $\mathbb{RP}^2$. The shape (representing the $\mathbb{RP}^2$) in the middle is obtained from the shape on the right by gluing together its upper and lower rims.

compact *morphing* circuits that can efficiently bridge between these codes. In this way, we tailor our fault-tolerant circuits to the desired end-to-end logical functionality by compactly knitting together efficient fault-tolerant components. In this work, we first design a code family, which we call the RP code, such that it has a certain symmetry—*fold-duality* [39] (introduced in Sec. III)—and is structurally similar to the rotated surface code. More specifically, the RP code is constructed by placing the rotated surface code on $\mathbb{RP}^2$. Although previous works have explored surface codes and, more generally, topological order on $\mathbb{RP}^2$ [40–42], our construction is based on a different cellulation of $\mathbb{RP}^2$ to endow our RP code with fold-duality. See Fig. 1 for an illustration of the torus, on which the toric code [43] (the predecessor of the surface code) is defined, and the $\mathbb{RP}^2$. For the distance-3 RP-3 code (or distance-5 RP-5 code) in the RP code family, we then design a self-dual Calderbank-Shor-Steane (CSS) code, the SRP-3 (or SRP-5) code, together with a compact morphing circuit that enables fast transfer between the RP-3 (or RP-5) code and the SRP-3 (or SRP-5) code. The motivations behind the structural requirements for the codes constructed above are as follows.

- (1) The self-duality of the SRP code implies that it admits a transversal logical H_{XY} operator, therefore facilitating the measurement of the logical H_{XY} operator.
- (2) Fold-duality of the RP code is enforced to guarantee the existence of a compact morphing circuit that can morph the RP code to a self-dual CSS code (an SRP code).
- (3) The structural similarity requirement makes it easy to find an efficient way to expand the RP code to a larger rotated surface code.

According to Monte Carlo sampling results, our MSC protocol requires nearly an order of magnitude (or half an order of magnitude) smaller space-time cost to reach a logical error rate around 1×10^{-9} (or 1.5×10^{-6}) than the

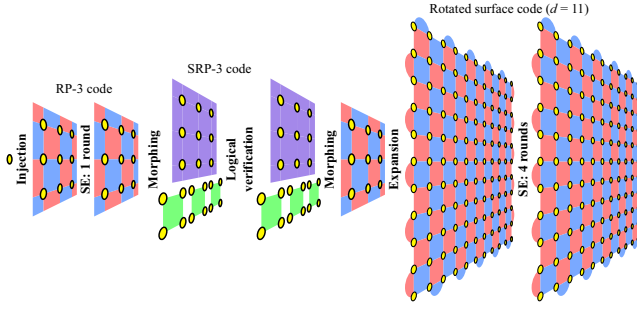


FIG. 2. The MSC-3 protocol. Data qubits are marked by yellow circles. Red (blue) tiles represent Pauli X (Z) stabilizers. Each stage of the protocol is illustrated between two code patches. A few stabilizers of the $[[15, 1, 3]]$ SRP-3 code are illustrated by green squares, each of which represents an X stabilizer and a Z stabilizer. Not all stabilizers of the SRP-3 code are illustrated. Within each slice of the SRP-3 code patches, some of the data qubits on the green squares are identified with qubits on the purple patch. The final syndrome extraction round on the rotated surface code is implemented by a perfect syndrome extraction circuit followed by a perfect logical readout.

original MSC protocol [29]. We overview our MSC protocol and results in the following section (Sec. I), at the end of which the structure of the rest of the paper is laid out.

I. RESULTS

Our MSC protocol has two versions, a smaller version named MSC-3 (Fig. 2) and a full-size version named MSC-5 that can reach lower logical error rates (Fig. 3). Both versions contain two processes, a cultivation process for producing a high-fidelity T state on a smaller code and an ES process for transferring the logical qubit from the smaller code to a larger rotated surface code. During the

cultivation process of the MSC-3 (MSC-5) procedure, a single-qubit T state is grown (gradually grown) to an RP-3 (RP-5) code with distance 3 (5). SE rounds and logical verifications are applied at the intervals of code growth to improve the fault distance, as in Ref. [29]. Interestingly, these SE rounds are only performed on RP codes, and logical verifications are only performed on SRP codes. Each MSC attempt would be discarded if any detector is triggered during the cultivation process. During the ES process of our MSC-3 (MSC-5) protocol, we first expand the RP-3 (RP-5) code to a distance-11 rotated surface code; then apply a few SE rounds on the rotated surface code (Figs. 2 and 3). Here, our choice of the distance of the rotated surface code ensures a fair comparison with the original MSC protocol (see Appendix B). Similar to Refs. [29,38], at the end of the ES process, another round of postselection, which we call the final-stage postselection, is implemented to discard MSC attempts that survive the cultivation process but may contain errors that would significantly degrade the logical fidelity. In Refs. [29,38], a soft output named complementary gap is calculated to estimate the decoding confidence for each decoding shot and is used to decide whether the shot should be discarded for the final-stage postselection. In our work, we adopt a different method proposed in Ref. [44] to obtain soft outputs. In particular, we extract a *two-dimensional soft-output* $(\phi_{\mathbb{RP}^2}, \phi_{bd}) \in \mathbb{Z}_{\geq 0}^2$ for each shot and set cutoffs on both $\phi_{\mathbb{RP}^2}$ and ϕ_{bd} for final-stage postselection. Exact simulation of our MSC protocols is inefficient due to the presence of T and $T^\dagger$ gates. Following Ref. [29], we instead replace all T ($T^\dagger$) gates in our protocols with S ($S^\dagger$) gates and perform Monte Carlo sampling on such altered circuits (cultivating $S|+\rangle$ states instead of T states) under uniform depolarizing circuit noise [29] with noise rate $p = 0.001$. It is observed in Ref. [29] that the error rate for cultivating a

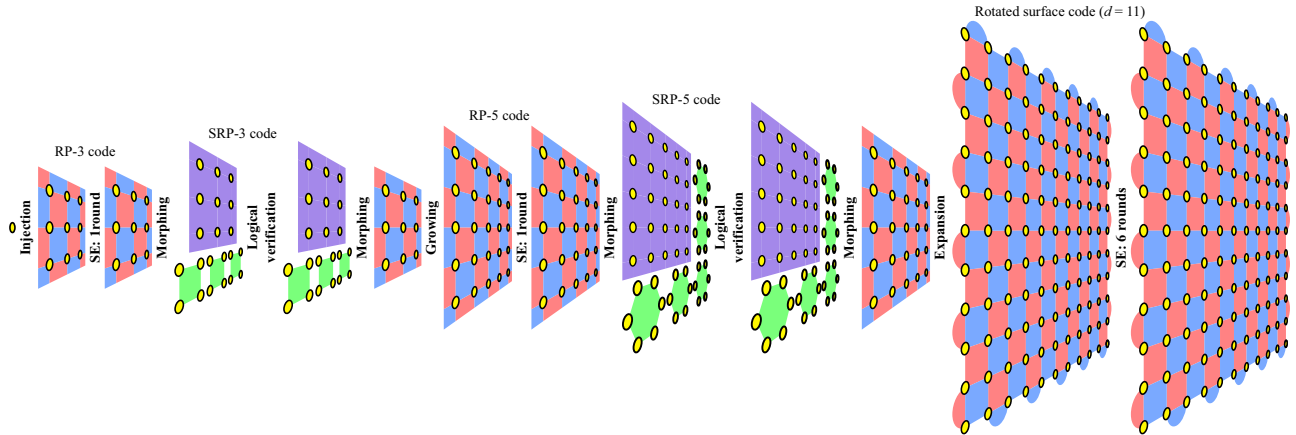


FIG. 3. The MSC-5 protocol. The convention of the illustration follows from Fig. 2. A few stabilizers of the $[[35, 1, 5]]$ SRP-5 code are illustrated by green hexagons, each of which represents an X stabilizer and a Z stabilizer. Not all stabilizers of the SRP-5 code are illustrated. Within each slice of the SRP-5 code patches, some of the data qubits on the green hexagons are identified with qubits on the purple patch. The final syndrome extraction round on the rotated surface code is implemented by a perfect syndrome extraction circuit followed by a perfect logical readout.

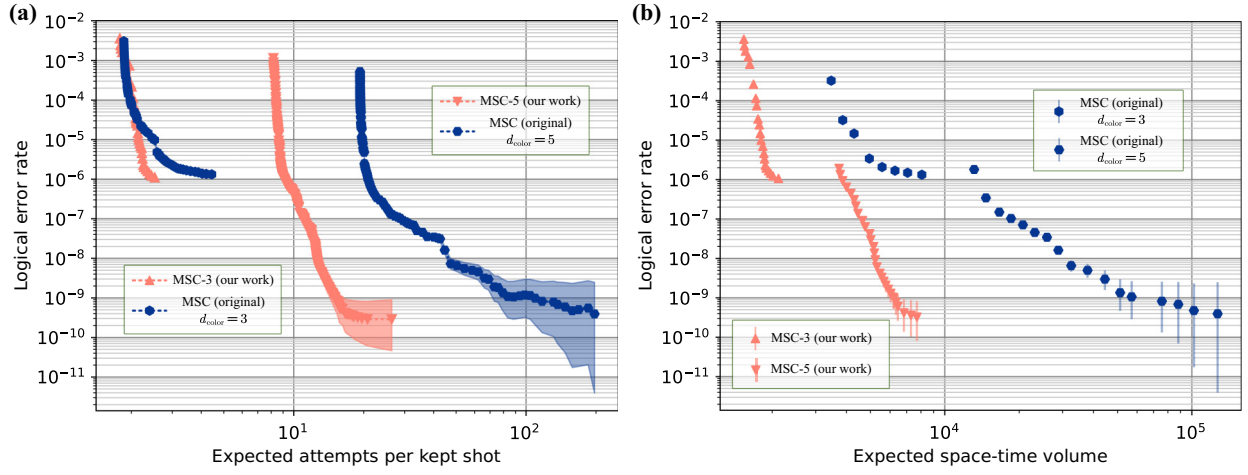


FIG. 4. Monte Carlo sampling results under uniform depolarizing circuit noise [29] with noise rate $p = 0.001$ for our magic state cultivation protocol and the original magic state cultivation protocol (taken directly from Ref. [29]). Here d_{color} is the distance of the color code right before the “grafting” process in the original magic state cultivation protocol [29]. (a) Trade-offs between the logical error rate and the expected number of attempts per kept shot. (b) Expected space-time cost to reach a certain logical error rate. Each data point of the original magic state cultivation protocol corresponds to a particular cutoff on the complementary gap value [29]. Similarly, each data point of our magic state cultivation protocol corresponds to a specific cutoff configuration on ϕ_{RP^2} and ϕ_{bd} . By varying the cutoffs, we can achieve different trade-offs between the logical error rate and the success rate (see Fig. 13 for the logical error rate and success rate at each cutoff configuration). Logical error rates are obtained via maximum likelihood estimation over collected samples [45]. Error bars and shaded regions both indicate logical error rate hypotheses whose likelihood is within a factor of 1000 of the maximum likelihood. See Appendix A for more details (including the noise model and evaluation of the space-time cost) on our simulation.

T state on a distance-3 color code is about two times larger than the error rate for cultivating an $S|+\rangle$ state. Future work is required to implement an exact simulator efficient enough to benchmark our cultivation process on the RP-3 code. In this work, we compare only the Clifford variant of our MSC protocol with that of the original MSC protocol [29]. Thanks to our highly efficient ES process, our MSC protocol has higher end-to-end survival rates [Fig. 4(a) and Table I] and requires a smaller code size at the end than the original MSC protocol [29] to achieve the same logical

error rates. As shown in Fig. 4(b), our MSC-5 (MSC-3) protocol requires nearly an order of magnitude (or half an order of magnitude) smaller space-time cost to achieve a logical error rate around 1×10^{-9} (1.5×10^{-6}) compared to the original MSC protocol [29]. Our MSC protocol requires nonlocal connectivity—for instance, the RP code contains nonlocal stabilizers. In addition, morphing an RP-3 (RP-5) code to an SRP-3 (SRP-5) code requires “folding” the RP code patch. Since the cultivation process is performed on small codes, the required nonlocality is also

TABLE I. Selected Monte Carlo sampling results under uniform depolarizing circuit noise [29] with noise rate $p = 0.001$. Here d_{color} is the distance of the color code right before the “grafting” process in the original magic state cultivation protocol [29]. Performance of each cultivation process alone (only producing T states on a small code) is labeled by “Ungrown”. Benchmarking the cultivation process alone is implemented by sampling its Clifford variant, which cultivates $S|+\rangle$ on the small code, appended by a round of perfect syndrome extraction and logical readout. Full postselection (discarding the shots that trigger any detector) is used to obtain the logical error rate and discard rate for each ungrown case. Data corresponding to a full magic state cultivation protocol are labeled by “End-to-end.” The end-to-end results are selected from the results in Fig. 4. The distance of the code at the end of an magic state cultivation protocol is denoted as the final distance. The footprint of an magic state cultivation protocol is the maximum number of active qubits throughout the protocol. Our magic state cultivation protocol, with high efficiency throughout, achieves a higher end-to-end efficiency compared to the original magic state cultivation protocol [29].

