Fault-Tolerant Gates on Hypergraph Product Codes

Anirudh Krishna and David Poulin
Département de Physique and Institut Quantique, Université de Sherbrooke, Sherbrooke, Québec J1K 2R1, Canada

(Received 2 April 2020; revised 22 August 2020; accepted 15 December 2020; published 4 February 2021)

Quantum computers have the capacity to change the landscape of computing. To make these devices robust to experimental imperfections, we require some form of quantum error correction. Current approaches focus on topological codes, as they appear to be in the realm of experimental capabilities. These techniques will likely lead to the first proof of principle of fault tolerance in the next 5 yr. In the long run, however, these approaches require a very large number of physical qubits to construct a fault-tolerant quantum circuit that can run interesting algorithms. As an alternative, it is proposed that quantum low-density parity-check (LDPC) codes could serve as an architecture that circumvents this problem. The best candidate class of quantum LDPC codes were discovered by Tillich and Zémor and are called hypergraph product codes. These codes have a constant encoding rate and were recently shown to have a constant fault-tolerant error threshold. With these features, they asymptotically offer a smaller overhead compared to topological codes. Here, we demonstrate how to perform Clifford gates on this class of codes using code deformation. To this end, we introduce wormhole defects on hypergraph product codes. Together with state injection, we can perform a universal set of gates within a single block of the class of hypergraph product codes.

DOI: 10.1103/PhysRevX.11.011023 Subject Areas: Quantum Information

I. INTRODUCTION

Quantum computers have the capacity to change the landscape of computing. Recent advances demonstrate that we are capable of constructing quantum devices that cannot be simulated by their classical counterparts [1]. The next major milestone on the road to quantum computers will be to demonstrate that we can perform quantum error correction. This correction will be necessary to run interesting quantum algorithms.

Quantum error correcting codes can be used to simulate an ideal quantum circuit using noisy circuit components [2–5]. These simulations are characterized by a tug of war between the size of the circuit and the accuracy of the result. Larger problem instances require larger circuits and, therefore, larger error correcting codes; this requirement, however, creates more opportunities for errors to accumulate. To compensate for this increased error accumulation, we must lower the logical fault rate. By choosing our quantum error correcting code appropriately, we can increase the code size n logarithmically with the size of the logical circuit and still find that the global logical error rate decreases exponentially in the code size.

Not all error correcting codes are equivalent. The size of the noisy circuit depends on the quantum error correcting code being used. Finding codes which minimize the size of the circuit to achieve a target logical error rate is a subject of active research (see, for example, Refs. [6–11]).

A large amount of work in both theory and experiment focuses on topological codes [12–17]. These codes have the desired ability to suppress errors exponentially in the size n of the codes. Furthermore, topology guarantees that each qubit, be it a data qubit or ancilla qubit for readout, is connected only to a constant number of other qubits in its neighborhood. Arguably, the most popular example of a topological code is the surface code [13]. This code can be laid out on a two-dimensional square grid; each qubit is placed on the vertices of the grid and connected only to its neighbors to the North, East, West, and South. Albeit simple, this code has been shown to perform well when studied using numerical simulations (see, for example, Ref. [18]). These properties, among others, make these codes suitable for serving as an architecture for quantum computers. One drawback of topological codes is their ability to store logical qubits. Indeed, each logical qubit in a topological code is encoded in a distinct block of size n. The number of physical qubits required to simulate a single logical qubit, therefore, increases with the size of the quantum computer.

In contrast, quantum *low-density parity-check* (LDPC) codes [19,20] are generalizations of topological codes that overcome this increasing encoding overhead. LDPC codes refer to families of codes where all qubits (data qubits and ancilla qubits for readout) are connected only to a constant number of other qubits. This constant is independent of the block size and, thus, simplifies the process of syndrome extraction. In Ref. [20], Gottesman proposes a construction that combines techniques for efficient syndrome extraction with ideas to perform logical gates on block codes. The result is a conditional statement: If "good" quantum LDPC codes exist, the number of physical qubits required to simulate a logical qubit becomes a constant, independent of the size of the quantum computer.

Good LDPC codes are elusive. Ideally, we would like to find quantum LDPC codes whose rate and relative distance are a constant relative to the size of the code. It is hard to enforce the commutation relations between stabilizers of a quantum code while simultaneously maintaining low connectivity. Topological codes use topology to achieve this task, but there are strong constraints on the number of logical qubits they can simulate and how effectively this task can be done [21,22]. The difference between LDPC codes and topological codes is that the connectivity need no longer be spatially local. By sacrificing locality, LDPC codes can overcome one of the shortcomings of topological codes, and this sacrifice permits a "wholesale effect." Increasing the size n of the code lets us encode k qubits, where k can increase with n. We can find codes for which the logical error probability decreases exponentially with n with a fixed k/n ratio. This result is to be contrasted with topological codes that also achieve an exponential error suppression with n but with a vanishing encoding rate. Although no doubt simpler to engineer, locality is not a fundamental constraint. There exist techniques that permit qubits that are not adjacent to share entanglement in various architectures [23–26]. This fact motivates theoretical investigations of LDPC codes that are not constrained by locality. Only with a complete understanding of the potential benefits of LDPC codes will we be able to decide if they are worth the extra experimental effort.

One of the leading candidates for LDPC codes are the so-called hypergraph product codes which engineer commutation relations using algebraic or graph-theoretic techniques [27]. The hypergraph product is itself not a code family but rather a technique to construct quantum codes from classical codes. If we input two classical codes to the hypergraph product machinery, the resulting quantum code inherits properties of the classical codes. Importantly, if the classical code families are LDPC, then the quantum code families are also LDPC. Furthermore, if the classical code families have a code dimension k scaling linearly in the block size n, then so does the code dimension of the quantum code family. With regards to distance, the hypergraph product code construction yields quantum codes with distance scaling as the square root of the block size. Up to constants, this result is the same functional dependence between the distance and the block size as the surface code (but it applies to all the logical qubits). In short, these codes achieve similar error suppression as topological codes but do so at a constant encoding rate.

A series of recent works also shows the suitability of these codes for fault-tolerant quantum computation. Leverrier, Tillich, and Zémor [28] show that a certain class of these codes can be equipped with an efficient decoding algorithm called small-set flip. Not only can this decoding algorithm correct adversarial errors of weight proportional to the distance, it can also correct a constant fraction of stochastic errors [29]. Furthermore, it is even resilient to syndrome errors [30]; we need only to measure the syndromes once in order to proceed with decoding. This resilience is in contrast to the surface code, where syndromes need to be measured several times before we can proceed with decoding [31]. These works put the hypergraph product code on solid theoretical foundation.

Photonic integrated circuits (PICs) are one potential avenue for experimental realizations of quantum LDPC codes [32–34]. In architectures where qubits are encoded in photons, we can likely support nonlocal interactions using linear optics. Furthermore, the measurements required in this setting are single-qubit measurements. Quantum computation on PICs is based on the model of measurement-based quantum computation originally due to Raussendorf and co-authors [15–17,35]. This model implements parity measurements via two-qubit entangling gates and single-qubit measurements in the Pauli basis. While the original model by Raussendorf *et al.* is based on the two-dimensional surface code, generalizations appear in a Letter by Bolt *et al.* [36]. The authors discuss how to obtain a very large class of error correcting codes via single-qubit measurements from a larger cluster state using a technique called *foliation*. A follow-up work [37] studies the decoding problem on certain classes of LDPC codes. With developments in hypergraph product codes, photonic architectures may permit one way to realize these codes some day.

Main contributions.—In this paper, we describe the first techniques to perform fault-tolerant gates on hypergraph product codes. Our method is an instance of code deformation, a general framework to perform gates on quantum codes. The idea behind code deformation is to iteratively modify a quantum error correcting code so that each step is an elementary transformation. These elementary transformations ensure that errors on the quantum error correcting code do not spread uncontrollably between successive iterations.

At the outset, it is unclear how to proceed with code deformation for hypergraph product codes. Constructing quantum LDPC codes is already a challenging task. Indeed, there have been attempts to construct good quantum LDPC

codes dating back to 2004 [38,39] based on Cayley graphs, using topology on hyperbolic manifolds [40–42], or using a homological product (related to the hypergraph product) [43]. Demanding the existence of an LDPC code family where each element in a sequence is related to its successor by an elementary transformation only compounds this issue. While topological codes boast techniques to perform gates via code deformation [15,44–47], we have already pointed out that they are rate limited. Recent proposals to perform gates on homological codes require that the underlying codes obey other properties in addition to being LDPC [48].

Here, we begin with a hypergraph product code created as the product of two good classical LDPC codes. We then modify the code by temporarily switching off certain patches of our code. In other words, we do not measure the state of the qubits in these patches, henceforth referred to as "defects" on the code. Iteratively changing the location of these defects facilitates fault-tolerant logical operations. In the spirit of the hypergraph product construction, we express defects as purely algebraic and graph-theoretic concepts. Importantly, the code remains LDPC over the course of code deformation. Our proposal also places no constraints on the underlying LDPC codes that are used to construct the quantum LDPC codes. Fault tolerance follows, because the modifications we make at each step are local (in a graph-theoretic sense). The generalized defects are capable of encoding several logical qubits. We choose to use a subset of these qubits to encode information and leave the rest in a trivial state. We refer to these qubits that do not encode information as gauge qubits. We discuss constraints on code deformation that keeps the spaces of logical and gauge qubits separate.

Techniques to perform gates on topological codes do not generalize to the hypergraph product code class in a natural way. Code deformation on topological codes relies on the simple geometry and resulting symmetries. As a consequence, it is unclear how to express these techniques on hypergraph product codes. Instead, we introduce a new class of defects on hypergraph product codes called wormholes. These defects allow us to perform a certain class of gates on qubits corresponding to the Clifford group. The Clifford group corresponds to a finite set of unitary gates that play a central role in quantum error correction [49]. They correspond to the set of unitary gates that map the Pauli group to themselves under conjugation. It is well known that these gates form a maximal finite subgroup of the set of unitary gates on qubits. Together with any gate outside the group, we can generate a universal gate set. For our purposes, we choose to use the T gate, defined as the 2×2 diagonal unitary gate $\mathrm{diag}(1, e^{i\pi/4})$. We show that we can achieve a universal gate set on the logical qubits via state injection [50]. State injection is a technique to perform gates essential for quantum computation and employs quantum teleportation as a primitive.

We emphasize at this point that we are not providing a technique to *compile* a Clifford gate of interest. We provide a framework within which it is possible to realize Clifford gates via code deformation. These Clifford gates can then be composed to generate a larger group of transformations. We show that the framework is sufficiently rich to realize all different types of generating gates, but, depending on the code, this framework may or may not encompass the entire space of Clifford operators. We conclude by showing that, in certain instances, it is possible to understand when a certain gate can be performed. We rephrase topological notions in the hypergraph product code in terms of graph connectivity. In particular, we show that this problem reduces to an Eulerian cycle. This result, in turn, implies that, in certain special instances, we can efficiently verify when a certain operation is possible.

This approach can be contrasted to Gottesman's work, which entails partitioning logical qubits into blocks of LDPC codes. Each block is of "intermediate" size, and a computation with k logical qubits requires blocks whose size scales as $O[k/\mathrm{poly}\log(k)]$. Gates are then performed by state injection using ancilla states. The size of these blocks limits the number of gates that can be implemented at any given time. Furthermore, the savings of LDPC codes become compelling only as we increase the block size. Partitioning the logical qubits into blocks implies that it will take longer for this effect to manifest. In contrast, we propose performing quantum computation on a single block. Our proposal does not limit the number of qubits that can be processed at any given time to a constant. On the other hand, the time required to perform a gate could scale, so it is unclear whether these gates will be faster.

Outline of the paper.—In Sec. II, we review essential properties of classical codes and hypergraph product codes. In Sec. III, we describe how to deform the hypergraph product code by introducing a *puncture*. This puncture shall itself be described as a hypergraph product of subgraphs of the graphs that together form the quantum code. Section III A includes the definition of a puncture and describes how they arise in two types: smooth and rough. In Sec. III C, we describe the logical operators supported on the puncture. We generalize puncture defects in Sec. IV and discuss how to create a wormhole. This defect is essential to perform Clifford gates.

In Sec. V, we study code deformation. Unlike a surface code, a puncture is capable of supporting several logical qubits. We encode qubits in some of these logical qubits and choose to use the rest as gauge qubits that do not encode information. In Sec. V A, we impose constraints on code deformation with multiple logical qubits to ensure that these two spaces do not mix. We proceed to list the requirements to perform all Clifford gates on the hypergraph product code in Sec. V B. After discussing state injection in Sec. V D, we conclude by discussing pointlike punctures in Sec. V E. Pointlike punctures are useful in

demonstrating how the notion of topology may generalize to purely graph-theoretic conditions. From a condensed-matter physics perspective, these punctures can be seen as a generalization of anyons from two-dimensional manifolds to general graph structures, which could be of independent interest. We conclude with a summary and list some of the many open questions that need to be addressed.