MSC protocol	Logical error rate (Ungrown)	Discard rate (Ungrown)	Logical error rate (End-to-end)	Discard rate (End-to-end)	Final distance (End-to-end)	Footprint (End-to-end)
MSC-3 (this work)	8.3×10^{-7}	48.6%	1.5×10^{-6}	58%	11	247 qubits
MSC ($d_{\text{color}} = 3$) [29]	3.4×10^{-7}	31.3%	1.5×10^{-6}	74%	15	454 qubits
MSC-5 (this work)	3.6×10^{-10}	90.5%	1×10^{-9}	93.4%	11	251 qubits
MSC ($d_{\text{color}} = 5$) [29]	3.5×10^{-10}	85.6%	1×10^{-9}	98.9%	15	463 qubits

bounded to this small scale. See Appendix B for discussion on experimental implementation of our protocol.

We will elaborate on our MSC protocols and their decoding in the rest of the paper, which is structured as follows. In Sec. II, we describe the design of the RP code and its SE circuit. In Sec. III, we first present a general framework of transforming a CSS code with fold-duality (a special case of ZX-duality [39]) to a self-dual CSS code. As the RP code is designed to host fold-duality, we then define two self-dual codes, named SRP-3 and SRP-5, respectively, and describe how to morph an RP-3 (RP-5) code into an SRP-3 (SRP-5) code and back. Logical verification on the SRP codes is described in Sec. IV. Circuits for code growth, including injection and expansion, are illustrated in Sec. V. We describe how *two-dimensional soft outputs* are obtained during decoding in Sec. VI. Finally, we discuss our results in Sec. VII.

II. SURFACE CODES ON $\mathbb{RP}^2$

In this section, we first specify the construction of the RP code. Our construction makes the RP code fold-dual and is slightly different from prior constructions that place the surface code on $\mathbb{RP}^2$ [40,42]. As per our construction, an RP code with a patch parameter d (an odd positive integer), named RP- d , has code parameters $[[d^2, 1, d]]$, just as a rotated surface code with the same patch parameter. For example, stabilizers of the RP-3 code are illustrated in Fig. 5(a), where the stabilizers in the bulk are the same as the ones in the distance-3 rotated surface code. More interestingly, each antipodal pair of red (blue) tiles along

the boundary in Fig. 5(a) represents a *single* X (Z) stabilizer. In Fig. 5(a), weight-two stabilizers at the corners are redundant. [The product of all noncorner X (Z) stabilizer generators yields the corner X (Z) stabilizer.] As shown in Figs. 5(b) and 5(c), the logical X (Z) operator can be hosted on any noncontractable loop on $\mathbb{RP}^2$. More generally, the RP- d code can be defined similarly, with d being the number of qubits on a side of the square lattice (of data qubits) as in Fig. 5(a).

We initialize the circuit-level study of the RP code by designing its SE circuit with depth 5 (counting only the layers of CNOT gates per SE round). As we are particularly interested in the RP-3 code and the RP-5 code in the family, their SE circuits are illustrated in Figs. 5(d) and 5(e). It is worth noticing that the orientation of “zigzag” gate orderings for the X stabilizer measurements in the bulk is aligned with the logical X operator in Fig. 5(b). While such alignment would halve the circuit-level distance for a rotated surface code (which has open boundaries), in our case, we verified that our SE circuit for an RP- d code (which does not have a boundary) achieves the full circuit-level distance d for all $d \in \{3, 5, 7, 9, 11\}$ using `stim` [46]. As shown in Figs. 5(d) and 5(e), weight-two redundant stabilizers are also measured. These corner stabilizers are crucial for soft-output extraction [Fig. 12(c) in Sec. VI]. For our MSC protocol, since our specially designed injection and patch growth circuits fix the signs of the corner stabilizers, it is technically not necessary to measure them. We still include such measurements in our SE circuit for completeness and adaptability to other designs of injection or patch growth circuits. Ancilla qubits for corner and boundary stabilizer measurement are associated with nonlocal CNOT gates. See Appendix B for discussion on practical implementation of the SE circuit.

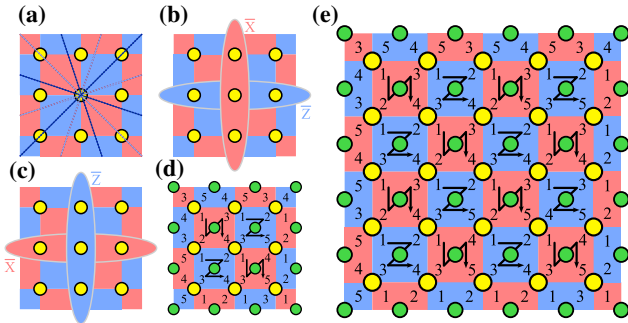


FIG. 5. The RP code. (a) Stabilizers of the RP-3 code with code parameters $[[9, 1, 3]]$. Data qubits are marked by yellow circles. Red (blue) tiles indicate Pauli X (Z) stabilizers. Each pair of antipodal red tiles (connected by red dotted lines) or blue tiles along the boundary is combined to form a single stabilizer. (b) and (c) Different representations of the logical X and Z operators. (d) and (e) CNOT gate orderings of the syndrome extraction circuits for the RP-3 code and the RP-5 code, respectively. Green circles represent ancilla qubits used during SE. On the boundary, each antipodal pair of ancilla qubits is identified as a single ancilla qubit.

III. MORPHING BETWEEN THE RP-3 (RP-5) CODE AND THE SRP-3 (SRP-5) CODE

To efficiently verify the logical T state during the cultivation process, we require the logical qubit to be hosted on a self-dual CSS code, which hosts a transversal logical H_{XY} operator. However, we also require that at the end of the cultivation process, the logical T state is prepared on the RP code, which can be efficiently grown to a larger rotated surface code. Thus, we need to simultaneously find a self-dual CSS code and an efficient way to morph between the self-dual code and the RP code. Thanks to the fold-duality of the RP code, we can find a compact circuit, which, upon acting on the RP code, guarantees the resulting code is a self-dual CSS code.

In this section, we first list a few standard definitions and notations (Definitions 1–4) in the following, and present a general framework for constructing a self-dual CSS code from a fold-dual one in Sec. III A. We then discuss

the SRP-3 and SRP-5 code and the construction of the morphing circuits in Sec. III B.

Definition 1 (CSS codes). A stabilizer code $\mathcal{C}$ is a Calderbank-Shor-Steane (CSS) code if $\mathcal{C}$ has a stabilizer generating set in which each element is a Pauli X string operator or a Pauli Z string operator. In the following, we denote the number of data qubits of $\mathcal{C}$ by $|\mathcal{C}|$. We also label the data qubits by $1, 2, \dots, |\mathcal{C}|$ and use qubits synonymously with their labels. The set of all data qubits is denoted by $[\mathcal{C}] := \{1, 2, \dots, |\mathcal{C}|\}$.

Definition 2 (Hadamard conjugated Pauli string operators). For a set $\mathcal{Q}$ of qubits, define the transversal H gate on $\mathcal{Q}$ as

$$H_{\mathcal{Q}} := \prod_{i \in \mathcal{Q}} H_i,$$

where H_i is the single-qubit Hadamard gate on qubit i . Denote the conjugation of P by the transversal H gate on the support of P as $h(P) := H_{\text{supp}(P)} P H_{\text{supp}(P)}^\dagger$, where $\text{supp}(P)$ (the support of P) is the set of qubits on each of which P has a nonidentity element.

Definition 3 (Self-dual CSS codes (with even distance)). A CSS code $\mathcal{C}$ is said to be self-dual if for each X (Z) stabilizer S , $h(S)$ is a Z (X) stabilizer. A self-dual CSS code $\mathcal{C}$ is said to have *even distance* if $\mathcal{C}$ encodes at least one logical qubit and any X or Z logical operator of $\mathcal{C}$ has even weight. [This definition is natural in the sense that any X (Z) logical operator L of $\mathcal{C}$ deformed by any X (Z) stabilizer has the same weight parity as L , as a result of the self-duality.]

Definition 4 (Fold-duality (a special case of ZX-dualities defined in Ref. [39])). Consider a CSS code $\mathcal{C}$. A *Folding* operation σ on $\mathcal{C}$ is defined as an order-2 permutation of code qubits. More precisely, σ is defined as a member of the symmetric group $\text{Sym}([\mathcal{C}])$ with $\sigma^2 = 1$. Define the *fold line* of σ as $\text{fix}(\sigma) := \{i = \sigma(i) | i \in [\mathcal{C}]\}$, the set of fixed qubits under σ . The unitary operator $\hat{\sigma}$ that implements the *folding* σ is simply the product of SWAP gates on each pair of qubits that form a size 2 orbit under σ ,

$$\hat{\sigma} := \prod_{\substack{(i, \sigma(i) > i), \\ i \in [\mathcal{C}]}} \text{SWAP}_{i, \sigma(i)}.$$

For a Pauli string operator P , denote its conjugation by $\hat{\sigma}$ as $\hat{\sigma}(P) := \hat{\sigma} P \hat{\sigma}^\dagger$. $\mathcal{C}$ is said to be fold-dual (or to have fold-duality) if there exists a *folding* σ on $\mathcal{C}$ that satisfies the following condition:

- (1) for each X (Z) stabilizer S of $\mathcal{C}$, $h(\hat{\sigma}(S))$ is a Z (X) stabilizer of $\mathcal{C}$.

Conversely, given a fold-dual code $\mathcal{C}$, if a *folding* σ on $\mathcal{C}$ satisfies the above condition, we say σ is an *associated folding* of $\mathcal{C}$. (Note that $\mathcal{C}$ may have more than one associated folding.)

A. Turning fold-dual CSS codes into self-dual ones via concatenation

Inspired by the previous observation that color codes can be constructed from surface codes [47–49] and, in particular, by the code concatenation perspective in Ref. [49], we will first present a more general recipe for transforming a fold-dual CSS code into a self-dual CSS code by concatenating the fold-dual code with a series of (small) even-distance self-dual CSS codes (Definition 3). Then, to illustrate the recipe more concretely, we return to the examples in Ref. [49] where color codes can be constructed by concatenating fold-dual surface codes with a series of $[[6, 4, 2]]$ and/or $[[4, 2, 2]]$ codes.

We will use the following property for even-distance self-dual CSS codes.

Lemma 1 (Logical effect of transversal H gates on an even-distance self-dual CSS code). Consider an even-distance self-dual CSS code $\mathcal{C}$ with code parameters $[[n := |\mathcal{C}|, k, d]]$. Then n , k , and d must be even numbers. Suppose $\mathcal{C}$ has $k = 2m$ logical qubits, then they can be chosen and arranged such that transversally applying H gates on all data qubits of $\mathcal{C}$ implements the following logical unitary:

$$\prod_{i=1}^m (\bar{H}_{2i-1} \otimes \bar{H}_{2i} \circ \overline{\text{SWAP}}_{2i-1, 2i}). \quad (1)$$

Proof. The proof idea is to recursively construct pairs of logical qubits such that the logical effect of the transversal H gate $H_{[\mathcal{C}]}$ on the i th pair (containing qubit $2i - 1$ and qubit $2i$) is

$$\bar{H}_{2i-1} \otimes \bar{H}_{2i} \circ \overline{\text{SWAP}}_{2i-1, 2i}. \quad (2)$$

In this way, we can prove k is even and also the logical effect of $H_{[\mathcal{C}]}$ is given by Eq. (1). Moreover, since the number of independent X stabilizers is the same as the number of independent Z stabilizers, n has the same parity as k and thus is also even.

First, we start with an arbitrarily chosen logical qubit (labeled by 1) with a logical X (Z) operator $\bar{X}_1$ ($\bar{Z}_1$). As $|\bar{X}_1|$ is even, $h(\bar{X}_1)$ commutes with $\bar{X}_1$ and thus is a logical Z operator, inequivalent to $\bar{Z}_1$. We define $\bar{Z}_2$ as $h(\bar{X}_1)$ and similarly $\bar{X}_2$ as $h(\bar{Z}_1)$. We see that $\{\bar{X}_1, \bar{Z}_1, \bar{X}_2, \bar{Z}_2\}$ specifies a pair of logical qubits which satisfies the requirement in Eq. (2).

Now, suppose we have constructed l pairs of logical qubits such that within each pair j (containing the qubit

$2j - 1$ and the qubit $2j$),

$$h(\bar{X}_{2j-1}) = \bar{Z}_{2j} \quad \text{and} \quad h(\bar{Z}_{2j-1}) = \bar{X}_{2j}.$$

If $\mathcal{C}$ encodes only $2l$ logical qubits, then the proof is complete. Otherwise, we can find another logical qubit $2l + 1$ such that its logical operators $\bar{X}_{2l+1}$ and $\bar{Z}_{2l+1}$ commute with $\bar{Z}_i$ and $\bar{X}_i$, respectively, for $\forall i \in \{1, 2, \dots, 2l\}$. Moreover, $h(\bar{X}_{2l+1})$ ($h(\bar{Z}_{2l+1})$) is also a logical operator that commutes with $\bar{X}_i$ ($\bar{Z}_i$) for $\forall i \in \{1, 2, \dots, 2l + 1\}$. In this way, a pair of qubits specified by $\{\bar{X}_{2l+1}, \bar{Z}_{2l+1}, \bar{X}_{2l+2} := h(\bar{Z}_{2l+1}), \bar{Z}_{2l+2} := h(\bar{X}_{2l+1})\}$ is constructed and satisfies the requirement in Eq. (2). ■

Based on Lemma 1, we tacitly assume in the rest of the paper that for any even-distance self-dual CSS code $\mathcal{C}$ with $2m$ logical qubits, its logical qubits are already arranged such that the transversal H gate on $\mathcal{C}$, $H_{[|C|]}$, implements the logical unitary in Eq. (1). Also, for each $i \in \{1, 2, \dots, m\}$, the logical qubits $2i - 1$ and $2i$ are grouped into a logical qubit pair.

Proposition 1 (Transforming a fold-dual CSS code into a self-dual CSS code via concatenation). Given a fold-dual CSS $[[n, k, d]]$ code $\mathcal{C}$ with an associated folding σ , and a series of even weight self-dual CSS codes $\mathcal{C}_a$ for $a = 1, 2, \dots, l$ with code parameters $[[n_a, k_a, d_a]]$ (n_a, k_a , and d_a are all positive even numbers) for each a , then as long as

$$n - |\text{fix}(\sigma)| = \sum_a k_a,$$

we can construct a self-dual CSS code $\tilde{\mathcal{C}}$ by encoding each data qubit pair $\{i, \sigma(i) > i\}$ in $\mathcal{C}$ as a logical qubit pair in $\mathcal{C}_a$ for some $a \in \{1, 2, \dots, l\}$.

Proof. Define an encoding map Enc that maps Pauli string operators on (data qubits of) $\mathcal{C}$ to Pauli string operators on $\tilde{\mathcal{C}}$ such that for each Pauli string operator P on $\mathcal{C}$, $\text{Enc}(P)$ is the corresponding encoded Pauli string operator on $\tilde{\mathcal{C}}$. Denote the generating set for the X stabilizers of $\mathcal{C}$ and $\mathcal{C}_a$ ($a \in \{1, 2, \dots, l\}$) as $\mathcal{S}_X$ and $\mathcal{S}_{X,a}$, respectively. The generating set for the X stabilizers of $\tilde{\mathcal{C}}$ can be chosen as $\tilde{\mathcal{S}}_X = (\sqcup_a \mathcal{S}_{X,a}) \sqcup \text{Enc}(\mathcal{S}_X)$, with $\text{Enc}(\mathcal{S}_X) := \{\text{Enc}(S) | S \in \mathcal{S}_X\}$.

Consider an X stabilizer $\tilde{S} \in \tilde{\mathcal{S}}_X$. If $\tilde{S} \in \sqcup_a \mathcal{S}_{X,a}$, then $h(\tilde{S})$ is a Z stabilizer of $\tilde{\mathcal{C}}$ due to the self-duality of the codes $\mathcal{C}_a$. On the other hand, if $\tilde{S} \in \text{Enc}(\mathcal{S}_X)$, there exists $S \in \mathcal{S}_X$ such that $\tilde{S} = \text{Enc}(S)$. For a qubit $c \in \text{fix}(\sigma)$ on the fold line of σ , $h(\text{Enc}(X_c)) = \text{Enc}(Z_c)$. For a qubit $t \in [|C|] \setminus \text{fix}(\sigma)$ outside the fold line, $h(\text{Enc}(X_t)) =$

$\text{Enc}(Z_{\sigma(t)})$. Thus

$$\begin{aligned} h(\tilde{S}) &= \prod_{c \in \text{supp}(S) \cap \text{fix}(\sigma)} \text{Enc}(Z_c) \cdot \prod_{t \in \text{supp}(S) \setminus \text{fix}(\sigma)} \text{Enc}(Z_{\sigma(t)}) \\ &= \text{Enc}(h(\hat{\sigma}(S))). \end{aligned} \quad (3)$$

Due to the fold-duality of $\mathcal{C}$, $h(\hat{\sigma}(S))$ is a Z stabilizer of $\mathcal{C}$. Thus, $h(\tilde{S})$ is a Z stabilizer of $\tilde{\mathcal{C}}$. Similarly, we can also prove that for each Z stabilizer $\tilde{S}$ of $\tilde{\mathcal{C}}$, $h(\tilde{S})$ is an X stabilizer of $\tilde{\mathcal{C}}$. Hence $\tilde{\mathcal{C}}$ is self-dual. ■

From Proposition 1, we see that the constructed self-dual code $\tilde{\mathcal{C}}$ is composed of data qubits from $\mathcal{C}$ that remain invariant under folding σ and data qubits from the small self-dual codes. The (small) even-distance self-dual CSS codes used for concatenation in Ref. [49] are the $[[4, 2, 2]]$ code and the $[[6, 4, 2]]$ code [Figs. 6(a) and 6(b)]. A $(6, 6, 6)$ -color code [Fig. 6(c)] with distance 7 can be constructed from a plane-shaped fold-dual surface code [Fig. 6(d)] with distance 4 by concatenating the surface code with three copies of the $[[6, 4, 2]]$ code and three copies of the $[[4, 2, 2]]$ code. Each green face of the color

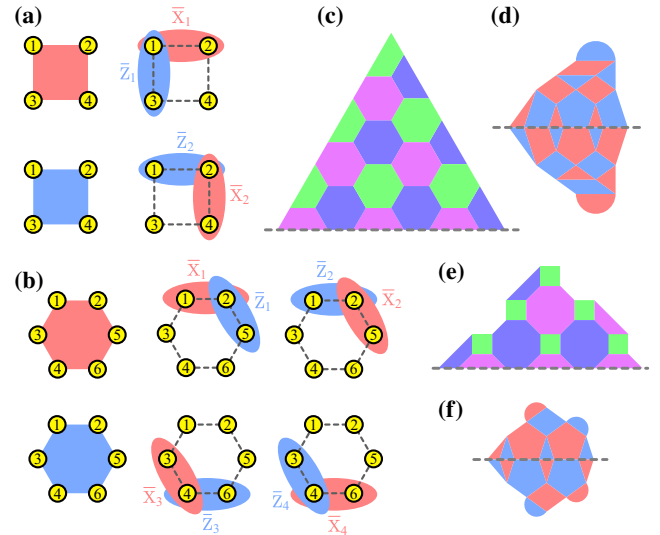


FIG. 6. Small even-distance self-dual CSS codes and the construction of color codes from surface codes. (a) Stabilizers of the $[[4, 2, 2]]$ code and its logical operators. Pauli X (Z) operators are colored red (blue). (b) Stabilizers of the $[[6, 4, 2]]$ code and its logical operators. (c) and (d) A distance-7 $(6, 6, 6)$ -color code [in (c)] can be constructed from a plane-shaped distance-4 surface code [in (d)]. (e) and (f) A distance-7 $(4, 8, 8)$ -color code [in (e)] can be constructed from a frog-shaped distance-4 surface code [in (f)]. Faces of color codes are colored in lavender, magenta, and green. The X and Z stabilizers on each green face correspond to the stabilizers of either a $[[4, 2, 2]]$ code in (a) or a $[[6, 4, 2]]$ code in (b). Dashed gray lines in (c) and (d) [or in (e) and (f)] depict the fold line of an associated folding of the plane-shaped (or frog-shaped) surface code.

code in Fig. 6(c) represents either a $[[6, 4, 2]]$ code or a $[[4, 2, 2]]$ code. Similarly, as shown in Figs. 6(e) and 6(f), a $(4, 8, 8)$ -color code with distance 7 can be constructed from a frog-shaped fold-dual surface code with distance 4 by concatenating it with six copies of the $[[4, 2, 2]]$ code, also marked by green faces. More generally, both $(6, 6, 6)$ - and $(4, 8, 8)$ -color codes with distance $2t - 1$ can be constructed similarly using surface codes with distance t . (Notice that any 2D color code that encodes only one logical qubit must have an odd distance due to Lemma 1.) We note that other works also use the technique of concatenation on top of the $[[4, 2, 2]]$ or $[[6, 4, 2]]$ codes [51–54]. Our work adds to these results by applying this technique in growing magic states.

B. Construction of the SRP codes and the morphing circuits

In this subsection, we will define the SRP-3 (SRP-5) code operationally by specifying how data qubits from an RP-3 (RP-5) code are encoded into copies of the $[[4, 2, 2]]$ ($[[6, 4, 2]]$) code. We start by describing the encoding and decoding circuits for the $[[4, 2, 2]]$ and $[[6, 4, 2]]$ codes. Morphing between RP codes and SRP codes can be implemented by running these encoding or decoding circuits according to our operational definition of the SRP codes.

As shown in Figs. 7(a) and 7(b), the encoding circuits for the $[[6, 4, 2]]$ and the $[[4, 2, 2]]$ code, respectively, are constructed to let each physical qubit labeled by i ($i \in \{1, 2, 3, 4\}$ for the $[[6, 4, 2]]$ code and $i \in \{1, 2\}$ for the $[[4, 2, 2]]$ code) morph into the logical qubit i in Fig. 6. The encoding circuits work by initializing a Bell pair on two ancilla qubits and growing its stabilizers to the stabilizers of either the $[[6, 4, 2]]$ code or the $[[4, 2, 2]]$ code. In particular, the structure of the encoding circuit for the $[[6, 4, 2]]$ code ensures errors within the qubit block $\{1, 2, 5\}$ or $\{3, 4, 6\}$ will only propagate within the block. (We note that the encoding circuit for the $[[6, 4, 2]]$ code used here is very similar to the one in Ref. [53].) In our work, we will morph an RP code into its corresponding SRP code and then back after performing logical verification on the SRP code. The backward morphing is implemented by the decoding circuits, which are encoding circuits run in reverse with resets replaced by measurements (Fig. 7). Since logical verification is supposed to leave the stabilizers of the SRP code invariant, the ancilla qubits measured at the end of the decoding circuits also serve as flag qubits to detect errors during the encoding, logical verification, and decoding stages.

Notice that each RP code is fold-dual and has an *associated folding* whose *fold line* is the set of qubits on the horizontal line through the center of the RP code patch (Fig. 8). Based on this observation, we define the SRP-3 code and the SRP-5 code operationally in Fig. 8. The data qubits of the RP-3 code and extra ancilla qubits are

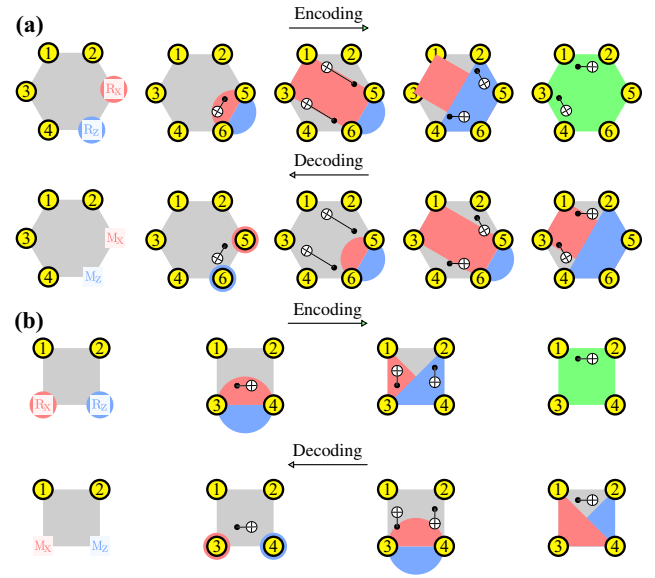


FIG. 7. Building blocks that enable fast morphing between an RP-3 (RP-5) code and an SRP-3 (SRP-5) code. (a) Encoding and decoding circuits for the $[[6, 4, 2]]$ code. (b) Encoding and decoding circuits for the $[[4, 2, 2]]$ code. Assuming the decoding circuit is run right after the encoding circuit, we track the detecting regions [50] generated by reset locations throughout the encoding and decoding circuit in (a) and (b). Each green face indicates an X stabilizer and a Z stabilizer.

arranged on gray faces at the moment immediately before the encoding step [Fig. 8(a)]. The SRP-3 code is defined as the resulting code after applying the encoding circuit on each gray face according to Fig. 7(b). The SRP-5 code is defined similarly in Fig. 8(b).