II. BACKGROUND AND NOTATION

A. Classical and quantum codes

1. Classical codes

A classical code $\mathcal{C} = [n, k, d] \subseteq \mathbb{F}_2^n$ over n bits is the (right) kernel of a matrix $H \in \mathbb{F}_2^{m\times n}$, known as its parity-check matrix. The code dimension k is the dimension of the kernel, $\dim[\ker(H)]$, and d is the minimum Hamming distance between a pair of vectors in $\mathcal{C}$. In general, we could have redundant checks, and so $m \geq n - k$. We let $\mathrm{rs}(H)$ denote the row span of the matrix and $\mathrm{im}(H)$ denote its image; H^t is the transpose of H. Each row of H encodes a parity constraint that we refer to as a check which we label $c \in C$. Likewise, we label the columns of H by variable indices $v \in V$. Let $\mathbb{1}_V$ and $\mathbb{1}_C$ denote the identity on $\mathbb{F}_2^V$ and $\mathbb{F}_2^C$, respectively. For $u, v \in \mathbb{F}_2^n$, we let $\langle u, v\rangle = \sum_{i=1}^n u_i v_i$ denote the inner product between them. For $u \in \mathbb{F}_2^n$, $\mathrm{supp}\{u\}$ denotes those indices in $\{1, \ldots, n\}$ such that u is nonzero.

An LDPC code is a code family $\{\mathcal{C}_n\}_n$ such that the number of bits in the support of a check is upper bounded by a constant as a function of the block size n, as is the number of checks a bit is connected to [51]. In other words, the number of nonzero elements in each row and column of the parity-check matrix is bounded by a constant with respect to the block size n. The factor graph $\mathcal{G}(\mathcal{C})$ of a code $\mathcal{C}$ lets us infer properties of the code $\mathcal{C}$ from the properties of the graph. The graph $\mathcal{G}(\mathcal{C}) = (V \cup C, E)$ is a bipartite graph, where $V = [n]$ and $C = [m]$. We draw an edge between check node $c \in C$ and variable node $v \in V$ if and only if $H_{cv} = 1$. Given a subset $P \subseteq V \cup C$, the neighborhood of P is denoted $\Gamma(P)$ and is defined as

$$\Gamma(P) = \{q | (q, p) \quad \text{or} \quad (p, q) \in E \quad \text{for } p \in P\}.$$

2. Quantum codes

Let $\mathcal{P} = \{I, X, Y, Z\}$ denote the Pauli group and $\mathcal{P}_n = \mathcal{P}^{\otimes n}$ denote the n-fold tensor product of the Pauli group. A quantum error correcting code is specified by a group $\mathcal{Q} \subseteq \mathcal{P}_n$. The stabilizer $\mathcal{S}$ of the code is the center $Z(\mathcal{Q})$, and the quotient group $\mathcal{G} := \mathcal{Q}/\mathcal{S}$ is called the (pure) gauge group. The set of logical operators is then defined as $\mathcal{N}(\mathcal{Q})\backslash\mathcal{Q}$, where $\mathcal{N}(\mathcal{Q})$ defines the normalizer of the group $\mathcal{Q}$. A stabilizer code is a code such that $\mathcal{G} = \varnothing$ and $\mathcal{Q}$ is an Abelian group.

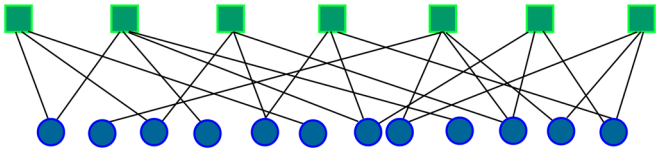


FIG. 1. A generic bipartite graph with two types of vertices C (green square vertices) and V (blue circular vertices). Edges run only between blue vertices and green vertices and never between two vertices of the same color.

Just like the individual rows of the classical parity-check matrix generate a linear space of constraints, we can choose a generating set of checks for the stabilizer group $\mathcal{S}$. Much like its classical counterpart, the factor graph $\mathcal{G}_Q$ can be use to represent the stabilizer generator and provides a visual representation of $\mathcal{Q}$. The only difference is that the edges could carry labels of Pauli elements X, Y, or Z, indicating the action of a check on a qubit.

B. The hypergraph product code

The hypergraph product code is a way to construct a quantum code $\mathcal{Q}$ given two classical codes $\mathcal{C}_1$ and $\mathcal{C}_2$. This code can naturally be extended to families of classical codes. If the two classical code families are LDPC, then so is the resulting quantum code family. The resulting code is a CSS code [52,53]; i.e., the stabilizer generators are products of either only X operators or only Z operators. For simplicity, we consider the graph product of a code $\mathcal{C}$ with itself.

1. Graph-theoretic description

Let $\mathcal{G} = (V \cup C, E)$ be a bipartite graph. Every hypergraph can equivalently be expressed as a bipartite graph, i.e., with two sets of nodes V and C such that all (undirected) edges are of the form $(v, c) \in V \times C$. Using this equivalence, the hypergraph product is just a graph product, obtained as the Cartesian product of the graph with itself. Such a bipartite graph is depicted in Fig. 1.

Let $\mathcal{Q}$ denote the quantum code obtained from the graph product of $\mathcal{G}$ with itself. The factor graph $\mathcal{G}_Q$ of $\mathcal{Q}$ is defined as $\mathcal{G}_Q = \mathcal{G} \times \mathcal{G}$. Its nodes are partitioned as follows:

(1) qubits $V \times V \cup C \times C$;
(2) X stabilizers $V \times C$;
(3) Z stabilizers $C \times V$.

This representation highlights that this construction yields two kinds of qubits—those emerging from the product of two variable nodes (VV nodes) and those emerging from the product of two check nodes (CC nodes). We draw an edge between (a_1, a_2) and (b_1, b_2) in $V \cup C \times V \cup C$ if either $(a_1, b_1) \in E$ and $a_2 = b_2$ or if $(a_2, b_2) \in E$ and $a_1 = b_1$.

2. Algebraic description

Let $H \in \mathbb{F}_2^{m\times n}$ define the codes $\mathcal{C} = [n, k, d]$ and $\tilde{\mathcal{C}} = [m, \tilde{k}, \tilde{d}]$ as

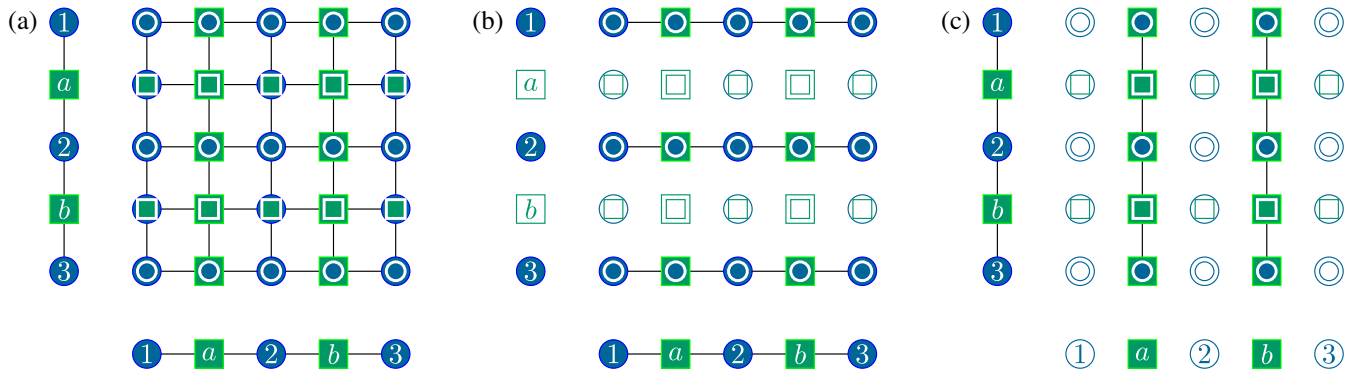


FIG. 2. (a) Hypergraph product representation of the surface code with smooth boundaries above and below and rough boundaries on the sides. (b) The action of X stabilizers on VV qubits and (c) CC qubits on the right. Filled nodes represent nodes involved in the stabilizer generator.

$$\mathcal{C} = \ker(H), \qquad \tilde{\mathcal{C}} = \ker(H^t). \tag{1}$$

The X and Z stabilizers of the code are specified via their symplectic representation [49]. The parity-check matrices of the quantum code are denoted H_X and H_Z, respectively, where

$$\begin{aligned} H_X &= (\mathbb{1}_V \otimes H | H^t \otimes \mathbb{1}_C), \\ H_Z &= (H \otimes \mathbb{1}_V | \mathbb{1}_C \otimes H^t). \end{aligned} \tag{2}$$

In this expression, VV nodes are in the left partition, while CC nodes are in the right partition. Let $d_{\min} = \min\{d, \tilde{d}\}$ denote the minimum of the distance of the two codes. The hypergraph product $\mathcal{Q}$ is a $[\![n^2 + m^2, k^2 + \tilde{k}^2, d_{\min}]\!]$ quantum code.

We refer to the logical operators of the quantum code $\mathcal{Q}$ as the embedded logical operators to distinguish them from the logical operators that we introduce later by creating defects. The embedded logical operators of the code are described below. We assume that H and H^t do not span $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$ respectively.

Lemma 1 (embedded logical operators).—

(1) The X logical operators of $\mathcal{Q}$ are spanned by

$$\begin{aligned} &\{\ker(H) \otimes [\mathbb{F}_2^n / \mathrm{rs}(H)] | \mathbf{0}_{m^2}\} \\ &\quad \cup \{\mathbf{0}_{n^2} | [\mathbb{F}_2^m / \mathrm{rs}(H^t)] \otimes \ker(H^t)\}. \end{aligned}$$

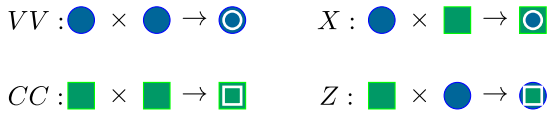
(2) The Z logical operators of $\mathcal{Q}$ are spanned by

$$\begin{aligned} &\{[\mathbb{F}_2^n / \mathrm{rs}(H)] \otimes \ker(H) | \mathbf{0}_{m^2}\} \\ &\quad \cup \{\mathbf{0}_{n^2} | \ker(H^t) \otimes [\mathbb{F}_2^m / \mathrm{rs}(H^t)]\}. \end{aligned}$$

The style of the proof follows arguments presented in Lemma 17 of Ref. [27]. For completeness, we include it in Supplemental Material, Sec. A [54].

3. Example

Consider the surface code generated using two copies of the repetition code [3,1,3] as shown in Fig. 2. Its factor graph $\mathcal{G}$ runs vertically on the left and horizontally on the bottom. Variable nodes are colored blue and indexed by numerals, whereas check nodes are green and indexed by letters. The product of two nodes is represented as



Mapping this product to the surface code, we choose a convention where Z stabilizers correspond to plaquettes and X stabilizers correspond to vertices. The two classical codes are the [3,1,3]-repetition codes to the left and bottom of the figure.

To break it down further, we could consider each of the constituent parts of the definition above. First, note that the qubits of VV type, depicted using two concentric circles, represent horizontal edges in the surface code, whereas qubits of CC type, depicted using concentric squares, are vertical edges of the surface code. The action of the X stabilizers on qubits of VV type is defined by the matrix $\mathbb{1}[1,2,3] \otimes H$ and is depicted in Fig. 2(c). Similarly, the action of the X stabilizers on qubits of CC type is defined by $H^t \otimes \mathbb{1}[A,B]$ and is depicted in Fig. 2(d).

In a similar manner, we can obtain the representation for Z stabilizers. By overlaying these diagrams, we obtain Fig. 2.

III. PUNCTURES

A puncture is a defect on the hypergraph product created by removing both qubits and stabilizers belonging to some (small) portion of the code. This defect shall be effected by measuring single-qubit Pauli operators within the interior of the puncture. This process is similar to creating a puncture on the surface code [55].

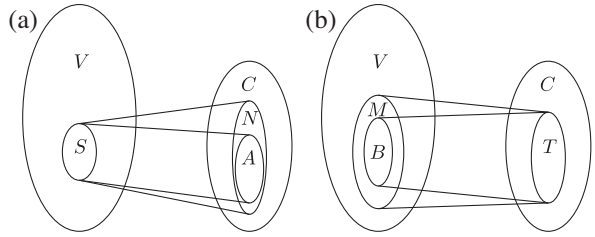


FIG. 3. Schematic of factor graphs. (a) denotes the subgraph induced by S. N is its neighborhood, and A is its ancestor. (b) denotes the subgraph induced by T. M is its neighborhood, and B is its ancestor.

A. Definition

We begin this section with some notation. Let $S \subseteq V$ denote a connected subset of variable nodes. In other words, for any two variable nodes $u, v \in S$, there exists a path connecting u and v such that each element of that path is in S or a neighbor of an element in S. $N = \Gamma(S) \subseteq C$ is its neighborhood, and $A = \Gamma^{-1}(S)$ is its ancestor as shown in Fig. 3(a):

$$N = \{c \in C : \exists\, u \in S \text{ such that } (u, c) \in E\},$$
$$A = \{c \in C : \forall\ \ u \in \Gamma(c), u \in S\}.$$

For any set $V' \subseteq V$, we let $\mathbb{1}_{V'}$ denote the projector on V' over $\mathbb{F}_2^V$. We also write $H_{V'} = H\mathbb{1}_{V'}$ for the restriction of the parity-check matrix to V'.