IV. LOGICAL VERIFICATION ON THE SRP-3 CODE AND THE SRP-5 CODE

Logical T state verification (also called logical double-checking in Ref. [29]) aims to detect logical errors on the logical T state, and is implemented by first measuring the logical H_{XY} operator and then resetting it. For logical H_{XY} measurement, a unitary circuit is designed to transform the logical H_{XY} operator into a Pauli X operator on a single-qubit. This qubit is then measured in the X basis to (effectively) measure the logical H_{XY} operator. The measured qubit is then reset (to $|+\rangle$) in the X basis, and the unitary circuit is run in reverse to (effectively) reset the logical H_{XY} operator. In the ideal setting with no circuit noise, the input logical T state is the $+1$ eigenstate of the logical H_{XY} operator; therefore, the X basis measurement result should also be $+1$. Then, after resetting the logical H_{XY} operator, we would get back the exact logical T state. We would also like to illustrate that resetting the logical H_{XY} is vital to counteract the logical H_{XY} measurement error by using the following example. Suppose that the input logical T state suffers a logical Z error prior to the

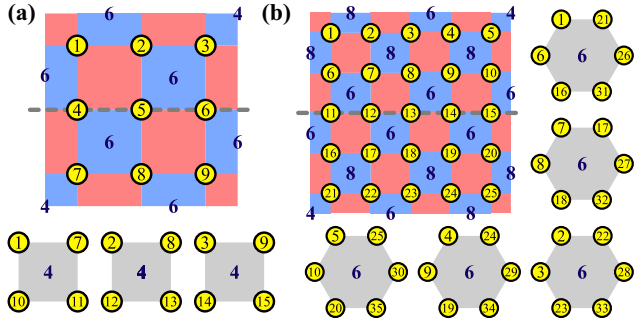


FIG. 8. Operational definition of the SRP-3 and SRP-5 code. Each RP code has an *associated folding* whose *fold line* is illustrated by a dashed gray line here. Yellow circles labeled by the same number represent the same qubit. (a) The $[[15, 1, 3]]$ SRP-3 code is obtained by encoding the data qubits of the RP-3 code that are not on the *fold line* into logical qubits of the $[[4, 2, 2]]$ code. Here, qubits 1–9 are the data qubits of the RP-3 code. Each gray square consists of two data qubits of the RP-3 code and two ancilla qubits. By executing the encoding circuit of the $[[4, 2, 2]]$ code in Fig. 7(b) on each gray square, we can morph the RP-3 code to the SRP-3 code. Each gray square will become a $[[4, 2, 2]]$ -code patch after the execution of the encoding circuits. (b) The $[[35, 1, 5]]$ SRP-5 code is obtained by encoding the data qubits of the RP-5 code that are not on the *fold line* into logical qubits of the $[[6, 4, 2]]$ code. Here, qubits 1–25 are the data qubits of the RP-5 code. Each gray hexagon consists of four data qubits of the RP-5 code and two ancilla qubits. By executing the encoding circuit of the $[[6, 4, 2]]$ code in Fig. 7(a) on each gray hexagon, we can morph the RP-5 code to the SRP-5 code. Each gray hexagon will become a $[[6, 4, 2]]$ -code patch after the execution of the encoding circuits. The weight of each Z stabilizer of the SRP-3 (SRP-5) code is presented in the following way. For each Z stabilizer of the RP-3 (RP-5) code, the weight of the corresponding encoded Z stabilizer of the SRP-3 (SRP-5) code is illustrated at the center of the original (unencoded) Z stabilizer. The weight of the Z stabilizer of each $[[4, 2, 2]]$ and $[[6, 4, 2]]$ code is illustrated at the center of each gray square and hexagon, respectively.

logical verification; now the input state is the -1 eigenstate of the logical H_{XY} operator, and should be detected by the logical H_{XY} measurement. However, if the logical H_{XY} measurement is also faulty by flipping the measurement outcome, the erroneous input state would pass the verification without triggering any detector. Fortunately, due to the logical reset, the output logical state after the verification step is still the correct logical T state. Thus, without the logical reset step (in this case, the unitary circuit is run in reverse immediately after performing the single-qubit measurement without resetting the qubit to $|+\rangle$), logical verification would only increase the fault distance of the T state preparation circuit by at most 1 due to the presence of logical H_{XY} measurement error. (Here, the fault distance is the number of circuit errors that result in a logical error in the output logical T state without triggering any detector.) In contrast, with the logical reset step, a single round

of logical verification can increase the fault distance by at most 2. In addition to the circuit used above to perform the logical H_{XY} measurement and reset, we also need to insert flag qubits to detect errors in the circuit [29]. Therefore, the logical verification circuit consists of two components: the logical processing component for implementing the logical verification described above and the flag component for detecting errors during the former component. We describe how to perform logical verification on the SRP-3 code and the SRP-5 code, respectively, in the following.

For the SRP-3 code, transversal $\tilde{H}_{XY}$ ($\tilde{H}_{XY} := \frac{1}{\sqrt{2}}(X - Y)$) on all data qubits implements the logical H_{XY} operator. On the other hand, for the SRP-5 code, the logical H_{XY} operator is implemented by transversal H_{XY} on all data qubits. Also, to ensure that such transversal implementation only affects the logical qubit and does not flip the sign of any stabilizer, we require the SRP-3 (SRP-5) code prepared in the $+1$ eigenspace of Z stabilizers with doubly even weight and the -1 eigenspace of Z stabilizers with singly even weight [29]. We specify how to achieve this with feed-forward Pauli X correction in Sec. V. We now move on to describe the logical verification circuits. For the SRP-3 code (Fig. 9), the logical processing component is composed of steps 0–1, 4–15, and 18. To measure the $\tilde{H}_{XY}^{\otimes 15}$ operator, we first transform it to $X^{\otimes 15}$ by applying a layer of T gates on all qubits. A series of CNOT gates are then applied to contract $X^{\otimes 15}$ to a single Pauli X on qubit 5, which is then measured in the X basis to read out the logical measurement result. The measurement circuit is then run in reverse—by first resetting qubit 5 in $|+\rangle$ and then running the unitary circuit in the previous steps in reverse—to restore the code stabilizers and the logical qubit. As for the flag component, at steps 2–3, we insert flag qubits, which are then decoupled from the rest of the circuit at step 16 and measured at step 17. Note that the flag component only checks for Z errors and leaves X errors unchecked. The latter may trigger detectors in the following SE rounds, but do not induce logical errors by themselves. This can be readily checked by running an ideal logical verification circuit and then measuring all stabilizers noiselessly. We note that there is a lot of freedom in designing logical verification circuits. For our design, we prioritize conceptual simplicity and explicit parallelizability (at each step shown in Fig. 9, we have the same operation on each bottom square). We leave a more compact design for future work. The logical verification circuit for the SRP-5 code is designed on the same principle and is shown in Fig. 10.

V. INJECTION AND GROWTH

In this section, we describe three types of circuits for patch growth: an injection circuit (for injecting a single-qubit state into an RP-3 code), an RP-growth circuit (for growing an RP-3 code to an RP-5 code), and expansion

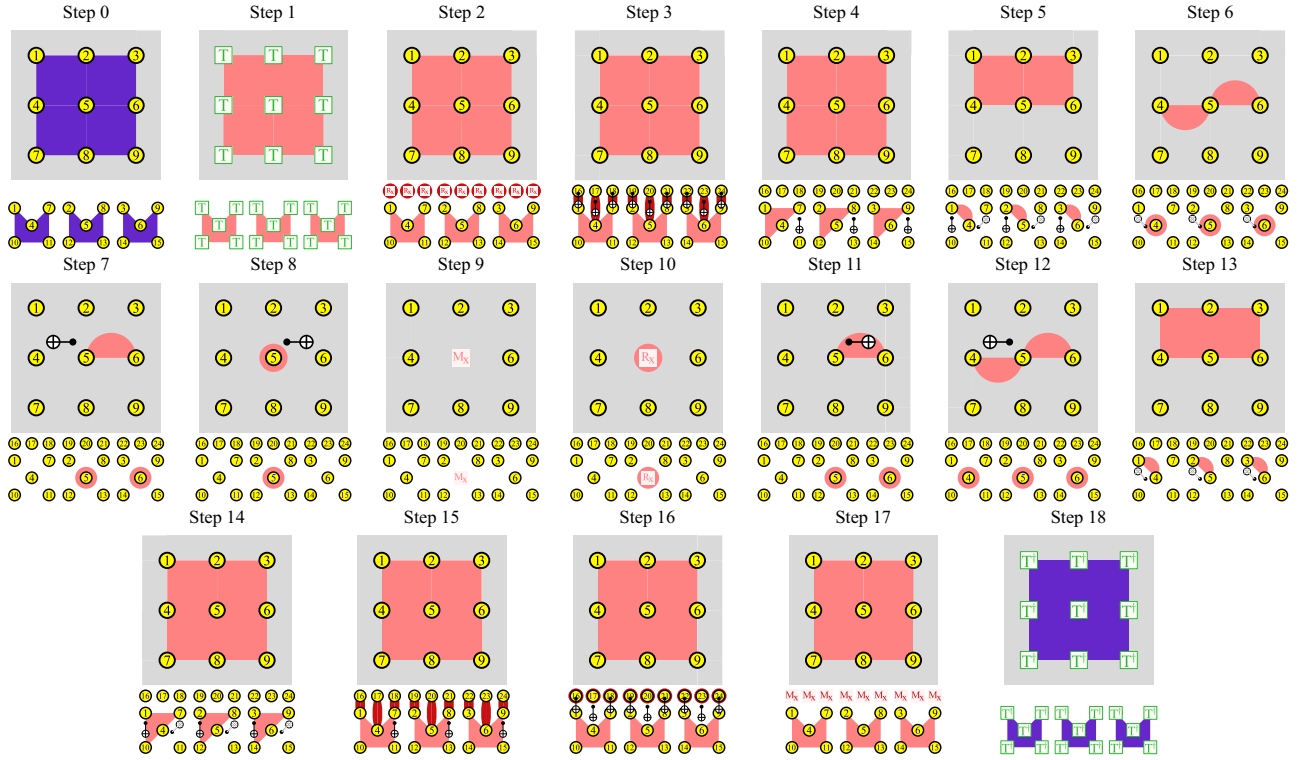


FIG. 9. Logical verification circuit for the SRP-3 code. The layout of the data qubits of the SRP-3 code at step 0 is similar to Fig. 8(a). At the center of each $[[4, 2, 2]]$ -code patch, an additional circle representing a qubit on the *fold line* is added. Since for many qubits, each is represented by more than one yellow circle, we set the following rules for drawing the operators and physical operations to illustrate explicitly the modularity of the logical verification circuit. First, an operator support or a single-qubit operation on each qubit is illustrated on each circle representing the qubit. Secondly, a two-qubit operation on each pair of qubits is illustrated on a single pair of circles representing the qubits. The illustration of two-qubit operations highlights the modularity of the logical verification circuit. At step 0, purple regions illustrate the operator $\tilde{H}_{XY}^{\otimes 15}$ that we need to measure. At step 1, $\tilde{H}_{XY}^{\otimes 15}$ is transformed to $X^{\otimes 15}$ whose evolution is illustrated by (light) red regions and tracked until step 17. Detector regions generated by the flag qubits (initialized at step 2) are colored in dark red and are only shown in steps 2, 3, 15, and 16. The purple regions at step 18 indicate that the state here should be a $+1$ eigenstate of $\tilde{H}_{XY}^{\otimes 15}$ when the circuit is noiseless.

circuits (for expanding an RP code to a larger rotated surface code). All our growth circuits grow a smaller code at the center outward and make use of Bell pairs. The injection circuit is implemented by a single round of SE on top of a single-qubit $|T\rangle$ at the center and four surrounding Bell pairs [Fig. 11(a)]. We fix the signs of the Z stabilizers of the RP-3 code by applying Pauli X corrections based on the Z stabilizer measurement results [Fig. 11(a)]. Similar to T state injection, growing the RP-3 code to an RP-5 code and fixing its Z stabilizer signs can be implemented according to Fig. 11(b). Expanding an RP code to a rotated surface code requires preparing a layer of Bell pairs surrounding the RP code and also initializing four blocks of $|0\rangle$ or $|+\rangle$ states [Figs. 11(c) and 11(d)]. (See Appendix B for discussion on the purpose of introducing Bell pairs and their preparation.) In our MSC-3 protocol, the expansion circuit (with a final distance of 11) is run concurrently with the SRP-3-to-RP-3 morphing circuit (Appendix A). Similarly, in our MSC-5 protocol, the RP-growth circuit is run concurrently with the SRP-3-to-RP-3 morphing circuit, and

the expansion circuit for the RP-5 code is run concurrently with the SRP-5-to-RP-5 morphing circuit.

VI. FINAL-STAGE POSTSELECTION BASED ON SOFT OUTPUTS

During the expansion stage, as well as its first few following SE rounds, the logical qubit is not yet fully protected at the larger surface code distance and is still more susceptible to errors. This necessitates another round of postselection at the end of the MSC protocol. Prior works [29,38] on magic state cultivation use the complementary gap method [55] to estimate, for each shot, the decoding confidence, which is used to determine whether this shot should be discarded. In this work, an alternative method, named the soft-output method, developed in Ref. [44] is used to obtain a measure of decoding confidence. In particular, following [44,56], we customized the PyMatching package [57] to compute the soft outputs from internal variables created during the decoding process for

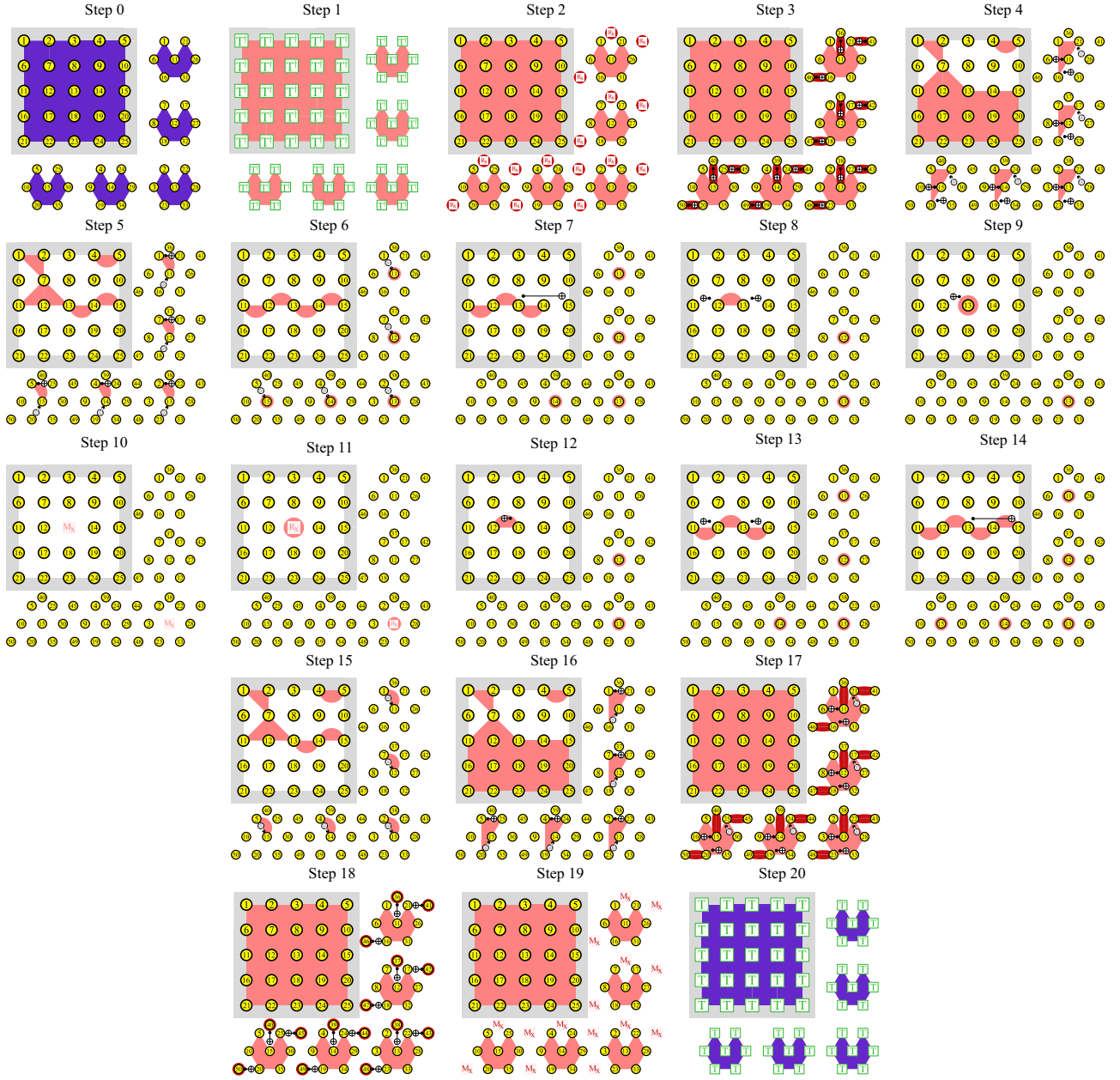


FIG. 10. Logical verification circuit for the SRP-5 code. The layout of the data qubits of the SRP-5 code at step 0 is similar to Fig. 8(b). At the center of each $[[6, 4, 2]]$ -code patch, an additional circle representing a qubit on the *fold line* is added. Since for many qubits, each is represented by more than one yellow circle, we follow the rules in Fig. 9 for drawing operators and physical operations to illustrate explicitly the modularity of the logical verification circuit. At step 0, purple regions represent the operator $H_{XY}^{\otimes 35}$ that we need to measure. At step 1, $H_{XY}^{\otimes 35}$ is transformed to $X^{\otimes 35}$ whose evolution is illustrated by (light) red regions and tracked until step 19. Detector regions generated by the flag qubits (initialized at step 2) are colored in dark red and are only shown in steps 2, 3, 17, and 18. The purple regions at step 20 indicate that the state here should be a $+1$ eigenstate of $H^{\otimes 35}$ if the circuit is noiseless. One key difference between the circuit here and the logical verification circuit for the SRP-3 code is that, here, a layer of $T^\dagger$ gates [instead of a layer of T gates for the SRP-3 code (Fig. 9)] is applied at step 1. The reason for such a difference is that the operator to be measured here is $H_{XY}^{\otimes 35}$ (instead of $\tilde{H}_{XY}^{\otimes 15}$ for the SRP-3 code).

each shot. The soft-output method is more versatile and suitable for our case, which involves the boundaryless RP code, as this method does not require any specific boundary

condition of the surface code to work, while the complementary gap method requires the open boundary condition possessed by the rotated or unrotated surface code.

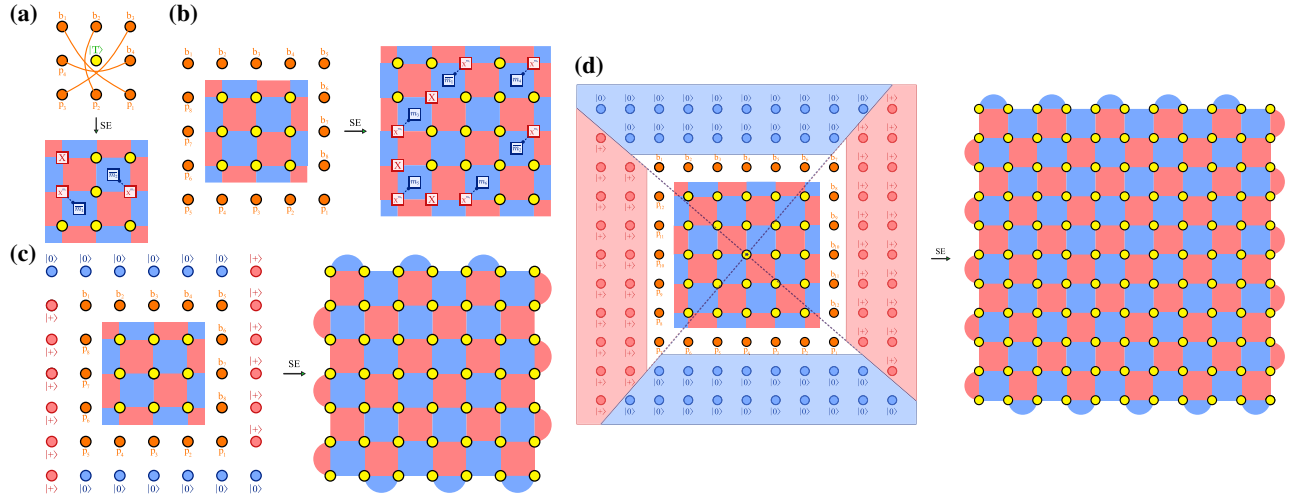


FIG. 11. Circuits for code growth. Each qubit pair $\{b_i, p_i\}$ ($i \in \mathbb{N}$) is a Bell pair. For each $a \in \mathbb{N}$, m_a or $\overline{m}_a$, at the center of a stabilizer, denotes the measurement result of this stabilizer (here $\overline{m}_a$ records the inverse of the actual measurement result). (a) Injection circuit. Qubits from the same Bell pair are connected by an orange curve. After the syndrome extraction round, a single-qubit T state is injected into an RP-3 code. (b) Growing an RP-3 code to an RP-5 code. In both (a) and (b), Pauli X corrections are applied at the end of the syndrome extraction round to fix the signs of the Z stabilizers. A Pauli X operation is linked by a dashed blue line to a stabilizer measurement result if it controls this Pauli X operation. (c) Expanding an RP-3 code to a distance-7 rotated surface code. Note that in our MSC-3 protocol, we expand to a distance-11 rotated surface code. Here, for simplicity, we illustrate the expansion to a distance-7 rotated surface code. (d) Expanding an RP-5 code to a distance-11 rotated surface code. Four blocks of initialized $|0\rangle$ states and $|+\rangle$ states are illustrated by light blue and light red regions, respectively.

Following [44,56], we first briefly review how the soft output [44] for decoding a rotated surface code memory can be obtained from the sparse blossom [57] decoding process used in PyMatching. As we can decode on the X detectors and on the Z detectors separately, we restrict ourselves to decoding on the X detectors for now. Consider a decoding graph $\mathcal{G}_X(\mathcal{V}_X, \mathcal{E}, w)$ ($\mathcal{G}_X$ for short) with the vertex set $\mathcal{V}_X$ (the set of all X detectors) and the edge set $\mathcal{E}$, in which each edge e may either be a normal edge connecting two vertices in $\mathcal{V}_X$ or a boundary edge connected to only one vertex in $\mathcal{V}_X$ and is assigned a non-negative weight $w(e) \in \mathbb{R}_{\geq 0}$. The distance $D(u, v)$ between two vertices $u, v \in \mathcal{V}_X$ is defined as the length of the shortest path in $\mathcal{G}_X$ connecting them. A *combined graph fill region* (or *region* for short) $\mathcal{R}$ is defined as a collection of vertices $V(\mathcal{R}) \subset \mathcal{V}_X$ equipped with a nonempty subset (named the source subset), $S(\mathcal{R}) \subset V(\mathcal{R})$, and a local radius map $r_{\mathcal{R}} : V(\mathcal{R}) \rightarrow \mathbb{R}_{\geq 0}$, such that,

- (1) a vertex $u \in V(\mathcal{R})$ iff there exists $s \in S(\mathcal{R})$ such that $D(u, s) \leq r_{\mathcal{R}}(s)$, and,
- (2) for $\forall v \in V(\mathcal{R})$,

$$r_{\mathcal{R}}(v) = \min_{s \in S(\mathcal{R})} (r_{\mathcal{R}}(s) - D(v, s)).$$

Heuristically, the range of $\mathcal{R}$ is characterized by its local radius map. A vertex $v \in V(\mathcal{R})$ is said to be contained in $\mathcal{R}$ with local radius $r_{\mathcal{R}}(v)$ and, in particular, is said to be a source of $\mathcal{R}$ iff $v \in S(\mathcal{R})$. (Actually, in Ref. [57], only

graph fill regions are defined. The *combined graph fill regions* defined here for our own convenience are combinations of *graph fill regions*. More precisely, a *combined graph fill region* is the combination of an *active graph fill region* [57] with all its descendants.) Consider a single decoding shot where we are given a set of triggered detectors $\mathcal{D} \subset \mathcal{V}_X$. The sparse blossom algorithm [57] starts by initializing, for each $u \in \mathcal{D}$, a *region* that only contains u with local radius 0. Throughout the algorithm, *regions* (their vertices, sources, and local radius maps) are varied, and they may merge into larger *regions* or shatter into smaller ones. Despite the evolution of *regions*, they remain mutually disjoint, and their source subsets only contain vertices in $\mathcal{D}$. Moreover, at any moment, any *region* $\mathcal{R}$ contains an odd number of sources ($|S(\mathcal{R})| \equiv 1 \pmod{2}$). At the end of the decoding procedure, all regions are matched either to each other or to the boundary and will no longer be varied. Heuristically, one may assign each detection event a +1 charge (here charges are measured modulo 2) and think of each region as a charged region containing an odd number of detection events, so that at the end of the decoding process, charged regions are “neutralized” by either matching two regions together or matching a region to the boundary, which absorbs charges. Now, given the existing *regions* at the end of the above decoding process, we can define for each edge e a postdecoding weight $\tilde{w}(e)$ by erasing the part of its original weight covered by them. More precisely, for each endpoint $v \in \partial e$ (∂e denotes the set of endpoints of e), define its local

radius $r(v)$ as $r_{\mathcal{R}}(v)$ if v is contained in a region $\mathcal{R}$ and 0 otherwise. Then $\tilde{w}(e)$ is defined as $\tilde{w}(e) = \max(0, w(e) - \sum_{v \in \partial e} r(v))$. [See Fig. 12(a) for an example where both endpoints of e are contained in two different *regions*.] Based on $\mathcal{G}_X$, we define a graph $\tilde{\mathcal{G}}_X(\mathcal{V}_X \sqcup \{v_{bd}\}, \tilde{\mathcal{E}}, \tilde{w})$ (or $\tilde{\mathcal{G}}_X$ for short) with an additional virtual boundary vertex v_{bd} and $\tilde{\mathcal{E}}$ is obtained from $\mathcal{E}$ by attaching each boundary edge in $\mathcal{E}$ to v_{bd} [Fig. 12(b)]. Denote the set of loops in $\tilde{\mathcal{G}}_X$ that also flip a logical X observable as $L(\tilde{\mathcal{G}}_X)$. The *soft-output* ϕ_X for the decoding shot above is defined as the minimum path length in $L(\tilde{\mathcal{G}}_X)$ [44],

$$\phi_X := \min_{\ell \in L(\tilde{\mathcal{G}}_X)} \left(\sum_{e \in \ell} \tilde{w}(e) \right).$$

To evaluate ϕ_X , v_{bd} is split into two distinct vertices $v_{bd,1}$ and $v_{bd,2}$ at the top and bottom boundary, respectively [Fig. 12(b)]. As a result, $\forall \ell \in L(\tilde{\mathcal{G}})$ is now simply a path connecting $v_{bd,1}$ to $v_{bd,2}$ [Fig. 12(b)]. The *soft-output* ϕ_X , now as the distance between $v_{bd,1}$ and $v_{bd,2}$, can be efficiently computed via Dijkstra's algorithm [44]. For decoding on the Z detectors, the soft-output ϕ_Z can be defined similarly. For the complete decoding problem on both X and Z detectors, the *soft-output* ϕ is defined as $\min(\phi_X, \phi_Z)$. Suppose that errors in a shot are very sparse and do not cluster or form long error chains; there would also be very few *regions* after the decoding process, and they would also have small radii. In this case, as the postdecoding edge weight would still be close to the original edge weight for almost all edges, the soft-output ϕ should still have a large value $\sim O(d)$ where d is the code distance. In contrast, proliferation of errors may result in larger *regions* and thus a smaller ϕ . Therefore, we expect that higher values of ϕ are correlated with stronger decoding confidence. As shown in Ref. [44], discarding all shots with soft outputs below a certain cutoff can significantly suppress the logical error rate and, at the same time, maintain a low discard rate.