Similarly, let $T \subseteq C$ denote a connected subset of check nodes. $M = \Gamma(T) \subseteq V$ is its neighborhood, and $B = \Gamma^{-1}(T)$ is its inverse neighborhood as shown in Fig. 3(b):

$$M = \{v \in V : \exists\, c \in T \text{ such that } (v, c) \in E\},$$
$$B = \{v \in V : \forall\ \ c \in \Gamma(v), c \in T\}.$$

For any set $C' \subseteq C$, we let $\mathbb{1}_{C'}$ denote the projector on C' over $\mathbb{F}_2^C$. We also write $H_{C'} = \mathbb{1}_{C'}H$ for the restriction of the parity-check matrix to C'.

At this juncture, we make some observations that will be useful later:

$$\Gamma(A) \subseteq S \Rightarrow \Gamma(S^c) \subseteq A^c, \qquad A^c = N^c \cup (N \backslash A),$$
$$\Gamma(B) \subseteq T \Rightarrow \Gamma(T^c) \subseteq B^c, \qquad B^c = M^c \cup (M \backslash B).$$

To create a puncture on the quantum code, we stop measuring certain stabilizers and modify others when carving out a portion of the interior. The punctures are classified by how stabilizers are modified.

Definition 2 (smooth puncture).—Let $S \subseteq V$ and $T \subseteq C$ be connected sets of variable and check nodes. Let N, A and M, B denote induced sets as defined above. A smooth puncture is defined by the stabilizers H'_X and H'_Z, where

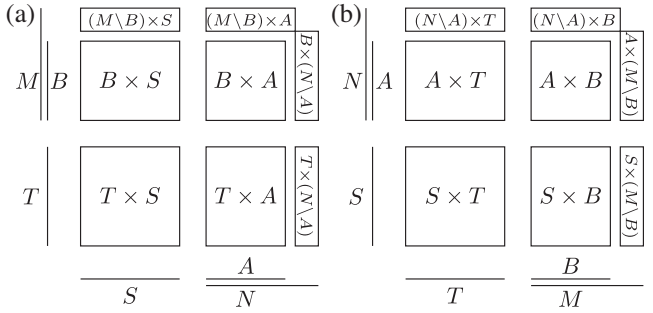


FIG. 4. Schematic for the puncture. The subgraphs selected by T and S are flattened and placed below and to the left. Their product is represented using four quadrants. The qubits are in the North-West and South-East quadrants. The Z stabilizers are in the South-West quadrant. The X stabilizers are in the North-East quadrant. (a) Smooth puncture defined by T and S. The Z stabilizers $T \times S$ are completely within the puncture and are, thus, not broken. (b) Rough puncture defined by S and T. The X stabilizers $S \times T$ are completely within the puncture and are, thus, not broken.

$$H'_X = (\mathbb{1}_B \otimes H_S | H_T^t \otimes \mathbb{1}_A),$$
$$H'_Z = (H_T \otimes \mathbb{1}_S | \mathbb{1}_T \otimes H_S^t).$$

Note that this puncture is not exactly the graph product of the two subgraphs selected by T and S. This result is verified by noting that it is missing elements from $M \times S$. Rather, it follows the hypergraph product construction on the interior nodes of both graphs. For simplicity, we abuse notation and refer to this result as the graph product $T \times S$.

The defining trait of a smooth puncture is that Z stabilizers are not broken across its boundary. We refer to the schematic in Fig. 4(a). Such stabilizers would have to be of the form (c, v) for some check $c \in C$ and $v \in V$ where either the check node c or the variable node v are in the boundary of T or S, respectively. This constraint does not exist by construction—check nodes in T are contained entirely within the puncture, as are variable nodes in S. The internal qubits of a smooth puncture are the nodes $B \times S \cup T \times A$ and are measured in the Z basis to create the puncture. The qubits on the boundary of a smooth puncture correspond to the sets

$$(M \backslash B) \times S \cup T \times (N \backslash A).$$

The X stabilizers on the boundary of a smooth puncture correspond to the sets

$$(M \backslash B) \times N \cup M \times (N \backslash A).$$

Their support on the interior of the puncture, $B \times S \cup T \times A$, is removed. Therefore, X stabilizers on the boundary of a smooth puncture are broken.

In a similar manner, a rough puncture can be created by interchanging the roles of T and S on the graphs. It is formally defined as follows.

TABLE I. Summary of properties of punctures. We assume that $S \subseteq V$ and $T \subseteq C$ are (connected) subsets of variable and check nodes. S induces the sets N and A, its neighborhood and ancestor, respectively, and similarly T induces the sets M and B.

	Smooth puncture	Rough puncture
H'_X	$(\mathbb{1}_B \otimes H_S \vert H_T^t \otimes \mathbb{1}_A)$	$(\mathbb{1}_S \otimes H_T \vert H_S^t \otimes \mathbb{1}_T)$
H'_Z	$(H_T \otimes \mathbb{1}_S \vert \mathbb{1}_T \otimes H_S^t)$	$(H_S \otimes \mathbb{1}_B \vert \mathbb{1}_A \otimes H_T^t)$
Internal qubits	$(B \times S) \cup (T \times A)$	$(S \times B) \cup (A \times T)$
Boundary qubits	$(M \backslash B) \times S \cup T \times (N \backslash A)$	$S \times (M \backslash B) \cup (N \backslash A) \times T$
Boundary X checks	$(M \backslash B) \times A \cup B \times (N \backslash A)$	$\varnothing$
Boundary Z checks	$\varnothing$	$A \times (M \backslash B) \cup (N \backslash A) \times B$

Definition 3 (rough puncture).—Let $S \subseteq V$ and $T \subseteq C$ be connected sets of variable and check nodes. Let N, A and M, B denote induced sets as defined above. A rough puncture is defined by the stabilizers H'_X and H'_Z, where

$$H'_X = (\mathbb{1}_S \otimes H_T \vert H_S^t \otimes \mathbb{1}_T),$$
$$H'_Z = (H_S \otimes \mathbb{1}_B \vert \mathbb{1}_A \otimes H_T^t).$$

Abusing notation, this puncture can be thought of as a graph product $S \times T$.

The defining trait of a rough puncture is that X stabilizers are not broken across the boundary. We refer to the schematic in Fig. 4(b) for the following discussion. Such stabilizers would have to be of the form (v, c) for some check $c \in C$ and $v \in V$ where either the check node c or the variable node v are in the boundary of S or T, respectively. For the same reasons as before, such nodes do not exist. The internal qubits of a rough puncture are the nodes $S \times B \cup A \times T$ and are measured in the X basis to create the puncture. The qubits on the boundary of a rough puncture correspond to the sets

$$S \times (M \backslash B) \cup (N \backslash A) \times T.$$

The Z stabilizers on the boundary of a rough puncture correspond to the sets

$$N \times (M \backslash B) \cup (N \backslash A) \times M.$$

Their support on the interior of the puncture, $S \times B \cup A \times T$, is removed. Hence, Z stabilizers on the boundary of a rough puncture are broken.

To deform $\mathcal{Q}$, we remove the edges that are contained in a puncture. Algebraically, it is described by $H_X + H'_X$ and $H_Z + H'_Z$. This code is itself not a hypergraph product code but is clearly LDPC. We are merely puncturing an LDPC code; by removing edges, we cannot increase the weight of checks.

We first show that the code defined this way obeys the desired commutation relations.

Lemma 4.—The punctured code forms a valid stabilizer code.

The proof of this lemma can be found in Supplemental Material, Sec. B [54].

For convenience, we summarize this section in Table I.

B. Example: Surface code

We illustrate these ideas with the surface code. This code is formed using two [5,1,5] repetition codes. We choose the sets $T = \{b, c\}$ and $S = \{3, 4\}$. For the sake of completeness, all the neighbors and ancestors are listed below. These subsets, along with the quantum code they generate, are shown in Fig. 5:

$$T = \{b, c\}, \quad M = \{2, 3, 4\}, \quad B = \{3\},$$
$$S = \{3, 4\}, \quad N = \{b, c, d\}, \quad A = \{c\}.$$

In the algebraic description, we let H_T and H_S be the matrices correspond to the subcodes corresponding to the subsets selected above:

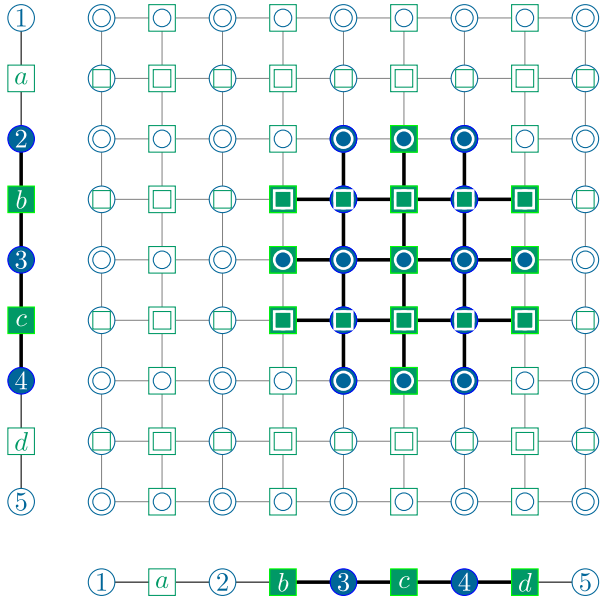


FIG. 5. Subcode created by $T \times S$. Shaded nodes are part of the puncture.

$$H_T = \mathbb{1}\{b,c\}H = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$H_S = H\mathbb{1}\{3,4\} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

The X and Z stabilizers that are removed from the puncture can then be described, respectively, as

$$H'_X = (\mathbb{1}_B \otimes H_S | H_T^t \otimes \mathbb{1}_A),$$
$$H'_Z = (H_T \otimes \mathbb{1}_S | \mathbb{1}_T \otimes H_S^t).$$

C. Logical Pauli operators for punctures

Mirroring the surface code, punctured hypergraph product codes support two types of logical operators—*loop-type operators* that exist only on the boundary of the puncture and *chain-type operators* that are supported on the boundary of the puncture and also extend into the rest of the code. For the rest of this section, we let $T \subseteq C$ and $S \subseteq V$ be some connected subsets. The sets $M \subseteq V$ and $N \subseteq C$ are the respective neighborhoods, and $B \subseteq V$ and $A \subseteq C$ are the respective ancestors. We derive the form of the logical operators for a smooth puncture defined as above. The logical operators of a rough puncture follow by exchanging the roles of S and T.

Before proceeding, we impose certain constraints on how S and T are chosen. These constraints apply to both smooth and rough punctures. The constraints stipulate that certain subcodes associated with S and T are *correctable*; i.e., if these portions of the code are erased, then we do not lose any code words of the underlying code in this process.

Definition 5 (correctability).—A puncture defined by $T \subseteq C$ and $S \subseteq V$ is *correctable* if and only if

(1) $\ker(H_S) = \ker(H_T^t) = \varnothing$ and
(2) $\ker(H_A^t) = \ker(H_B) = \varnothing$.

This definition helps establish that creating a puncture does not affect the embedded logical operators. It leads to the proof of the following lemmas; the proofs can be found in Supplemental Material [54], Sec. C.

We begin by considering the embedded Z logical operators.

Lemma 6.—There are no embedded Z logical operators supported within the interior of the puncture.

Similarly, we can consider the embedded X logical operators.

Lemma 7.—There are no embedded logical X operators within the interior of the puncture.

We later argue that these conditions can be used together with the cleaning lemma [21] to guarantee that the embedded logical operators of the quantum code are unaffected by the puncture.

We eventually show that certain operators are orthogonal to the stabilizers $H_X + H'_X$ and $H_Z + H'_Z$. Under certain conditions, some of these operators do not lie in the span of the stabilizers themselves. To this end, we now state and prove two lemmas that are useful for proving this claim.

We assume that the classical codes obey certain independence relations.

Definition 8.—A puncture defined by $T \subseteq C$ and $S \subseteq V$ obeys *independence relations* if and only if

(1) $\mathrm{rs}(H_{T^c}) \cap \mathrm{rs}(H_T) = \mathrm{rs}(H_{S^c}^t) \cap \mathrm{rs}(H_S^t) = \varnothing$ and
(2) $\ker(H_{T^c}) \cap \ker(H_T) = \ker(H_{S^c}^t) \cap \ker(H_S^t) = \varnothing$.

Remark.—We clarify the meaning of condition 2. Of course, if $g \in \ker(H)$, then $g \in \ker(H_{T^c})$ as well as $\ker(H_T)$. We disregard these operators—$\ker(H_{T^c})$ and $\ker(H_T)$ refer to $\ker(H_{T^c}) \backslash \ker(H)$ and $\ker(H_T) \backslash \ker(H)$, respectively. Likewise, $\ker(H_{S^c}^t)$ and $\ker(H_S^t)$ refer to $\ker(H_{S^c}^t) \backslash \ker(H^t)$ and $\ker(H_S^t) \backslash \ker(H^t)$, respectively.

The next lemma is useful in showing that operators in the row space of H'_Z are not in the span of the Z stabilizers $H_Z + H'_Z$; i.e., they are not redundant. These operators are later used to construct Z logical operators.