We now describe how a *two-dimensional soft output* is extracted for decoding each shot of our MSC protocol. As mentioned in Sec. I, we perform full postselection on the cultivation process and only decode the ES process. For our MSC-3 (MSC-5) protocol, the ES process consists of the expansion circuit, which expands an RP-3 (RP-5) code to a distance-11 rotated surface code, and a few rounds of SE on the latter. In the following, for simplicity, we illustrate how to extract *two-dimensional soft outputs* from decoding an ES process that expands an RP-3 code to a distance-7 rotated surface code. (Soft-output extraction for our MSC protocol can be performed similarly.) Consider decoding on X detectors first and denote the decoding graph as $\mathcal{G}_{\text{MSC},X}$. Similar to constructing $\tilde{\mathcal{G}}_X$ from $\mathcal{G}_X$ above, for each decoding shot, we construct

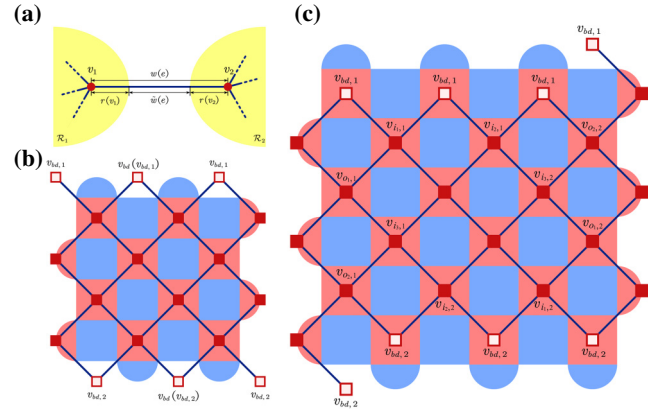


FIG. 12. Obtaining soft outputs. (a) Obtaining postdecoding edge weights based on *regions* at the end of a decoding process. The edge e between the vertex v_1 and v_2 with original weight $w(e)$ is reweighted to $\tilde{w}(e)$. *Regions* $\mathcal{R}_1$ and $\mathcal{R}_2$ are colored yellow. (b) Slice of $\tilde{\mathcal{G}}_X$ on the vertices (X detectors) generated at a single syndrome extraction round. Stabilizers of the rotated surface code are plotted simply to indicate positions of the X detectors. X detectors are marked by smaller dark red squares, and error mechanisms (edges of $\tilde{\mathcal{G}}_X$) by blue lines. A virtual boundary vertex v_{bd} added to $\tilde{\mathcal{G}}_X$ is connected to boundary edges and is then split into two distinct vertices $v_{bd,1}$ and $v_{bd,2}$ (hollow squares) at the upper and lower boundaries, respectively [44]. (c) Slice of $\tilde{\mathcal{G}}_{\text{MSC},X}$ on the vertices (X detectors) generated immediately after expanding an RP-3 code to a distance-7 rotated surface code. Similar to (b), there are two distinct virtual boundary vertices $v_{bd,1}$ and $v_{bd,2}$ (hollow squares) at the upper and lower boundaries, respectively. Each antipodal pair of vertices, such as $\{v_{i,a,0}, v_{i,a,1}\}$ for $a \in \{1, 2, 3\}$ and $\{v_{o,b,0}, v_{o,b,1}\}$ for $b \in \{1, 2\}$, originally representing the same vertex (detector), is split into two distinct vertices. Inner antipodal pairs $\{v_{i,a,0}, v_{i,a,1}\}_{a \in \{1,2,3\}}$ correspond to corner and boundary X stabilizers of the RP-3 code (Fig. 5).

$\tilde{\mathcal{G}}_{\text{MSC},X}$ from $\mathcal{G}_{\text{MSC},X}$ by replacing edge weights with postdecoding edge weights and connecting boundary edges to a virtual boundary vertex v_{bd} , which is then split into two boundary vertices $v_{bd,1}$ and $v_{bd,2}$ at the top and bottom boundaries, respectively. Right after the expansion step, the X detectors and the slice of $\tilde{\mathcal{G}}_{\text{MSC},X}$ on them are illustrated in Fig. 12(c). Interestingly, as a result of our expansion circuit, each antipodal pair $\{v_{i,a,0}, v_{i,a,1}\}$ ($a \in \{1, 2, 3\}$) or $\{v_{o,b,0}, v_{o,b,1}\}$ ($b \in \{1, 2\}$) of vertices in Fig. 12(c) actually represents a single vertex (or equivalently a single X detector). Here, we also treat each of these antipodal pairs of vertices as distinct vertices. As a result, each path from $v_{i,a,0}$ to $v_{i,a,1}$ for $\forall a \in \{1, 2, 3\}$ corresponds to a logical Z error. Similar to obtaining ϕ_X above, we calculate, via Dijkstra's algorithm, the minimum distance $\phi_{X,a}$ between $v_{i,a,0}$ and $v_{i,a,1}$ for $\forall a \in \{1, 2, 3\}$ and also the minimum distance $\phi_{X,bd}$ between $v_{bd,0}$ and $v_{bd,1}$. We then define $\phi_{X,\mathbb{RP}^2} := \min_{a \in \{1,2,3\}} \phi_{X,a}$. Similarly, for decoding on the Z detectors, we can obtain $\phi_{Z,\mathbb{RP}^2}$ and $\phi_{Z,bd}$. Finally, the

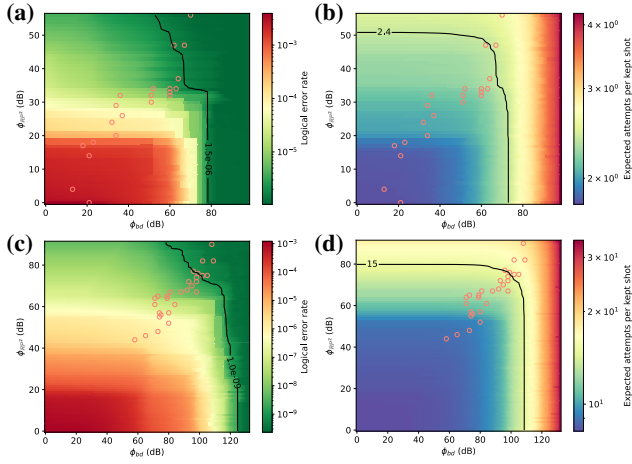


FIG. 13. (a) and (b) Logical error rate and expected attempts per kept shot for our MSC-3 protocol at each soft-output cutoff. (c) and (d) Logical error rate and expected attempts per kept shot for our MSC-5 protocol at each soft-output cutoff. Cutoffs used in Fig. 4(b) are marked by red circles.

two-dimensional soft output is defined as

$$\begin{aligned} \phi_{\text{MSC}} &:= (\phi_{\mathbb{RP}^2}, \phi_{bd}) \\ &:= (\min\{\phi_{X,\mathbb{RP}^2}, \phi_{Z,\mathbb{RP}^2}\}, \min\{\phi_{X,bd}, \phi_{Z,bd}\}). \end{aligned} \quad (4)$$

In our Monte Carlo simulations, ϕ_{MSC} for each shot is measured in dB and is rounded to a tuple of integers. For a given soft-output cutoff $(\phi_1, \phi_2) \in \mathbb{Z}_{\geq 0}^2$, we perform final-stage postselection by only retaining shots whose soft outputs $(\phi_{\mathbb{RP}^2}, \phi_{bd})$ satisfy $\phi_{\mathbb{RP}^2} \geq \phi_1$ and $\phi_{bd} \geq \phi_2$. The resulting logical error rate and the expected number of attempts (per kept shot) under each choice of cutoff (ϕ_1, ϕ_2) are shown in Figs. 13(a) and 13(b) for our MSC-3 protocol and in Figs. 13(c) and 13(d) for our MSC-5 protocol. Around a certain discard rate (or a certain expected number of attempts per kept shot), we can select a cutoff with the lowest logical error rate. Such a cutoff is said to achieve almost optimal local trade-offs (between discard rate and logical error rate). Data points used in Fig. 4 correspond to cutoffs with almost optimal local trade-offs. In particular, the selected cutoffs used in Fig. 4(b) are illustrated by red circles in Fig. 13.

VII. DISCUSSION AND OUTLOOK

Physical platforms with long-range connectivity, such as neutral atoms and ion traps, allow implementations of transversal logical Clifford gates on the rotated surface code. Therefore, to keep up with fast transversal logical Clifford gates, logical magic state production with high efficiency is desired. To this end, we designed a MSC protocol that utilizes nonlocal connectivity to produce high-fidelity logical T states on the (regular) rotated surface code with reduced space-time cost. According to Monte

Carlo sampling on the Clifford variant of our protocol, we observe that it achieves nearly an order of magnitude space-time cost reduction at a logical error rate around 1.5×10^{-6} (for our MSC-3 version) or 1×10^{-9} (for our MSC-5 version) compared to the original MSC protocol [29]. Our protocol features both functionality-oriented design, in which structured codes and circuits are tailored to perform a certain logical task, and codesigning efficient fault-tolerant components with compact interfaces. The benefit of this design philosophy is that high efficiency is maintained throughout our protocol. More specifically, we simultaneously have a cultivation process with efficiency comparable to that of the original MSC protocol, and a much more efficient ES process compared to the “grafting” process in Ref. [29]. Notice that the “grafting” process is used to bypass the color code decoding problem. Therefore, it would be interesting to improve current color code decoders and then compare the efficiency of the original MSC protocol (with “grafting” replaced by color code growth) with ours and, more generally, compare the color code scheme for FTQC with the surface code one. Moreover, we expect that our cultivation process still has room for further improvements. For instance, the logical verification circuits in this work are designed to be regular and conceptually simple. Future work is needed to make logical verification more compact by improving our circuits or finding more efficient ways to implement logical verification. Injecting the T state and growing the RP code using unitary circuits may help to reduce the number of SE rounds in the cultivation process, thereby improving the postselection survival rate. At physical noise rate $p = 0.001$, the fidelity of logical T states produced in our protocol still falls slightly short for large-scale quantum algorithms such as factoring 2048-bit RSA integers [58]. In such cases, another round of magic state distillation (MSD) on the cultivated T states is needed to produce logical CCZ or T states at a sufficiently low error rate [14, 19]. Thus, it would be important to estimate the resource cost for running MSD circuits [59] on cultivated T states and to design high-yield, high-efficiency magic state factories for large-scale quantum algorithms.

ACKNOWLEDGMENTS

We thank Nadine Meister for discussions on the *soft-output* method in Ref. [44]. This work was supported by the HFNL Self-Deployed Project (Nos. ZB2024010101, ZB2024010102, ZB2024010201, and ZB2024010501), the Quantum Science and Technology-National Science and Technology Major Project (No. 2021ZD0301405), the National Key R&D Program of China (No. 2022ZD0160101), the Shanghai Municipal Science and Technology Commission (No. 24DP2600300), and the National Natural Science Foundation of China (No. 12322415).

DATA AVAILABILITY

The data that support the findings of this article are openly available [60].

APPENDIX A: SIMULATION DETAILS

In this appendix, we describe details of our simulation, including

- (1) the circuit noise model,
- (2) how certain circuits are run concurrently, and
- (3) how space-time cost is computed.

We use the uniform depolarizing circuit noise model as in Ref. [29]. More specifically, our circuit error configurations are listed in the following.

- (1) Each single-qubit gate (including idling) is followed by a single-qubit depolarization channel with noise strength p ,
- (2) each two-qubit gate is followed by a two-qubit depolarization channel with noise strength p ,
- (3) initialization in $\pm Z$ bases is followed by a single-qubit X error with probability p ,
- (4) initialization in $\pm X$ bases is followed by a single-qubit Z error with probability p , and
- (5) each single-qubit measurement result is flipped with probability p .

As mentioned in Sec. I, we sampled on a Clifford variant of our MSC protocol by replacing T ($T^\dagger$) gates with S ($S^\dagger$) gates following [29]. The Clifford variant would produce a logical $S|+\rangle$ state instead of a T state. We perform accurate state vector simulation [29] on the cultivation process of our MSC-3 protocol without circuit noise to verify its correctness. In Ref. [29], state vector simulation of the cultivation process (with $d_{\text{color}} = 3$) under circuit noise is also performed. It is observed in Ref. [29] that, at circuit noise rate $p = 0.001$, the error rate for cultivating a T state (estimated via state vector simulation) on a $d_{\text{color}} = 3$ color code is about two times larger than the error rate for cultivating an $S|+\rangle$ state (estimated via Monte Carlo sampling). In our work, however, state vector simulation is not efficient enough to provide enough samples to estimate the logical error rate under circuit noise rate $p = 0.001$, since our cultivation process (of the MSC-3 protocol) has a larger spatial footprint (24 qubits) compared to the cultivation process with $d_{\text{color}} = 3$ (using 15 qubits) in the original MSC protocol. It would be interesting to adopt more efficient sampling methods [61,62] to sample our cultivation process. We leave such an implementation to future work.

As RP-growth circuits and expansion circuits both require preparing Bell pairs, we run them concurrently with morphing circuits that transform SRP codes back to

RP codes. Take the expansion circuit in our MSC-3 protocol as an example. Morphing an SRP-3 code back to an RP-3 code is implemented by running three copies of the decoding circuit [containing four steps as in Fig. 7(b)] for the $[[4, 2, 2]]$ code. We see that since the third step of the decoding circuit, data qubits of the RP-3 code are no longer used in the decoding circuit and are available for the SE part of the expansion circuit. Thus, to prepare the Bell pairs, as a preliminary requirement for the expansion circuit, we first insert an initialization step that initializes all ancilla and data qubits for the expansion circuit between the first step and the second step of the decoding circuit. Then, at the second step of the decoding circuit, we concurrently run eight extra CNOT gates to prepare the Bell pairs in Fig. 11(c). At the third step of the decoding circuit, we can concurrently run the first round of SE in the expansion circuit. The rest of the SE circuit is then run after the last step, the decoding circuit. In this way, an RP-growth step or an expansion step is naturally merged with the preceding (backward) morphing stage.

Following Ref. [29], we parse our MSC protocol into circuit rounds as in Fig. 14. These rounds are naturally assigned different round lengths according to the number of resets and measurements contained in each round. More specifically,

- (1) morphing from an RP code to an SRP code is considered as a single round with length 0.5,
- (2) morphing from an SRP code to an RP code and growing (expanding) the RP code are bundled together as a round with length 1.5, and
- (3) other rounds all have length 1.

For each round, the expected space-time cost is defined as the product of the number of active qubits for this round, the survival rate at the beginning of this round, and the round length. Similar to Ref. [29], the space-time cost for our MSC protocol is defined as the sum of the expected space-time cost for each round divided by the survival rate at the end of the protocol.

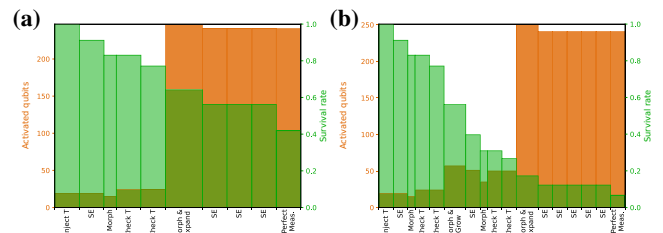


FIG. 14. (a) Activated qubits and survival rate at each round of the MSC-3 protocol. (b) Activated qubits and survival rate at each round of the MSC-5 protocol. Each bin corresponds to a single round and has a width proportional to the round length.

APPENDIX B: DETAILS ON CIRCUIT DESIGN

1. How to choose the final distance

Here, we present the reason for setting the final distance to 11 when we compare our MSC protocol with the original one [29]. We will also discuss the effect of the final distance on the success rate of our MSC protocol.

According to Fig. 16 in Ref. [29], the distance-15 grafted surface codes, as the final fully grown codes at the end of the original MSC protocol, share similar behavior in terms of the logical error rate with the distance-11 rotated surface code. Thus, in our MSC-3 and MSC-5 protocols, expanding the RP-3 or RP-5 code to a distance-11 rotated surface code suffices to compare fairly with the original MSC protocol [29]. Notice that the lowest achievable logical error rate of our MSC-3 protocol is orders of magnitude higher than that of our MSC-5 protocol. Thus, in our MSC-3 protocol, expanding the RP-3 code directly to a distance-11 rotated surface code (as we do in the MSC-5 protocol) can be wasteful. In Fig. 15, we see that for our MSC-3 protocol, the trade-offs between the logical error rate and the success rate are not affected by the choice of the distance of the rotated surface code that we grow our RP-3 code to. In practice, to store the prepared logical T state in a large rotated surface code with distance d , we can first expand our RP code to a rotated surface code with distance $d_1 < d$, and perform a round of post-selection based on the soft outputs (for instance, expanding the RP-3 code to a distance-7 rotated surface code for our MSC-3 protocol), and then expand the rotated surface code to a larger one with distance d . In this way, the first expansion stage with a non-negligible discard rate ($\sim 20\%$ for our MSC-3 protocol) has a smaller spatial footprint compared to directly expanding to a distance- d rotated surface code.

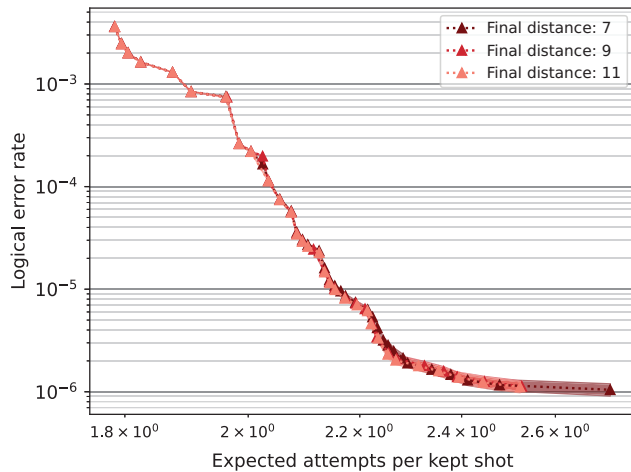


FIG. 15. Performance of our MSC-3 protocol under various final distances.

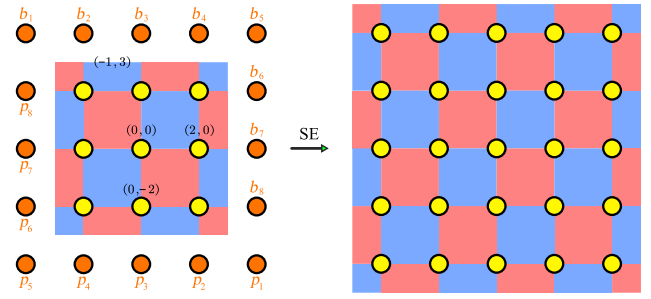


FIG. 16. Role of Bell pairs for growing an RP code. The coordinate system has a horizontal x -axis and a vertical y -axis and guarantees that qubits have even-integer coordinates and stabilizers have odd-integer coordinates.

2. Bell pairs for growth and expansion

In our MSC protocol, Bell pairs are used during injection, RP code growth, and RP code expansion to provide suitable detectors (or detecting regions [50]). In the following, we illustrate the functionalities of Bell pairs through a concrete example—growing an RP-3 code to an RP-5 code [Figs. 11(b) and 16]. For ease of referencing stabilizers and qubits, we set a coordinate system as in Fig. 16. Each stabilizer on an RP-3 (or RP-5) code patch is labeled by its center coordinates (x, y) and denoted by $S_{RP-3, (x, y)}$ (or $S_{RP-5, (x, y)}$). First, the Bell pairs guarantee that each stabilizer of the RP-3 code will be effectively measured again at the growth SE round that grows the RP-3 code to an RP-5 code. For instance, consider a boundary Z stabilizer $S_{RP-3, (-1, 3)}$ of the RP-3 code. By multiplying with the Z stabilizers provided by Bell pairs $\{b_2, p_2\}$ and $\{b_3, p_3\}$, $S_{RP-3, (-1, 3)}$ becomes the product of two stabilizers on the RP-5 code, $S_{RP-5, (-1, 3)} \cdot S_{RP-5, (1, -3)}$, which will be measured in the growth SE round. In this way, the measurement result on $S_{RP-3, (-1, 3)}$ at SE round immediately before the growth step and the measurement results on $S_{RP-5, (-1, 3)}$ and $S_{RP-5, (1, -3)}$ at the growth SE round form a detector. Moreover, Bell pairs fix the sign of all corner and boundary stabilizers on the RP-5 code. For instance, $\{b_1, p_1\}$ imposes a $+1$ corner stabilizer $S_{RP-5, (-5, 5)}$ on the RP-5 code. Similarly, $\{b_1, b_2, p_1, p_2\}$ generates a $+1$ boundary stabilizer $S_{RP-5, (-3, 5)}$. In our simulation, Bell pairs are created in a standard way by applying CNOT gates on pairs of qubits initialized in $|0\rangle$ and $|+\rangle$, respectively. Consequently, Bell-pair preparation is also subject to circuit noise according to the uniform depolarizing circuit noise model (as in Ref. [29]) described in Appendix A.

3. Guidelines on experimental implementation

Our MSC protocol requires nonlocal connectivity (or qubit reconfiguration) at a restricted scale. For neutral atom platforms on which modular and parallel operations are preferred [63], careful optimization is required for the implementation of our protocol. In the following, we

will go through all components of our MSC protocol and present guidelines on how to implement them efficiently with neutral atoms. As experimental techniques and hardware architecture designs are still under active development, the guidelines are not expected to be a detailed list of instructions but are intended to illustrate, in principle, how to tailor our MSC protocol to have high modularity—a feature well-suited for neutral atoms.

- (1) SE on the RP code. Nonlocal CNOT gates are needed to measure corner and boundary stabilizers as shown in Figs. 5(d) and 5(e). A straightforward implementation would be to move the ancilla qubits associated with these stabilizers across the code patch during SE. One alternative that removes the need to perform intra-cycle cross-patch movements [for the nonlocal CNOT gates in Figs. 5(d) and 5(e)] is to replace each corner and boundary ancilla qubit with a Bell pair prepared offline. For example, a weight-four Z stabilizer measurement circuit with a single ancilla qubit in Fig. 17(a) can be replaced by the measurement circuit in Fig. 17(b) with a Bell pair. We can see from Fig. 17(c) that with all corner and boundary ancilla qubits replaced by Bell pairs, all CNOT gates of the original SE circuit are now local. We will still need to generate

and rearrange these Bell pairs and perform Bell measurement on them after completing the CNOT gates in Fig. 17(c). These operations require non-local qubit rearrangement but can be performed offline. Notice that similar configurations of Bell pairs to the one in Fig. 17(c) are also required for the injection, code-growth, and expansion circuits. Thus, in practice, it would be convenient to allocate a particular space for the offline generation and rearrangement of Bell pairs. Bell-pair generation and rearrangement are illustrated in Fig. 17(e) and will be discussed later on. As mentioned in Sec. II, for our MSC protocol, it is not necessary to measure the corner stabilizers. This observation inspires another way to simplify the rearrangement operations. After removing the corner ancilla qubits and the associated CNOT gates, we can see that the first two (or the last two) CNOT gates associated with a boundary ancilla qubit are on the same boundary. We can modify the schedule of the CNOT gates such that all ancilla-qubit rearrangements occur between the third and fourth layer of CNOT gates [Fig. 17(d)]. (Our modification does not alter the relative ordering between the CNOT gates. We make our SE circuit less compact to ease the requirement on ancilla-qubit rearrangement, which is a reasonable

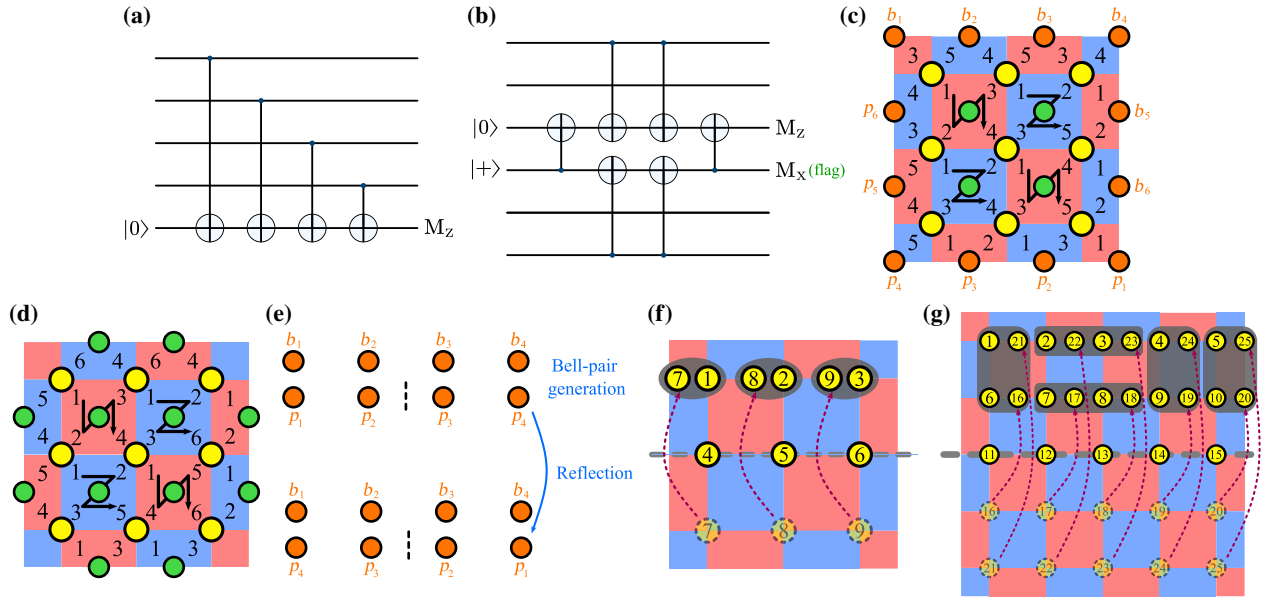


FIG. 17. Practical implementation of our MSC protocol. (a) Measuring a weight-4 Z stabilizer by a single ancilla qubit. (b) Measuring a weight-4 Z stabilizer by a Bell pair. One qubit in the Bell pair provides the stabilizer measurement result; the other functions as a flag qubit. (c) Syndrome extraction circuit on the RP-3 code with the boundary and corner ancilla qubits replaced by Bell pairs. All CNOT gates on layers 1–5 now only require local connectivity. (d) Syndrome extraction circuit without corner stabilizer measurements. Scheduling of the CNOT layers is loosened to synchronize the rearrangement operations for boundary ancilla qubits. (e) Preparation of a specific configuration of Bell pairs along the top and bottom boundaries in (c). (f) Folding the RP-3 code patch. Each shaded region represents a pair of qubits that will be encoded into a $[[4, 2, 2]]$ code. (g) Folding the RP-5 code patch. Each shaded region represents a group of qubits that will be encoded into a $[[6, 4, 2]]$ code. Qubits in each line-shaped region need to be rearranged to the same configuration as the rectangular-shaped regions.

trade-off given the long qubit coherence time for neutral atoms.) We leave the benchmarking of these altered SE circuits, as well as the designing of more compact and experimentally friendly SE circuits, for future work.