Lemma 9.—The stabilizers $H_Z + H'_Z$ outside the puncture are independent of the stabilizers H'_Z within the puncture; i.e.,

$$\mathrm{rs}(H_Z + H'_Z) \cap \mathrm{rs}(H'_Z) = \varnothing.$$

The proof of this lemma can be found in Supplemental Material [54], Sec. D. In a similar way, the following lemma is useful in showing that certain X operators in the normalizer are not in the span of the stabilizer. In turn, this fact is used to construct the X logical operators.

Lemma 10.—Let $\alpha \in \mathbb{F}_2^{n^2+m^2}$ such that $\mathrm{supp}\{\alpha\} \cap (M \backslash B) \times S \cup T \times (N \backslash A) \neq \varnothing$ and it lies in one of the two following sets:

(1) $[\ker(H_{T^c}) \otimes \mathbb{F}_2^S / \mathrm{rs}(H_A) | \mathbf{0}_{m^2}]$;
(2) $[\mathbf{0}_{n^2} | \mathbb{F}_2^T / \mathrm{rs}(H_B') \otimes \ker(H_{S^c}^t)]$.

Then, α does not lie in the row span of $H_X + H'_X$.

The proof of this lemma can be found in Supplemental Material [54], Sec. D.

Logical operators.—With these conditions, we can study the logical Z operators that emerge by creating a puncture. These operators can be classified in terms of the (classical) code spaces associated with H_A and H_B'.

Theorem 11.—Let $\tilde{G}_B$ and G_A be the generator matrices for the code spaces defined by H_B^t and H_A, respectively; i.e., the rows of $\tilde{G}_B$ and G_A span $\ker(H_B^t)$ and $\ker(H_A)$, respectively. The logical Z operators are spanned by

$$(\tilde{G}_B^t \otimes G_A) H'_Z.$$

Proof.—We begin from first principles. The stabilizers are given by $H_X + H'_X$, and the single-qubit operators in the

interior of the puncture are described by the matrix $I_{\text{int}} = (\mathbb{1}_B \otimes \mathbb{1}_S | \mathbb{1}_T \otimes \mathbb{1}_A)$. The logical operators are defined as $\ker(H_X + H'_X + I_{\text{int}})/\text{rs}(H_Z + H'_Z)$.

Part 1: $\ker(H_X + H'_X + I_{\text{int}})$.—Suppose $\alpha \in \mathbb{F}_2^{n^2+m^2}$ such that $\alpha \in \ker(H_X + H'_X + I_{\text{int}})$. We can assume that α is not supported in the interior and consider the kernel of $H_X + H'_X$ instead of $H_X + H'_X + I_{\text{int}}$. We use Lemma 6 together with the cleaning lemma [21] to note that the embedded logicals are unaffected by the puncture. Any other Z-type operator that is supported in the interior anticommutes with the single-qubit X measurements used to generate the puncture and, therefore, is removed.

The operator α must, therefore, lie in the kernel of H_X outside the puncture. This kernel contains the embedded logical operators and products of old stabilizers that are not in the interior. We focus on only the latter here in order to obtain the new logical operators.

The stabilizers in the interior are spanned by H'_Z. Those operators in $\text{rs}(H'_Z)$ but not supported in the interior are thus what we seek. The interior of the puncture corresponds to $B \times S \cup T \times A$. Let $a \in \mathbb{F}_2^{C \times V}$ such that $\text{supp}\{a\} \subseteq T \times S$. We want the projection of aH'_Z to vanish in the interior, i.e.,

$$aH'_Z \mathbb{1}_{B \times S \cup T \times A} = 0 \tag{3}$$

$$\Rightarrow a(H_B \otimes \mathbb{1}_S | \mathbb{1}_T \otimes H_A^t) = 0. \tag{4}$$

Inspecting the VV and CC parts of this equation separately, we find that we must have

$$a^t \in \ker(H_B^t) \otimes \ker(H_A). \tag{5}$$

Equivalently, the space we desire is spanned by

$$(\tilde{G}_B^t \otimes G_A) H'_Z. \tag{6}$$

Part 2: $\text{rs}(H_Z + H'_Z)$.—We refer to Lemma 9, which states that the span of the stabilizers from within the puncture are independent of those outside the puncture. Therefore, the space defined by Eq. (6) is not in the span of the Z stabilizers. ∎

We now discuss the logical X operators associated to a smooth puncture.

The next lemma helps show that certain X operators are not in the span of the stabilizer $H_X + H'_X$. These operators are then used to construct logical X operators. These are comprised of code words of the classical codes that are complementary to the subgraphs chosen by S and T. In other words, they involve the terms $\ker(H_{T^c})$ and $\ker(H_{S^c}^t)$.

In the proof that follows, we make certain claims on these spaces. Note that, since the neighborhood of the set B is contained in the set T, it implies that the neighborhood of T^c is contained within B^c. Similarly, the neighborhood of S^c is contained within A^c. Therefore, when studying $\ker(H_{T^c})$ and $\ker(H_{S^c}^t)$, we assume that their support is contained in B^c and A^c, respectively.

These operators can be classified in terms of the (classical) code spaces associated with H_{T^c} and $H_{S^c}^t$.

Theorem 12.—Let O_Z be the Z operators defined by

$$O_Z = (H_M \otimes \mathbb{1}_S | \mathbb{1}_T \otimes H_N^t),$$

and let $\Omega_X = \ker(H_Z + O_Z)$ denote the X-type operators in its kernel.

The logical X operators are described by

$$\begin{aligned}&(\{\ker(H_{T^c}) \otimes [\mathbb{F}_2^S/\text{rs}(H_A)] | \mathbf{0}_{m^2}\} \\ &\quad \cup \{\mathbf{0}_{n^2} | [\mathbb{F}_2^T/\text{rs}(H_B^t)] \otimes \ker(H_{S^c}^t)\})/\Omega_X.\end{aligned} \tag{7}$$

Before proceeding to the proof, we make the following observations and highlight important features of this claim. At first glance, the logical operators appear to break into two types, the VV-type logicals defined by $\ker(H_{T^c})$ and the CC-type operators defined by $\ker(H_{S^c}^t)$. Thus, the logical X operators are defined by the code spaces that are left over after the portions corresponding to T and S are carved out.

The set $H_Z + O_Z$ represents Z stabilizers *outside* the puncture. Vectors in the kernel of $H_Z + O_Z$ are unaffected by the addition of the puncture and in that sense represent some invariant space. The space Ω_X, thus, contains X stabilizers and logicals whose support does not overlap with the puncture. To help this object seem less alien, let us return to the surface code and consider an example.

Consider a smooth puncture defined on a surface code with only smooth boundaries. This puncture could define a logical qubit with the X string running from the boundary of the smooth puncture to one of the boundaries of the lattice. Depending on the arrangement, this logical operator could be supported only on CC qubits or only VV qubits as shown in Fig. 6. However, these two objects are equivalent up to the stabilizer. This equivalence is captured by Ω_X.

Furthermore, consider a lattice with two smooth and rough boundaries as shown in Figs. 6(c) and 6(d). The two representations of the logical X operator shown running from the smooth puncture to the boundary are equivalent up to an embedded logical X operator and X stabilizers. This equivalence is also captured by Ω_X.

With these comments, we proceed to the proof of Theorem 12.

Proof.—We start from first principles. Recall that we define the matrix $I_{\text{int}} = (\mathbb{1}_B \otimes \mathbb{1}_S | \mathbb{1}_T \otimes \mathbb{1}_A)$. The logicals are defined as

$$\ker(H_Z + H'_Z)/\text{rs}(H_X + H'_X + I_{\text{int}}).$$

Part 1: $\ker(H_Z + H'_Z)$.—We would like to understand the structure of vectors $\alpha \in \ker(H_Z + H'_Z)$. We assume that α is not supported on the interior of the puncture

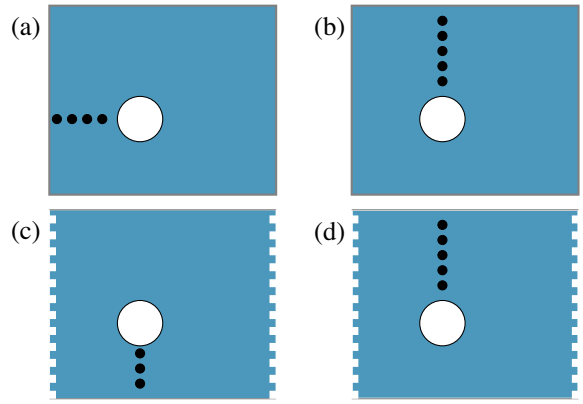


FIG. 6. (a) and (b) feature a smooth puncture defined on a lattice with only smooth boundaries. The strings of X operators defined only on VV qubits [as in (a)] or only on CC qubits [as in (b)] are equivalent. (c) and (d) feature a lattice with smooth and rough boundaries, with a smooth punctured carved out from the inside. The logical X shown running from the smooth puncture to the boundary in (c) and (d) are equivalent up to an embedded logical X.

$B \times S \cup T \times A$. If α is supported within the interior, it can be removed using the single-qubit operators described by I_{int}. As shown by Lemma 7, the embedded logical X operators are unaffected by the puncture.

We argue that α ought to have a certain structure using a proxy. Let us define $a \in \mathbb{F}_2^{C\times V}$ as

$$a = H_Z\alpha = H'_Z\alpha. \tag{8}$$

The only occasion when a is nontrivial is when α is supported on the boundary. If not, α is either a stabilizer or logical belonging to the code that is unaffected by the puncture. We shall mod out by this set, and this result corresponds to Ω_X.

The VV and CC portions of Eq. (8) stipulate that
(1) $a \subseteq \text{im}(H_T \mathbb{1}_{M\setminus B}) \cap \text{im}(H \mathbb{1}_{B^c}) \otimes \mathbb{F}_2^S$;
(2) $a \subseteq \mathbb{F}_2^T \otimes \text{im}(H_S^t \mathbb{1}_{N\setminus A}) \cap \text{im}(H \mathbb{1}_{A^c})$,
respectively.

Equivalently, this stipulation means that

$$\alpha \in [\ker(H_{T^c}) \otimes \mathbb{F}_2^S | \mathbf{0}_{m^2}] \cup [\mathbf{0}_{n^2} | \mathbb{F}_2^T \otimes \ker(H_{S^c}^t)]. \tag{9}$$

Part 2: $\text{rs}(H_X + H'_X + I_{\text{int}})$.—Since the operators we are interested in lie only outside the puncture by assumption, we may ignore I_{int} and need only concern ourselves with $\text{rs}(H_X + H'_X)$. Let $g \in \ker(H_{T^c})$ and $x \in \text{rs}(H_A)$ such that $aH_A = x$. Its product $g \otimes x$ is clearly in $\ker(H_{T^c}) \otimes \text{rs}(H_A)$ and, therefore, in the kernel of $H_Z + H'_Z$. We show that this vector lies in the span of the X stabilizers as well. Indeed, it can be expressed as

$$(g \otimes a)(H_X + H'_X) = g \otimes x.$$

In a similar manner, we can show that any vector in $\text{rs}(H'_B) \otimes \ker(H_{S^c}^t)$ is in the row span of $H_X + H'_X$.

As already shown in Lemma 9, any vector $\alpha \in \mathbb{F}_2^{n^2+m^2}$ that is in the row span of
(1) $[\ker(H_{T^c}) \otimes \mathbb{F}_2^S / \text{rs}(H_A) | \mathbf{0}_{m^2}]$ or
(2) $[\mathbf{0}_{n^2} | \mathbb{F}_2^T / \text{rs}(H'_B) \otimes \ker(H_{S^c}^t)]$
does not lie in the row span of the stabilizer $H_X + H'_X$.

The proof is complete. ∎

IV. WORMHOLES

We have now established how to construct punctures by carving out portions of the hypergraph product code. On the surface code, punctures facilitate CNOT gates on encoded qubits via braiding. While this braiding results in a nontrivial gate, it is limited as it maps physical (and logical) X operators to X operators and Z operators to Z operators. In other words, it is a CSS-preserving operation; i.e., stabilizer generators before and after the process act on qubits in their support as only X or only Z. To complete even just the Clifford group, we require operations that can map X operators to Z operators on the physical and logical levels. In particular, we need ways to perform logical single-qubit Clifford operations. On a two-dimensional code, twist defects [46,47,56,57] can be used to encode qubits and also perform single-qubit Clifford gates on these qubits.

Twist defects, however, rely on the structure of two-dimensional codes that do not naturally extend to general LDPC codes. For instance, an error chain on the surface code has two frustrated stabilizers on either end regardless of the length of the chain. On two-dimensional codes, twist defects rely on stabilizers becoming frustrated pairwise in order to create *hybrid* stabilizers, i.e., stabilizer generators whose support is both X and Z. LDPC codes, however, do not possess these properties. For instance, expander codes have the property that the number of frustrated stabilizers grows with the size of the error. It becomes highly nontrivial to construct a defect using these stabilizers. For these reasons, we have to look for other ways of generalizing twist defects.