- (2) Morphing the RP code to the SRP code. This step encodes groups of data qubits on the RP code into $[[4, 2, 2]]$ or $[[6, 4, 2]]$ codes. To implement this, we first need to fold the RP code patch along its fold line as shown in Figs. 17(f) and 17(g). In our MSC protocol, the folding operation involves at most two steps of row translations [Figs. 17(f) and 17(g)]. (Folding operations are also used for implementing the logical S gate transversally on the rotated or unrotated surface code [12, 13].) Then, we need to run, in parallel, the encoding circuit of the $[[4, 2, 2]]$ (or $[[6, 4, 2]]$) code on the data qubits in each shaded region in Fig. 17(f) [or in Fig. 17(g)].
- (3) Logical verification. As illustrated in Fig. 9 and Fig. 10, the logical verification circuits are highly modularized and, therefore, are well-suited for neutral atoms.
- (4) Bell-pair preparation. This is a crucial ingredient used multiple times in our MSC protocol. Without loss of generality, we discuss the preparation of Bell pairs along the top and bottom boundaries in Fig. 17(c). We first generate a row of Bell pairs in a standard way by applying a CNOT gate to each pair of qubits initialized in $|+\rangle$ and $|0\rangle$, respectively [Fig. 17(e)]. We then perform a reflection operation according to Fig. 17(e) such that each qubit on the bottom row switches places with its mirrored qubit across the symmetric axis [dashed lines in Fig. 17(e)] through the midpoint of the bottom row. By moving the bottom row of qubits downward, we can obtain the Bell-pair configuration along the top and bottom boundaries in Fig. 17(c).

- [1] S. B. Bravyi and A. Y. Kitaev, Quantum codes on a lattice with boundary, [arXiv:quant-ph/9811052](#).
- [2] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Topological quantum memory, *J. Math. Phys.* **43**, 4452 (2002).
- [3] D. Horsman, A. G. Fowler, S. Devitt, and R. V. Meter, Surface code quantum computing by lattice surgery, *New J. Phys.* **14**, 123011 (2012).
- [4] Google Quantum AI and Collaborators, Quantum error correction below the surface code threshold, *Nature* **638**, 920 (2025).
- [5] 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).
- [6] D. Litinski, A game of surface codes: Large-scale quantum computing with lattice surgery, *Quantum* **3**, 128 (2019).

- [7] H. Bombin, C. Dawson, R. V. Mishmash, N. Nickerson, F. Pastawski, and S. Roberts, Logical blocks for fault-tolerant topological quantum computation, *PRX Quantum* **4**, 020303 (2023).
- [8] M. E. Beverland, S. Huang, and V. Kliuchnikov, Fault tolerance of stabilizer channels, [arXiv:2401.12017](#).
- [9] G. P. Gehér, C. McLauchlan, E. T. Campbell, A. E. Moylett, and O. Crawford, Error-corrected Hadamard gate simulated at the circuit level, *Quantum* **8**, 1394 (2024).
- [10] B. J. Brown, K. Laubscher, M. S. Kesselring, and J. R. Wootton, Poking holes and cutting corners to achieve clifford gates with the surface code, *Phys. Rev. X* **7**, 021029 (2017).
- [11] C. Gidney, Inplace access to the surface code Y basis, *Quantum* **8**, 1310 (2024).
- [12] J. E. Moussa, Transversal Clifford gates on folded surface codes, *Phys. Rev. A* **94**, 042316 (2016).
- [13] Z.-H. Chen, M.-C. Chen, C.-Y. Lu, and J.-W. Pan, Transversal logical Clifford gates on rotated surface codes with reconfigurable neutral atom arrays, [arXiv:2412.01391](#).
- [14] S. Bravyi and A. Kitaev, Universal quantum computation with ideal Clifford gates and noisy ancillas, *Phys. Rev. A* **71**, 022316 (2005).
- [15] A. G. Fowler and S. J. Devitt, A bridge to lower overhead quantum computation, [arXiv:1209.0510](#).
- [16] C. Jones, Low-overhead constructions for the fault-tolerant Toffoli gate, *Phys. Rev. A* **87**, 022328 (2013).
- [17] Y. Li, A magic state's fidelity can be superior to the operations that created it, *New J. Phys.* **17**, 023037 (2015).
- [18] A. G. Fowler and C. Gidney, Low overhead quantum computation using lattice surgery, [arXiv:1808.06709](#).
- [19] C. Gidney and A. G. Fowler, Efficient magic state factories with a catalyzed $|CCZ\rangle \rightarrow 2|T\rangle$ transformation, *Quantum* **3**, 135 (2019).
- [20] C. Gidney and A. G. Fowler, Flexible layout of surface code computations using AutoCCZ states, [arXiv:1905.08916](#).
- [21] T. Itogawa, Y. Takada, Y. Hirano, and K. Fujii, Even more efficient magic state distillation by zero-level distillation, *PRX Quantum* **6**, 020356 (2025).
- [22] Y. Hirano, T. Itogawa, and K. Fujii, in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, Montreal, Canada), Vol. 1, pp. 843–853.
- [23] D. Gottesman and I. L. Chuang, Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations, *Nature* **402**, 390 (1999).
- [24] X. Zhou, D. W. Leung, and I. L. Chuang, Methodology for quantum logic gate construction, *Phys. Rev. A* **62**, 052316 (2000).
- [25] M. Vasmer and D. E. Browne, Three-dimensional surface codes: Transversal gates and fault-tolerant architectures, *Phys. Rev. A* **100**, 012312 (2019).
- [26] 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).
- [27] S. Heußen and J. Hilder, Efficient fault-tolerant code switching via one-way transversal CNOT gates, *Quantum* **9**, 1846 (2025).

- [28] S.-H. Lee, F. Thomsen, N. Fazio, B. J. Brown, and S. D. Bartlett, Low-overhead magic state distillation with color codes, *PRX Quantum* **6**, 030317 (2025).
- [29] C. Gidney, N. Shutty, and C. Jones, Magic state cultivation: growing T states as cheap as CNOT gates, [arXiv:2409.17595](https://arxiv.org/abs/2409.17595).
- [30] D. Bluvstein, H. Levine, G. Semeghini, T. T. Wang, S. Ebadi, M. Kalinowski, A. Keesling, N. Maskara, H. Pichler, M. Greiner, V. Vuletić, and M. D. Lukin, A quantum processor based on coherent transport of entangled atom arrays, *Nature* **604**, 451 (2022).
- [31] Quantinuum and Collaborators, A race-track trapped-ion quantum processor, *Phys. Rev. X* **13**, 041052 (2023).
- [32] H. Zhou, C. Zhao, M. Cain, D. Bluvstein, C. Duckering, H.-Y. Hu, S.-T. Wang, A. Kubica, and M. D. Lukin, Low-overhead transversal fault tolerance for universal quantum computation, *Nature* **646**, 303 (2025).
- [33] M. Serra-Peralta, M. H. Shaw, and B. M. Terhal, Decoding across transversal Clifford gates in the surface code, [arXiv:2505.13599](https://arxiv.org/abs/2505.13599).
- [34] M. Cain, D. Bluvstein, C. Zhao, S. Gu, N. Maskara, M. Kalinowski, A. A. Geim, A. Kubica, M. D. Lukin, and H. Zhou, Fast correlated decoding of transversal logical algorithms, [arXiv:2505.13587](https://arxiv.org/abs/2505.13587).
- [35] M. L. Turner, E. T. Campbell, O. Crawford, N. I. Gillespie, and J. Camps, Scalable decoding protocols for fast transversal logic in the surface code, [arXiv:2505.23567](https://arxiv.org/abs/2505.23567).
- [36] E. Knill, Fault-tolerant postselected quantum computation: Schemes, [arXiv:quant-ph/0402171](https://arxiv.org/abs/quant-ph/0402171).
- [37] H. Goto, Step-by-step magic state encoding for efficient fault-tolerant quantum computation, *Sci. Rep.* **4**, 7501 (2014).
- [38] Y. Vakhnin, S. Jacoby, A. Grimsmo, and A. Retzker, Magic state cultivation on the surface code, [arXiv:2502.01743](https://arxiv.org/abs/2502.01743).
- [39] N. P. Breuckmann and S. Burton, Fold-transversal clifford gates for quantum codes, *Quantum* **8**, 1372 (2024).
- [40] M. H. Freedman and D. A. Meyer, Projective plane and planar quantum codes, *Found. Comput. Math.* **1**, 325 (2001).
- [41] M. Barkeshli, P. Bonderson, M. Cheng, C.-M. Jian, and K. Walker, Reflection and time reversal symmetry enriched topological phases of matter: Path integrals, non-orientable manifolds, and anomalies, *Commun. Math. Phys.* **374**, 1021 (2020).
- [42] R. Kobayashi and G. Zhu, Cross-cap defects and fault-tolerant logical gates in the surface code and the honeycomb floquet code, *PRX Quantum* **5**, 020360 (2024).
- [43] A. Kitaev, Fault-tolerant quantum computation by anyons, *Ann. Phys.* **303**, 2 (2003).
- [44] N. Meister, C. A. Pattison, and J. Preskill, Efficient soft-output decoders for the surface code, [arXiv:2405.07433](https://arxiv.org/abs/2405.07433).
- [45] C. Gidney, Sinter: Fast QEC sampling, <https://pypi.org/project/sinter/>, accessed: 2025-03-13.
- [46] C. Gidney, Stim: A fast stabilizer circuit simulator, *Quantum* **5**, 497 (2021).
- [47] H. Bombin, G. Duclos-Cianci, and D. Poulin, Universal topological phase of two-dimensional stabilizer codes, *New J. Phys.* **14**, 073048 (2012).
- [48] A. Kubica, B. Yoshida, and F. Pastawski, Unfolding the color code, *New J. Phys.* **17**, 083026 (2015).
- [49] B. Criger and B. Terhal, Noise thresholds for the $[[4, 2, 2]]$ -concatenated toric code, *Quantum Inf. Comput.* **16**, 1261 (2016).
- [50] M. McEwen, D. Bacon, and C. Gidney, Relaxing hardware requirements for surface code circuits using time-dynamics, *Quantum* **7**, 1172 (2023).
- [51] E. Knill, Quantum computing with realistically noisy devices, *Nature* **434**, 39 (2005).
- [52] S. Yoshida, S. Tamiya, and H. Yamasaki, Concatenate codes, save qubits, *npj Quantum Inf.* **11**, 88 (2025).
- [53] H. Goto, High-performance fault-tolerant quantum computing with many-hypercube codes, *Sci. Adv.* **10**, eadp6388 (2024).
- [54] N. Berthussen, S. J. S. Tan, E. Huang, and D. Gottesman, Adaptive syndrome extraction, *PRX Quantum* **6**, 030307 (2025).
- [55] C. Gidney, M. Newman, P. Brooks, and C. Jones, Yoked surface codes, *Nat. Commun.* **16**, 4498 (2025).
- [56] C. Gidney, Answer to “How to force PyMatching into the opposite equivalence class” on quantum computing stack exchange (2024), <https://quantumcomputing.stackexchange.com/a/35582>.
- [57] O. Higgott and C. Gidney, Sparse blossom: Correcting a million errors per core second with minimum-weight matching, *Quantum* **9**, 1600 (2025).
- [58] C. Gidney and M. Ekerå, How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits, *Quantum* **5**, 433 (2021).
- [59] N. Fazio, M. Webster, and Z. Cai, Low-overhead magic state circuits with transversal CNOTs, [arXiv:2501.10291](https://arxiv.org/abs/2501.10291).
- [60] Z. Chen, Cultiv-T-RP2, https://github.com/Zihan-Chen-PhMA/Cultiv_T_RP2, accessed: 2025-03-21
- [61] S. Bravyi, D. Browne, P. Calpin, E. Campbell, D. Gosset, and M. Howard, Simulation of quantum circuits by low-rank stabilizer decompositions, *Quantum* **3**, 181 (2019).
- [62] A. Kissinger, J. van de Wetering, and R. Vilmart, in *LIPICs, TQC* (Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Urbana Champaign, Illinois, USA, 2022), Vol. 232, p. 5:1.
- [63] D. Bluvstein *et al.*, Logical quantum processor based on reconfigurable atom arrays, *Nature* **626**, 58 (2024).