In a companion paper [58], we introduce a defect called a wormhole that addresses this issue. Rather than rely on linelike defects, it builds upon and generalizes puncture defects. Since we already know how to construct punctures on the hypergraph product code, it is natural to extend them to wormholes. We briefly summarize the idea from our companion paper here for the sake of convenience. The key idea in the surface code is to entangle stabilizers along the boundaries of punctures. What this entanglement serves is to do is entangle the two punctures and put a twist on the boundary of one of the punctures. We can use this defect to encode logical qubits. Upon performing the entangling measurement, we obtain hybrid stabilizers whose weight does not scale with the size of the puncture. In this context as well, a hybrid stabilizer is one which acts on qubits in its

support as either X or Z. The resulting stabilizers are nonlocal—it is supported on the boundaries of both punctures. It is obtained as a product of two stabilizer generators from the underlying surface code. As shown there, and as repeated here, these stabilizers are created by measuring two-qubit Pauli operators. These measurements locally break the CSS nature of the code and serve as a resource to complete the Clifford group.

Returning to the general case of a hypergraph product code, we observe that, unlike a linelike defect, the boundary of a puncture is a well-defined entity. We therefore perform two-qubit measurements along the boundary of the puncture in this setting to define a wormhole defect. These two-qubit measurements serve to define hybrid stabilizers in much the same way.

Let $\mathcal{G} = (V \cup C, E)$ be a bipartite graph corresponding to a classical code $\mathcal{C}$. Consider a hypergraph product of a graph $\mathcal{G}$ with itself. As before, let $S \subseteq V$ and $T \subseteq C$ be connected subsets of variable nodes and check nodes, respectively. Furthermore, the induced subgraphs do not overlap; i.e., they obey $N \cap T = M \cap S = \varnothing$. These sets must be correctable; i.e., they obey conditions specified in Definition 5.

The wormhole is created by entangling the stabilizers along the boundaries of two punctures. This entanglement alters the structure of the code along the boundaries, and we must ensure that these enlarged regions remain correctable. Hence, we need to strengthen the notion of correctability to include the neighborhoods that define the punctures.

Definition 13 (extended constraints).—In addition to obeying Definitions 5 and 8, wormholes also need to obey

(1) $\ker(H_{\bar{S}}) = \ker(H^t_{\bar{T}}) = \varnothing$;
(2) $\mathrm{rs}(H_{\bar{T}^c}) \cap \mathrm{rs}(H_{\bar{T}}) = \mathrm{rs}(H_{\bar{S}^c}) \cap \mathrm{rs}(H_{\bar{S}}) = \varnothing$; and
(3) $\ker(H_{\bar{T}^c}) \cap \ker(H_{\bar{T}}) = \ker(H_{\bar{S}^c}) \cap \ker(H_{\bar{S}}) = \varnothing$.

This constraint is necessary because the stabilizers on the boundary of the puncture are removed to form hybrid stabilizers. To argue that the logical operators that emerge have certain properties, we use the above extended correctability condition. Equivalently, these can be thought of as the conditions for a puncture defined using the sets $\bar{S} \coloneqq M$ and $\bar{T} \coloneqq N$.

With these constraints established, we can associate a smooth puncture to the product $T \times S$ and a rough puncture to the product $S \times T$. These punctures are used to construct a wormhole in the following sections.

A. Measurements and hybrid stabilizers

Within the interior of the punctures, we perform the same measurements as we do to initialize a puncture—single-qubit X measurements within the smooth puncture and single-qubit Z measurements within the rough puncture. We then perform two-qubit measurements along the boundaries of the two punctures. This measurement yields hybrid stabilizers.

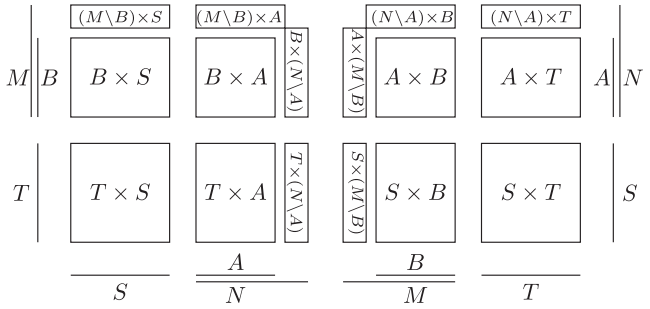


FIG. 7. Schematic to denote the wormhole. Smooth puncture on the left and rough puncture on the right.

For what follows, it is helpful to use the schematic for the smooth and rough puncture shown in Fig. 7. Recall that the boundaries of punctures are described as follows:

(1) Smooth puncture: $T \times (N\backslash A) \cup (M\backslash B) \times S$.
(2) Rough puncture: $(N\backslash A) \times T \cup S \times (M\backslash B)$.

For any VV qubit (u, u') or CC qubit (c, c'), we let $P(u, u')$ or $P(c, c')$, respectively, denote the single-qubit Pauli operator P on that qubit.

The hybrid stabilizers are generated by the following measurements along these boundaries.

(1) CC qubits: For every $c \in N\backslash A$, $c' \in T$, measure $X(c, c') \otimes Z(c', c)$; we denote this measurement as

$$(N\backslash A) \times T \leftrightarrow T \times (N\backslash A).$$

(2) VV qubits: For every $u \in S$, $u' \in M\backslash B$, measure $X(u, u') \otimes Z(u', u)$; we denote this measurement as

$$S \times (M\backslash B) \leftrightarrow (M\backslash B) \times S.$$

These two-qubit measurements do not commute with the stabilizers located on the boundary of the puncture. The next proposition provides an appropriate choice of hybrid stabilizers that commute with these measurements. Suppose P and Q are some sets of variable and check nodes, respectively; sets of the form $P \times Q$ refer to X stabilizers and $Q \times P$ to Z stabilizers. We write $P \times Q \leftrightarrow Q \times P$ to denote the hybrid stabilizer formed by pairing stabilizers in a natural way. In other words, for $p \in P$ and $q \in Q$, we let $P \times Q \leftrightarrow Q \times P$ denote the hybrid stabilizers acting as X on the support of (p, q) and Z on the support of (q, p). As a matter of convention, the operators with X support are always denoted on the left and those with Z support on the right.

The set of proposed hybrid stabilizers contains stabilizers that are adjacent to the puncture minus those in the interior. In other words, this set of stabilizers lives on the boundary of the punctures.

Lemma 14.—Upon performing the measurements listed above, the new hybrid stabilizers are associated with

$$(M \times N)\backslash(B \times A) \leftrightarrow (N \times M)\backslash(A \times B),$$

where the double arrow denotes the one-to-one pairing between the two sets as described above.

The proof of this lemma can be found in Supplemental Material [54], Sec. E.

The weight of the hybrid stabilizers is, thus, independent of the size of the code. For this reason, this construction guarantees that we still have an LDPC code.

Thus, when creating a wormhole, we begin as before by carving out certain portions of the code. Then, we perform two-qubit measurements along the boundaries of these punctures. We remove X and Z stabilizers from the code either because they lie within a puncture or they anticommute with a two-qubit measurement and are replaced by a hybrid stabilizer.

The code, thus, has the following X, Z, and hybrid (denoted h) stabilizers:

$$\begin{aligned} X&: (V\times C)\backslash[(S\times T)\cup(M\times N)],\\ Z&: (C\times V)\backslash[(T\times S)\cup(N\times M)],\\ h&: (M\times N)\backslash(B\times A)\leftrightarrow(N\times M)\backslash(A\times B),\\ &\quad (N\backslash A)\times T\leftrightarrow T\times(N\backslash A),\\ &\quad S\times(M\backslash B)\leftrightarrow(M\backslash B)\times S. \end{aligned}$$

We conclude by reiterating the origin of these objects. Note that there are two punctures that are used to create the wormhole, one smooth and one rough. The set of all X stabilizers corresponds to $V\times C$, but we subtract those stabilizers in these punctures. This result corresponds to $S\times T$ from the rough puncture and $M\times N$ from the smooth puncture and the hybrid stabilizers. Similarly, the set of all Z stabilizers corresponds to $C\times V$, but we subtract the stabilizers $T\times S$ from the smooth puncture and $N\times M$ from the rough puncture and the hybrid stabilizers. The hybrid stabilizers are created using the boundaries of these sets and are stated in Proposition 14. The last two lines of hybrid stabilizers correspond to the two-qubit measurements used to generate the wormhole. We remind the reader that, in this notation, operators with X support are to the left of the arrow, and those with Z support are to the right of the arrow.

B. Logical Pauli operators for wormholes

When we create a wormhole from two punctures, there are two ways in which stabilizers are updated. First, there are stabilizers within the puncture that are jettisoned, because they anticommute with the single-qubit measurements. Second, there are stabilizers on the boundary of the puncture that are replaced by hybrid stabilizers. This fact results in two sets of logical operators for the wormhole as we see below. As in the case of punctures, there are two varieties of logical operators: loop-type operators and chain-type operators. These logical operators are inherited from the underlying punctures.

The first set of logical operators can be described as the logical operators corresponding to the punctures $S\times T$ and $T\times S$. Given the symmetry of the construction, there is a one-to-one correspondence between the loop-type logical X operators around the rough puncture $S\times T$ and the loop-type logical Z operators around the smooth puncture $T\times S$.

Lemma 15.—The logical Z operators correspond to loop-type operators around one of the punctures. They come in two sets which are described as follows.

Type 1.—The first set of logical Z operators corresponds to the smooth puncture $T\times S$.

Type 2.—The second set of logical Z operators corresponds to the smooth puncture $N\times M$.

Proof.—To show that these objects are no longer part of the stabilizer group but commute with the X and Z stabilizers, we point to the proof of Theorem 11. The development is similar save for the new stabilizers, the new two-qubit operators that are measured along the boundaries of the two punctures.

Type 1.—Note that the measurements surrounding the smooth puncture are of Z type and, therefore, commute with the loop-type logical Z operators of type 1. Furthermore, these logical operators are defined only along the boundary of the puncture $T\times S$. A product of the two-qubit stabilizers is necessarily defined on both punctures, as X on the puncture $T\times S$ and Z on the puncture $S\times T$. Therefore, this constraint implies that the proposed logical operators of type 1 cannot be in the span of the stabilizers.

Type 2.—The logical operators of type 2 are defined on $[\Gamma(N)\backslash S]\times M\cup N\times[\Gamma(M)\backslash T]$. Thus, they do not interact with the two-qubit measurements. For the same reason, they cannot be expressed as a product of the two-qubit measurement operators. By Definition 13, these objects do not affect the embedded logical operators and are themselves not in the span of the Z stabilizer. ∎

Before proceeding to the conjugate logical operators, it is useful to highlight a symmetry of this construction. For logical Z operators, we choose the vector of Z operators around the puncture $T\times S$ for type 1 and $N\times M$ for type 2. Equivalently, we could choose the vector of X operators around the puncture $S\times T$ for type 1 or $M\times N$ for type 2. The next lemma states that these two choices are equivalent.

Lemma 16.—Every logical loop-type operator for a wormhole has two equivalent representations: an X-type loop around one puncture or a Z-type loop around the other.

Proof.—We deal with each type in turn.

Type 1.—Consider loop-type logical X operators that emerge from the puncture $S\times T$. These logical operators are supported on $(N\backslash A)\times T\cup S\times(M\backslash B)$. Each qubit on this boundary has a unique partner on the other boundary $T\times(N\backslash A)\cup(M\backslash B)\times S$. By symmetry, there is a one-to-one correspondence between the loop-type logical X operators on $S\times T$ and the loop-type logical Z operators on $T\times S$. These can be mapped to one another because of the two-qubit measurements.

Type 2.—Let $\tilde{G}_S$ and G_T be the matrices whose rows span $\ker(H_S^t)$ and $\ker(H_T)$, respectively.

Let $g \in \tilde{G}_S^t$ and $f \in G_T^t$ be any two rows of $\tilde{G}_S^t$ and G_T^t, respectively. By the considerations above and Theorem 11, we can define α_Z as a loop-type operator around $T \times S$, where

$$\alpha_Z \coloneqq (g \otimes f)(H_N \otimes \mathbb{1}_M | \mathbb{1}_N \otimes H_M^t).$$

Similarly, α_X can be defined as a loop-type operator around $S \times T$, where

$$\alpha_X \coloneqq (f \otimes g)(\mathbb{1}_M \otimes H_N | H_M^t \otimes \mathbb{1}_N)$$

is also a logical operator.

To show that these are in the span of the stabilizers, note that the hybrid stabilizers are given by

$$(H_N \otimes \mathbb{1}_M | \mathbb{1}_N \otimes H_M^t)_Z + (H_A \otimes \mathbb{1}_B | \mathbb{1}_A \otimes H_B^t)_Z \leftrightarrow (\mathbb{1}_M \otimes H_N | H_M^t \otimes \mathbb{1}_N)_X + (\mathbb{1}_B \otimes H_A | H_B^t \otimes \mathbb{1}_A)_X.$$

Thus, the operator

$$(g \otimes f)(H_N \otimes \mathbb{1}_M | \mathbb{1}_N \otimes H_M^t)_Z \leftrightarrow (f \otimes g)(\mathbb{1}_M \otimes H_N | H_M^t \otimes \mathbb{1}_N)_X$$

maps the loop-type logical Z operator to the loop-type logical X operator.

Since this result is true for arbitrary f and g, any operator in the space can be mapped between one puncture and the other. ∎

The logical X operator for the wormhole are products of chain-type operators. The form of these operators are given in Theorem 12.

Lemma 17.—For every loop-type logical Z operator L_Z around $T \times S$ and the unique loop-type logical X operator L_X on $S \times T$ corresponding to L_Z, let the conjugate chain-type operators be Q_X and Q_Z, respectively. The product $Q_X Q_Z$ is the conjugate logical operator to the operator L_Z.

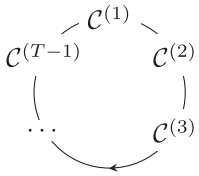
Proof.—Following the proof of Theorem 12, the logical chain-type operators evidently commute with the X and Z stabilizers and are not spanned by them.

From the symmetry of the construction, any overlap with the two-qubit measurement operators always occurs in pairs if at all. Therefore, these operators commute with the two-qubit stabilizers.

Furthermore, since the two-qubit measurement operators are supported only on the boundary, it cannot span the logical chain-type operators. Since the chain-type operators anticommute with the logical loop-type operators, it cannot be expressed as a product of stabilizers alone. ∎

V. CODE DEFORMATION

Having described how to create defects, we can now proceed to discuss how to use them. Logical transformations are effected using a technique called code deformation (see, for instance, Refs. [47,55,59,60]). The core idea behind this technique is to perform a sequence of $T-1$ elementary transformations of a code $\mathcal{C} =: \mathcal{C}^{(1)}$, obtaining codes $\mathcal{C}^{(2)}, \ldots, \mathcal{C}^{(T-1)}, \mathcal{C}^{(T)}$ in the process:



Each elementary transformation is comprised of measurements of Pauli operators. As shown in the schematic, the overall result is to leave the code space globally unchanged, i.e., $\mathcal{C}^{(1)} = \mathcal{C}^{(T)} = \mathcal{C}$, but the logical operators of the code may, and hopefully will, undergo a nontrivial transformation. Since this transformation maps all Pauli operators to Pauli operators, the resulting operation must be a logical Clifford operation.

We begin by reviewing code deformation to highlight some useful properties. Rather than focus right away on hypergraph product codes, we step back and study code deformation as it applies to general quantum codes. Our intent is to track the transformation of the logical operators.

A. Nonmixing

For all $t \in \{1, \ldots, T\}$, the quantum error correcting code $\mathcal{C}^{(t)}$ is defined on n qubits for some fixed n. At step t, $\mathcal{C}^{(t)}$ is the eigenspace of the stabilizer group $\mathcal{S}^{(t)}$.

The logical operators of the code are denoted $\mathcal{L}^{(t)}$. These are the objects that we wish to track as we transform the code. Let $\mathcal{Z} \coloneqq \mathcal{S}^{(t)} \cap \mathcal{S}^{(t+1)}$ be the operators that are common to both $\mathcal{S}^{(t)}$ and $\mathcal{S}^{(t+1)}$.

The following lemma states that, when we transition from $\mathcal{C}^{(t)}$ to $\mathcal{C}^{(t+1)}$, the logical operators that need to be updated are either removed entirely or mapped by multiplying by a stabilizer element.

Lemma 18.—If a logical operator $L \in \mathcal{L}^{(t)}$ anticommutes with the measurement of $S^{(t+1)} \in \mathcal{S}^{(t+1)}/\mathcal{Z}$, then either

(1) L is moved to the space of errors; or
(2) there exists a unique $S' \in \mathcal{S}^{(t)} \backslash \mathcal{Z}$ such that $L \mapsto LS'$.

The proof of this lemma can be found in Supplemental Material [54], Sec. F.

In addition, the sets $\mathcal{S}^{(t)}$ and $\mathcal{L}^{(t)}$ can also exchange operators. The operators in $\mathcal{S}^{(t)} \backslash \mathcal{S}^{(t+1)}$ that commute with all of $\mathcal{S}^{(t+1)}$ are transformed into logical operators. For instance, this transformation happens when we create a new logical qubit in the surface code by forming a puncture. Recall that stabilizer generators and errors can be partitioned into pairs such that a stabilizer generator S and error E anticommute only with each other and commute with all other operators. When a stabilizer S is transformed into a logical operator, the conjugate error E becomes the unique

conjugate error. If there is no unique conjugate error, then this space cannot be used to store a qubit. For instance, this result happens when we have a smooth puncture on a lattice with only rough boundaries. The string of X from the smooth boundary of the puncture cannot be terminated on the boundary. Thus, creating a smooth puncture on a lattice with only rough boundaries cannot be used to store a qubit (because there are redundant checks).

The operators in $\mathcal{L}^{(t)}$ that are in the span of $\mathcal{S}^{(t+1)}\backslash\mathcal{S}^{(t)}$ are removed from the stabilizer group.

This analysis proves that logical operators transform linearly as summarized by the following lemma.

Lemma 19.—For $t \in \{1, \ldots, T\}$, let $\mathcal{L}^{(t)}$ denote the set of logical operators of the code $\mathcal{C}^{(t)}$. Let $\mathbb{L}^{(t)} \in \mathbb{F}_2^{k\times n}$ and $\mathbb{S}^{(t)} \in \mathbb{F}_2^{(n-k)\times n}$ be the generator matrix for this space. There exists a matrix $Q^{(t)}$ such that we can write the logical operators $\mathbb{L}^{(t)}$ as

$$\mathbb{L}^{(t+1)} = Q^{(t)} \begin{pmatrix} \mathbb{L}^{(t)} \\ \mathbb{S}^{(t)} \end{pmatrix}. \tag{10}$$

As we proceed with code deformation, we encounter problems unique to codes that carry several logical qubits. We define below the notions of nonmixing and small transformations as guidelines for studying such transformations.

First, there may potentially be several ways of updating the logical operators, because there is no preferred basis for us to express the logical operators in the intermediary steps. Equivalently, the matrix $Q^{(t)}$ is not unique, as there could be several different ways of expressing the logical operators over the course of code deformation. However, the global transformation $Q = Q^{(T)}Q^{(T-1)}\ldots Q^{(2)}Q^{(1)}$ generated by the entire sequence of code deformation is unique if we choose the same logical operator basis for $Q^{(1)} = Q^{(T)}$.

For $t \in \{1, \ldots, T\}$, let $\mathcal{L}^{(t)}$ denote the set of logical operators of the code $\mathcal{C}^{(t)}$. Let

$$\langle L_j^{(t)}\rangle_j \coloneqq \langle L_g^{(t)}\rangle_g \times \langle L_b^{(t)}\rangle_b$$

be a partition of the set of logical operators $\mathcal{L}^{(t)}$ into good and bad operators. These sets are defined such that the operators in the set g all have weight above some threshold, whereas those in b have weight below this threshold. Furthermore, assume that $\langle L_g^{(t)}\rangle$ contains $k' < k$ independent operators. The set of qubits defined by $\{L_b^{(t)}\}_b$ is considered as gauge qubits. We wish to avoid the space of gauge qubits interacting with our logical qubits and, to this end, define the nonmixing condition.

Definition 20 (nonmixing).—We say that code deformation is *nonmixing* with respect to the partition if there exists a direct-sum decomposition

$$Q^{(t)} = Q_g^{(t)} \oplus Q_b^{(t)},$$

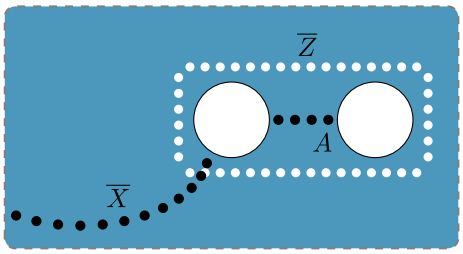


FIG. 8. A pair of punctures used to encode a single logical qubit on the surface code.

where matrices $Q_g^{(t)}$ and $Q_b^{(t)}$ act only on the spaces $\langle L_g^{(t)}\rangle_g$ and $\langle L_b^{(t)}\rangle_b$, respectively. Each elementary step in code deformation is small with respect to $\mathcal{Q}_g^{(t)}$ if $\mathcal{Q}_g^{(t)}$ is rank k' over the good partition.

By guaranteeing that an operation is nonmixing with respect to this partition, we can show that the gauge qubits do not affect the logical qubits. The constraint on the rank guarantees that none of the good logical operators are mapped to either the stabilizers or gauge operators over the course of code deformation.

The nonmixing condition is important, because we cannot guarantee that the number of logical operators remains a constant over the course of code deformation. In general, hypergraph product codes need not be translation invariant like the toric code; even if we maintain a puncture of a fixed radius, the number of logical operators created by this puncture could change as it moves. If we move a puncture by enlarging it and then shrinking it, this change could also change the number of logical operators supported by the puncture. However, the two conditions on the high-weight operators regulate their transformation.

To illustrate, we consider encoding logical qubits on the surface in a slightly unusual way. Consider the pair of punctures on the surface code shown in Fig. 8. This pair shall be treated as a single entity that encodes a logical qubit. The logical Z operator is the loop encircling the pair, denoted $\bar{Z}$. The logical X operator is a string of X's running from the boundary of a puncture to the boundary of the lattice, denoted $\bar{X}$. The operator A running between punctures is a low-weight string of X's, and this logical operator is a potential liability. We therefore treat it as a gauge qubit. When encoding information, we store only logical information using the qubit defined by $\bar{Z}$ and $\bar{X}$. So long as we do not braid using A or drag another defect between these two punctures, this troublesome chain does not cause a problem. In this way, the operation is nonmixing, because the operator A is never entangled with other qubits of interest.

B. Code deformation on the hypergraph product code

We now return to hypergraph product codes and in this section shall discuss how to perform Clifford gates with the

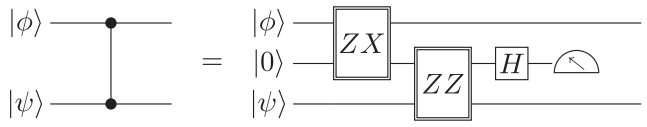


FIG. 9. A measurement-based circuit to perform controlled-Z. We introduce an ancilla prepared in the $|0\rangle$ state. The double boxes indicate a nondestructive projective measurement. The labels PQ on these measurements indicate that the projection is performed along the $+1$ and -1 eigenstates of the two-qubit Pauli operator PQ. Finally, we perform a Hadamard and destructively measure the ancilla qubit in the computational basis.

help of an ancilla. We begin by recalling Lemma 1 from Ref. [58]. It states that we can perform all Clifford gates on a qubit of interest, labeled 1, with the help on an ancillary qubit, labeled a, as follows.

Lemma 21.—Let A and B be distinct, nontrivial single-qubit Pauli operators. Let S and T be two Pauli operators, not necessarily distinct. The two-qubit measurements A_1S_a and B_1T_a, together with all single-qubit Pauli measurements on qubit a, are sufficient to generate the single-qubit Clifford group on qubit 1.

First, we point out that regardless of whether the qubit 1 is an embedded logical qubit, or merely another wormhole, the ancilla qubit(s) shall be encoded in a wormhole. Second, we require these wormholes to encode Y resource states. For reasons that become clear shortly, we find it difficult to perform single-qubit Y measurements on the logical level. If, however, we are provided ancilla qubits that are prepared in the Y state, we may use these objects catalytically to perform a Y measurement. This measurement is necessary, at least as per Lemma 21, to complete the Clifford group.

In addition to these single-qubit gates, we also require entangling gates between multiple logical qubits. The circuit shown in Fig. 9 illustrates how this entanglement, too, can be accomplished on the logical level with the help of an ancilla and Pauli measurements. The requirements to perform Clifford gates are complete. In the next subsection, we study how exactly to perform these measurements.

C. Measurements and traceability

The measurements of Pauli operators described above have to be performed on the logical operator and the ancilla qubit encoded in a wormhole. In addition, we use a regular puncture-based qubit to perform the measurement. This puncture is referred to as a needle.

We are interested in logical operators of the needle that can be measured fault tolerantly. In turn, these operators are used to measure the logical Pauli operators of a wormhole or even an embedded logical qubit. For instance, suppose we have a smooth puncture that has high-weight X and Z operators. The loop-type operator can be measured by shrinking the size of the puncture while simultaneously maintaining the size of the chain-type conjugate logical operator. This result, of course, makes the logical qubit susceptible to logical Z error. However, this result does not affect the measurement outcome so long as the logical X operator remains high weight. Similarly, the chain-type logical X operator can be measured fault tolerantly by shrinking its size while maintaining the size of the loop-type Z logical operator. Unfortunately, the logical Y operator of a puncture cannot be measured fault tolerantly, as this measurement would require that we minimize both the size of the chain-type operator as well as that of the loop-type operator. The logical qubit would then become unprotected and the measurement outcome error prone. The impossibility to fault-tolerantly measure Y operators causes some problems as we see below.

Let w denote a logical qubit or sets of logical qubits whose state we wish to measure, which could refer to a set of logicals on the wormhole, or an embedded logical, or some combination thereof. Let P_p refer to a logical operator of the puncture that is fault-tolerantly measurable. A logical operator Q_w on system w is said to be *traceable* if there is a unitary operation U implementable by code deformation such that

$$U^\dagger P_p U = Q_w P_p.$$

Such operations are used to measure traceable operators Q_w in order to effect measurements of logical operators in Lemma 21. The operator Q_w is measured using the standard ancilla-assisted way: We prepare the ancilla in an eigenstate of P_p, applying the unitary U, and then measure the operator P_p.

If Q_w is either X or Z, then we need to find an operator P_p and an operation U such that

$$U^\dagger P_p U \to Q_w P_p.$$

On the other hand, if $Q_w = Y$, then the situation is complicated for reasons we discuss shortly. In such a case, we assume that we are given access to an another system b prepared in a Y state. The aim is to perform the operation

$$U^\dagger P_p U \to Y_w Y_b P_p,$$

which is not possible without the system b. If we now use this operation to perform a Y_wY_b measurement, then we can do so without affecting the state of system b. In this way, the system b serves as a catalyst and can be used for the next Y measurement on w. In the next subsection, we discuss how to obtain such a resource states to serve as catalysts.

We now make some remarks about traceability, but, before doing so, we remark that the requirements to perform Clifford gates on the system w are complete.

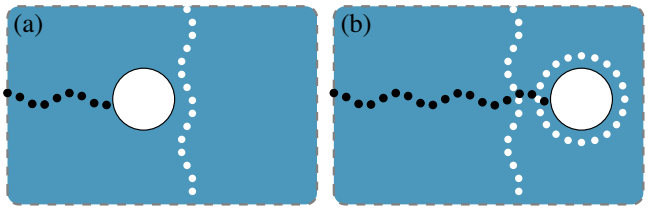


FIG. 10. A puncture crossing its own path. The puncture first leaves a trail of Z's and passes through a wormhole. Upon crossing Z, the logical X is mapped to XZ. This operator is no longer guaranteed to be needle measurable.

We begin by noting that the advantage of the wormhole in this process is that it permits both the X- and Z-type logical operators associated to a defect to be traced. This advantage has been via example in our companion paper [58]; whether or not a logical operator is traceable on a specific code is a code-dependent question. The other advantage of a wormhole is that it permits the use of a Y resource state whose role is catalytic. Without the wormhole, the Y resource states are consumed, and we would therefore require a constant supply of these states.

In the case of the surface code, the new wormhole defects that we introduce make it possible to trace both the X- and Z-type logicals associated to a wormhole [58]. Suppose that P_p above is the logical X operator of a smooth puncture. As it traces the support of a logical $Y = iXZ$ operator, it encounters the location when the X and Z logical operators cross. As shown in Fig. 10, the trailing chain-type X operator is mapped to the logical Y operator, and this mapping has the effect of breaking the original protocol. So, instead of mapping X_p to $Y_w X_p$ as desired, we are mapping it to $Y_w Y_p$. Completing the protocol requires measuring Y_p, but this measurement cannot be done because it is not fault-tolerantly measurable.

Not all is lost, however; we may find that *products* of logical Y operators are traceable. As an example, this result is demonstrated in the case of the surface code with wormhole defects [58], so the framework we describe is sufficiently rich to enable the complete Clifford group, in principle.

We would like to highlight that we are not providing a constructive approach to compiling specific logical operators. Compiling the operation required to trace operators depends not only on the code, but also on the representation of the logical we are interested in performing. The process described merely provides a framework within which to search for such an operation. For instance, given a specific code, we could search for nontrivial Clifford operators that we can perform using brute force. Once a set of Clifford operations is found, these can be used as a basis to compose Clifford gates of interest using standard compiling tools.

We consider an example, perhaps perverse, to illustrate that simply having a wormhole and a puncture is insufficient to generate all Clifford gates. Suppose we have two copies of the toric code that are disconnected from each other. If we initialize a wormhole and a needle on one of the two codes, then clearly this process is insufficient to perform all gates fault tolerantly on all logical qubits. This result is because there is clearly no way for the needle to move to the second code. This result suggests that, in general, the ability to perform all gates may be closely tied to graph connectivity. We return to this idea in Sec. V E.

D. Resource states

Clifford gates by themselves generate only a finite group [49]. Furthermore, the techniques that we discuss above map only Pauli operators to Pauli operators. Therefore, they can at best generate logical Clifford operations.

As is well known, it is sufficient to add any gate that is not already in the Clifford group to achieve a universal gate set. One such non-Clifford gate is the T gate, defined as $T = e^{i\pi Z/8}$. We assume that we have access to physical T gates and wish to construct a logical T gate. The technique presented here means that this logical T gate is not inherently fault tolerant. In turn, the T gate is noisy and, therefore, needs to be distilled [61].

By definition, the logical $T = e^{i\pi Z/8}$ gate can be executed on the support of the logical Z operator. Suppose we have a logical qubit encoded in a smooth puncture prepared in the $|+\rangle$ state. The logical Z operator is a loop-type operator that is supported only on the boundary of the puncture. We can inject the T gate onto the code by performing an explicit circuit on this boundary. One might hope that the stabilizers and logicals on the boundary obey some symmetry such as triorthogonality [11]. In that case, the circuit to inject the T gate could be made fault tolerant, as we merely require transversal physical T gates in order to implement the transversal logical T gate. However, we do not assume that the puncture obeys these symmetries, and, thus, the circuit to perform the T is not fault tolerant. Thus, we want to minimize the size of the circuit to minimize the number of faulty locations. To this end, we may shrink the puncture, i.e., reduce the size of the boundary that supports the logical qubit. Upon performing the T circuit, we increase the size of the puncture again to make it resistant to logical Z-type errors. While doing so, of course, the logical qubit is still subject to logical Z errors and is, thus, subject to dephasing errors. The end result is a noisy version of the T state defined as $|A\rangle = T|+\rangle$.

Once we have several such logical qubits carrying potentially noisy T states, we can perform state distillation on the hypergraph product code. State distillation is a technique that uses several noisy T states and produces fewer, but higher-fidelity, copies of the T state. These higher-fidelity copies can then be used in the computation if they are sufficiently reliable. State distillation requires only Clifford gates and Pauli measurements, and these are operations we already have the ingredients to perform.

To perform a logical T on an embedded logical qubit, we can use the T gate and a single-qubit teleportation circuit [50].

In addition to using resource states to inject the T gate, we also require resource states that are prepared in the Y basis as discussed in the previous section. However, since these resource states are used catalytically, they can be prepared once before the beginning of the computation. To prepare these states, we follow a procedure similar to the preparation of a T state. We perform the circuit required to prepare a puncture qubit in a logical Y state. We note that the circuit to prepare the Y state has to be performed on the support of both the X and Z logical operators. To minimize the size of this circuit, we therefore reduce the size of both the loop-type and the chain-type logical operators. This circuit likely is not fault tolerant; after performing the circuit, we have to increase the size of the loop-type and chain-type logical operators. During this period, the logical qubit supported on the puncture may be subject to depolarizing noise.

Once we prepare several such logical qubits, we have access to several noisy Y resource states. To purify them, we have to perform distillation. Of course, there may be more optimal ways to prepare these states, but this preparation is sufficient to generate the desired resource states.

E. Pointlike punctures

Before we conclude, we consider a scheme that involves the movement of pointlike punctures. This scheme deviates slightly from the framework that we discuss above, and, since the punctures are pointlike, the setup is not fault tolerant. However, we feel that it still may help understand movement in these codes.

When discussing the surface code, traceability is relatively simple. We can move a rough puncture around a smooth puncture and thereby perform a nontrivial Clifford operation. Whether or not such an operation is possible is dictated by topology; a rough puncture can trace any closed loop of Z operators. The situation is not so simple in the case of hypergraph product codes.

In this subsection, we discuss when logical operators are traceable using a pointlike puncture to serve as the needle. Of course, using a pointlike puncture is not fault tolerant, as the loop-type logical operators are low weight and, therefore, error prone. This discussion, however, sheds light on when something nontrivial is possible. It also illustrates that we can generalize braiding to graph-theoretic concepts.

We show that, in the case of pointlike punctures, traceability can be cast as walks on a graph. Thus, whether or not a logical operator is traceable boils down to verifying whether a certain path on a graph exists. This result shows that it may be possible to efficiently verify when an operator is traceable.

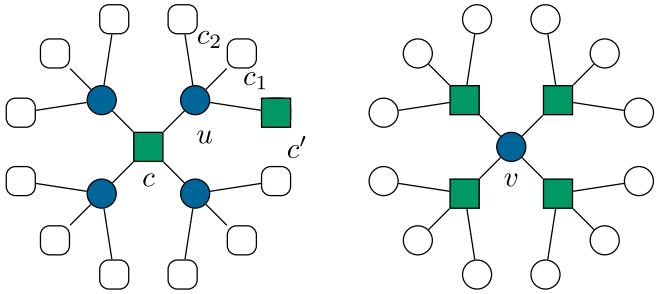


FIG. 11. A pointlike puncture centered at c, v. The check c' is connected to c via the variable node u.

Consider a pointlike puncture, i.e., one created by removing a single stabilizer generator. For the sake of illustration, this puncture is smooth. Code deformation entails that there exist a series of steps such that the pointlike puncture corresponds to $T_j \times S_j$ for $j = 1, \ldots, N$.

Let $T_j = \{c_j\}$ and $S_j = \{v_j\}$ be singleton sets and let $T_j \times S_j$ be the associated pointlike puncture. The logical Z operator α that emerges is then just the support of the Z stabilizer (c_j, v_j), i.e., $\alpha_j \coloneqq \Gamma(c_j) \times v_j \cup c_j \times \Gamma(v_j)$. Let the conjugate logical operator be β_j. Let us study how these objects transform as we transition from step j to step $j+1$.

1. Moving a single step

Suppose we consider moving this puncture by changing $T_j \to T_{j+1} = \{c_{j+1}\}$, where c_j and c_{j+1} share a bit u_j in their common neighborhood as shown in Fig. 11.

In the growth phase, we measure X on the qubit (u, v_j). This result anticommutes with all the Z stabilizers that are incident to (u, v_j), i.e., (c_1, v_j), (c_2, v_j), and (c_{j+1}, v_j). The logical Z operator α_j [corresponding to (c_j, v_j)] also anticommutes with this operation. These objects are updated per the stabilizer update rule. We multiply the operators (c_1, v_j), (c_2, v_j), and α_j by (c_{j+1}, v_j). The stabilizer (c_{j+1}, v_j) is itself removed from the stabilizer group.

In the contraction phase, we measure the stabilizer (c_j, v_j), returning it to the stabilizer group. This result anticommutes with the measurement $X(u, v_j)$. This result also anticommutes with the logical operator β_j. To resolve this anticommutation relation, we map $\beta_j \to \beta_{j+1} \coloneqq \beta_j X(u, v)$. We then discard $X(u, v)$ from the stabilizer group.

Thus, the logical operator β_j grows by a single qubit. However, we have not yet returned to the code space. The logical operator β_{j+1} still anticommutes with the operators (c_1, v_j) and (c_2, v_j). These objects have still not been returned to the stabilizer group. We highlight this matter, because it is this issue that does not allow us to fit the movement of a pointlike puncture into the framework described in the previous sections.

There are two concerns associated with these frustrated stabilizers:

(1) Will they return to the stabilizer group?
(2) Will the weight of these objects grow or remain upper bounded by some constant?

First, since the path we traverse corresponds to a logical, it must commute with all the stabilizers. Thus, at some point during the course of the pointlike puncture moving, it commutes with each stabilizer that it frustrated. Exactly when these operators are returned to the stabilizer group depends on the path being traversed.

Second, the weight of these stabilizers is always guaranteed to be upper bounded by a constant. In fact, these frustrated operators are always pairwise products of the puncture at step j and themselves. For instance, consider the example above where we transition by one step, from j to $j+1$. In the very next step, suppose the next point to be removed corresponds to the stabilizer (c_{j+2}, v_j). The stabilizer (c_{j+1}, v_j) is returned to the stabilizer group. Therefore, the frustrated stabilizers (c_1, v_j) and (c_2, v_j) are now multiplied by (c_{j+2}, v_j). This process continues until we measure another qubit in the support of (c_1, v_j) and (c_2, v_j), at which point they return to the stabilizer group.

For a pointlike puncture, this analysis shows that the logical can grow one qubit at a time. With this insight, we can cast the problem of whether or not a logical is traceable as a graph problem. In this problem, we first consider a logical operator, say, Q, that has no Y operators in its support. We use this operator to define a graph $\mathcal{G}_Q$ as follows. The stabilizers that are adjacent to the qubits become the vertices of $\mathcal{G}_Q$, and the qubits in the support of Q become the edges of $\mathcal{G}_Q$. If there exists a sequence of stabilizers that we can puncture, each connected by a single qubit, then this path becomes traceable.

In particular, this result can be cast as an Eulerian cycle [62]. The Eulerian cycle is an efficient algorithm that can be stated as follows: It is well known that a Eulerian cycle exists only when the degree of each vertex in the graph is even. Thus, given a graph with n vertices, the existence of an Eulerian cycle can be verified efficiently. This example shows that braiding may generalize to a purely graph-theoretic concept. Moreover, there may exist an efficient algorithm to answer when a logical is traceable.

Algorithm 1. *(Eulerian cycle).*—

1. **Input:** Graph $\mathcal{G} = (V, E)$.
2. **Output:** A path on the graph such that every edge is traversed exactly once if it exists.

VI. CONCLUSIONS

We have provided a general framework to implement Clifford gates on hypergraph product codes. This framework is based on code deformation and generalizes defect-based encoding from topological codes. In particular, we generalize wormhole defects introduced in a companion paper [58]. These defects ensure that the code remains LDPC at each step of code deformation.

In contrast to a previous scheme suggested by Gottesman, these operations are defined on a single block. The generalized punctures that we obtain are capable of encoding several logical qubits. We discussed a framework that is rich enough to permit all Clifford gates on encoded qubits. Whether a particular code permits these gates is a code-dependent question. Finally, we discussed the movement of pointlike charges on these graphs. These defects serve to illustrate that something nontrivial can be accomplished on hypergraph product codes. Furthermore, they demonstrate how braiding can be generalized to a purely graph-theoretic notion.

Of course, this scheme is merely a proof of concept, and there is a lot of work to be done in the future. In no particular order, we discuss some issues that need to be addressed; this list is by no means complete. Using pointlike punctures helps us understand that traceability in certain instances is connected to an Eulerian cycle. As punctures become larger, however, it is unclear how this algorithm will generalize. Furthermore, we would like efficient algorithms that can verify whether a given representation of a logical is traceable. Once these punctures have been introduced, under what conditions can we decode? Over the course of code deformation, how does small-set flip need to be modified to work for a non-CSS code?

We believe that addressing these questions will be intimately connected to the specific code we wish to use. It is, of course, important to consider specific classes of codes to understand which Clifford gates we can generate using this process. At this juncture, however, this consideration seems premature, as there is as yet no consensus as to which hypergraph product codes offer the best performance. As the theory progresses, this will likely be informed by decoding algorithms, but perhaps the ability to perform Clifford gates could also factor into this choice. The punctures may exhibit symmetries that do not require state distillation to prepare resource states. Since these codes are no longer local, it is unclear what sorts of gates can be implemented transversally and which cannot.

Addressing these questions will help establish the role of hypergraph product codes in quantum computation.

ACKNOWLEDGMENTS

A. K. thanks Christophe Vuillot, Daniel Gottesman, Vivien Londe, Leonid Pryadko, and Nicolas Delfosse for discussions. A. K. also thanks Thom Bohdanowicz and Anthony Leverrier for pointing out errors in an earlier draft. A. K. acknowledges the support of the Fonds de Recherche—Nature et Technologie (FRQNT) for the B2X scholarship. This work was partially funded by Canada's NRC and NSERC. D. P. is a CIFAR Fellow in the Quantum Information Science program for support.

[1] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, *Quantum Supremacy Using a Programmable Superconducting Processor*, Nature (London) **574**, 505 (2019).

[2] D. Aharonov and M. Ben-Or, *Fault-Tolerant Quantum Computation with Constant Error*, in *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (Association for Computing Machinery, New York, 1997), pp. 176–188.

[3] P. Aliferis, D. Gottesman, and J. Preskill, *Quantum Accuracy Threshold for Concatenated Distance-3 Codes*, Quantum Inf. Comput. **6**, 97 (2006), https://dl.acm.org/doi/10.5555/2011665.2011666.

[4] A. Y. Kitaev, *Quantum Computations: Algorithms and Error Correction*, Russ. Math. Surv. **52**, 1191 (1997).

[5] E. Knill, R. Laflamme, and W. H. Zurek, *Resilient Quantum Computation: Error Models and Thresholds*, Proc. R. Soc. A **454**, 365 (1998).

[6] N. Delfosse, P. Iyer, and D. Poulin, *Generalized Surface Codes and Packing of Logical Qubits*, arXiv:1606.07116.

[7] N. P. Breuckmann, C. Vuillot, E. Campbell, A. Krishna, and B. M. Terhal, *Hyperbolic and Semi-hyperbolic Surface Codes for Quantum Storage*, Quantum Sci. Technol. **2**, 035007 (2017).

[8] J. Conrad, C. Chamberland, N. P. Breuckmann, and B. M. Terhal, *The Small Stellated Dodecahedron Code and Friends*, Phil. Trans. R. Soc. A **376**, 20170323 (2018).

[9] M. Suchara, A. Faruque, C.-Y. Lai, G. Paz, F. T. Chong, and J. Kubiatowicz, *Comparing the Overhead of Topological and Concatenated Quantum Error Correction*, arXiv:1312.2316.

[10] A. G. Fowler and S. J. Devitt, *A Bridge to Lower Overhead Quantum Computation*, arXiv:1209.0510.

[11] S. Bravyi and J. Haah, *Magic-State Distillation with Low Overhead*, Phys. Rev. A **86**, 052329 (2012).

[12] A. Y. Kitaev, *Fault-Tolerant Quantum Computation by Anyons*, Ann. Phys. (Amsterdam) **303**, 2 (2003).

[13] S. B. Bravyi and A. Y. Kitaev, *Quantum Codes on a Lattice with Boundary*, arXiv:quant-ph/9811052.

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

[15] R. Raussendorf and J. Harrington, *Fault-Tolerant Quantum Computation with High Threshold in Two Dimensions*, Phys. Rev. Lett. **98**, 190504 (2007).

[16] R. Raussendorf, J. Harrington, and K. Goyal, *A Fault-Tolerant One-Way Quantum Computer*, Ann. Phys. (Amsterdam) **321**, 2242 (2006).

[17] R. Raussendorf, J. Harrington, and K. Goyal, *Topological Fault-Tolerance in Cluster State Quantum Computation*, New J. Phys. **9**, 199 (2007).

[18] 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).

[19] A. A. Kovalev and L. P. Pryadko, *Fault Tolerance of Quantum Low-Density Parity Check Codes with Sublinear Distance Scaling*, Phys. Rev. A **87**, 020304(R) (2013).

[20] D. Gottesman, *Fault-Tolerant Quantum Computation with Constant Overhead*, Quantum Inf. Comput. **14**, 1338 (2014), https://dl.acm.org/doi/10.5555/2685179.2685184.

[21] S. Bravyi and B. Terhal, *A No-Go Theorem for a Two-Dimensional Self-Correcting Quantum Memory Based on Stabilizer Codes*, New J. Phys. **11**, 043029 (2009).

[22] S. Bravyi, D. Poulin, and B. Terhal, *Tradeoffs for Reliable Quantum Information Storage in 2D Systems*, Phys. Rev. Lett. **104**, 050503 (2010).

[23] N. H. Nickerson, Y. Li, and S. C. Benjamin, *Topological Quantum Computing with a Very Noisy Network and Local Error Rates Approaching One Percent*, Nat. Commun. **4**, 1756 (2013).

[24] P. Campagne-Ibarcq, E. Zalys-Geller, A. Narla, S. Shankar, P. Reinhold, L. Burkhart, C. Axline, W. Pfaff, L. Frunzio, R. J. Schoelkopf, and M. H. Devoret, *Deterministic Remote Entanglement of Superconducting Circuits through Microwave Two-Photon Transitions*, Phys. Rev. Lett. **120**, 200501 (2018).

[25] C. J. Axline, L. D. Burkhart, W. Pfaff, M. Zhang, K. Chou, P. Campagne-Ibarcq, P. Reinhold, L. Frunzio, S. Girvin, L. Jiang *et al.*, *On-Demand Quantum State Transfer and Entanglement between Remote Microwave Cavity Memories*, Nat. Phys. **14**, 705 (2018).

[26] P. Kurpiers, P. Magnard, T. Walter, B. Royer, M. Pechal, J. Heinsoo, Y. Salathé, A. Akin, S. Storz, J.-C. Besse *et al.*, *Deterministic Quantum State Transfer and Remote Entanglement Using Microwave Photons*, Nature (London) **558**, 264 (2018).

[27] J.-P. Tillich and G. Zémor, *Quantum LDPC Codes with Positive Rate and Minimum Distance Proportional to the Square Root of the Blocklength*, IEEE Trans. Inf. Theory **60**, 1193 (2014).

[28] A. Leverrier, J.-P. Tillich, and G. Zémor, *Quantum Expander Codes*, in *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)* (IEEE, New York, 2015), pp. 810–824.

[29] O. Fawzi, A. Grospellier, and A. Leverrier, *Efficient Decoding of Random Errors for Quantum Expander Codes*, in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing* (Association for Computing Machinery, New York, 2018), pp. 521–534.

[30] O. Fawzi, A. Grospellier, and A. Leverrier, *Constant Overhead Quantum Fault-Tolerance with Quantum Expander Codes*, arXiv:1808.03821.

[31] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, *Topological Quantum Memory*, J. Math. Phys. (N.Y.) **43**, 4452 (2002).

[32] T. Rudolph, *Why I Am Optimistic about the Silicon-Photonic Route to Quantum Computing*, APL Photonics **2**, 030901 (2017).
[33] M. Gimeno-Segovia, P. Shadbolt, D. E. Browne, and T. Rudolph, *From Three-Photon Greenberger-Horne-Zeilinger States to Ballistic Universal Quantum Computation*, Phys. Rev. Lett. **115**, 020502 (2015).
[34] M. Gimeno-Segovia, *Towards Practical Linear Optical Quantum Computing*, Ph.D. Thesis, Imperial College London, 2015.
[35] R. Raussendorf, S. Bravyi, and J. Harrington, *Long-Range Quantum Entanglement in Noisy Cluster States*, Phys. Rev. A **71**, 062313 (2005).
[36] A. Bolt, G. Duclos-Cianci, D. Poulin, and T. Stace, *Foliated Quantum Error-Correcting Codes*, Phys. Rev. Lett. **117**, 070501 (2016).
[37] A. Bolt, D. Poulin, and T. Stace, *Decoding Schemes for Foliated Sparse Quantum Error-Correcting Codes*, Phys. Rev. A **98**, 062302 (2018).
[38] D. J. MacKay, G. Mitchison, and P. L. McFadden, *Sparse-Graph Codes for Quantum Error Correction*, IEEE Trans. Inf. Theory **50**, 2315 (2004).
[39] A. Couvreur, N. Delfosse, and G. Zémor, *A Construction of Quantum LDPC Codes from Cayley Graphs*, IEEE Trans. Inf. Theory **59**, 6087 (2013).
[40] M. H. Freedman, D. A. Meyer, and F. Luo, *Z2-Systolic Freedom and Quantum Codes*, *Mathematics of Quantum Computation* (Chapman and Hall/CRC, London, 2002), pp. 287–320.
[41] L. Guth and A. Lubotzky, *Quantum Error Correcting Codes and 4-Dimensional Arithmetic Hyperbolic Manifolds*, J. Math. Phys. (N.Y.) **55**, 082202 (2014).
[42] V. Londe and A. Leverrier, *Golden Codes: Quantum LDPC Codes Built from Regular Tessellations of Hyperbolic 4-Manifolds*, Quantum Inf. Comput. **19**, 361 (2019), https://dl.acm.org/doi/10.5555/3370251.3370252.
[43] S. Bravyi and M. B. Hastings, *Homological Product Codes*, in *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing* (Association for Computing Machinery, New York, 2014), pp. 273–282.
[44] A. Lavasani, G. Zhu, and M. Barkeshli, *Universal Logical Gates with Constant Overhead: Instantaneous Dehn Twists for Hyperbolic Quantum Codes*, Quantum **3**, 180 (2019).
[45] A. Lavasani and M. Barkeshli, *Low Overhead Clifford Gates from Joint Measurements in Surface, Color, and Hyperbolic Codes*, Phys. Rev. A **98**, 052319 (2018).
[46] 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).
[47] H. Bombin, *Clifford Gates by Code Deformation*, New J. Phys. **13**, 043005 (2011).
[48] T. Jochym-O'Connor, *Fault-Tolerant Gates via Homological Product Codes*, Quantum **3**, 120 (2019).
[49] M. A. Nielsen and I. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, New York,2000).
[50] X. Zhou, D. W. Leung, and I. L. Chuang, *Methodology for Quantum Logic Gate Construction*, Phys. Rev. A **62**, 052316 (2000).
[51] T. Richardson and R. Urbanke, *Modern Coding Theory* (Cambridge University Press, Cambridge, England, 2008).
[52] A. R. Calderbank and P. W. Shor, *Good Quantum Error-Correcting Codes Exist*, Phys. Rev. A **54**, 1098 (1996).
[53] A. Steane, *Multiple-Particle Interference and Quantum Error Correction*, Proc. R. Soc. A **452**, 2551 (1996).
[54] See Supplemental Material at http://link.aps.org/supplemental/10.1103/PhysRevX.11.011023 for proofs of some lemmas appearing in the main text.
[55] C. Vuillot, L. Lao, B. Criger, C. G. Almudever, K. Bertels, and B. Terhal, *Code Deformation and Lattice Surgery Are Gauge Fixing*, New J. Phys. **21**, 033028 (2019).
[56] H. Bombin, *Topological Order with a Twist: Ising Anyons from an Abelian Model*, Phys. Rev. Lett. **105**, 030403 (2010).
[57] T. J. Yoder and I. H. Kim, *The Surface Code with a Twist*, Quantum **1**, 2 (2017).
[58] A. Krishna and D. Poulin, *Topological Wormholes*, Phys. Rev. Research **2**, 023116 (2020).
[59] J. T. Anderson, G. Duclos-Cianci, and D. Poulin, *Fault-Tolerant Conversion between the Steane and Reed-Muller Quantum Codes*, Phys. Rev. Lett. **113**, 080501 (2014).
[60] H. Bombin and M. A. Martin-Delgado, *Quantum Measurements and Gates by Code Deformation*, J. Phys. A **42**, 095302 (2009).
[61] S. Bravyi and A. Kitaev, *Universal Quantum Computation with Ideal Clifford Gates and Noisy Ancillas*, Phys. Rev. A **71**, 022316 (2005).
[62] C. Moore and S. Mertens, *The Nature of Computation* (Oxford University, New York, 2011).