

Fault-Tolerant Logical Clifford Gates from Code Automorphisms

Hasan Sayginel^{1,2,*}, Stergios Koutsoumpas^{1,*}, Mark Webster^{1,*}, Abhishek Rajput³, and Dan E. Browne¹

¹*Department of Physics and Astronomy, University College London, London WC1E 6BT, United Kingdom*

²*National Physical Laboratory, Teddington TW11 0LW, United Kingdom*

³*Department of Computer Science, University College London, London WC1E 6BT, United Kingdom*

 (Received 22 December 2024; revised 2 May 2025; accepted 10 July 2025; published 5 September 2025)

We study the implementation of fault-tolerant logical Clifford gates on stabilizer quantum error-correcting codes based on their symmetries. Our approach is to map the stabilizer code to a binary linear code, compute its automorphism group, and impose constraints based on the Clifford operators permitted. We provide a rigorous formulation of the method for finding automorphisms of stabilizer codes and generalize the ZX dualities previously introduced for Calderbank-Shor-Steane (CSS) codes to non-CSS codes. We provide a PYTHON package implementing our algorithms that uses the computational-algebra MAGMA software system for certain subroutines. Our algorithms map automorphism-group generators to physical circuits, calculate Pauli corrections based on the destabilizers of the code, and determine their logical action. We discuss the fault tolerance of the circuits and include examples of gates through automorphisms for the $[[4, 2, 2]]$ and perfect $[[5, 1, 3]]$ codes, bivariate bicycle codes, and the best-known distance codes.

DOI: [10.1103/vf7v-cpq9](https://doi.org/10.1103/vf7v-cpq9)

I. INTRODUCTION

Major obstacles to the construction of reliable large-scale quantum computers are the noise and errors in the qubits, gates, and measurements that make up their implementation. There has thus been significant research interest in the development of fault-tolerant quantum computation techniques that would allow reliable computation in the presence of hardware errors.

Quantum error correction is one of the most promising approaches toward fault-tolerant quantum computation, where errors are detected and corrected by encoding the information across multiple physical qubits.

Protecting the encoded information is not enough, however, and it is crucial that our codes allow for efficient implementation of logical operators on the encoded information for the algorithm that we aim to perform. In particular, low depth and low-resource-overhead implementations of logical gates are highly desirable.

Transversal logical operators are an example of fault-tolerant gates [1]. In transversal logical gates, no physical

operation acts upon more than one qubit in a code block. This therefore avoids multiqubit errors within the code block, which could effectively reduce its distance. Most stabilizer codes only admit a very limited set of transversal logical operators and it has been shown that it is impossible for a code to have a universal set of transversal logical gates [2].

The notion of transversality can be slightly relaxed to SWAP-transversal logical gates, which include both qubit permutations as well as single-qubit Clifford gates. For qubit architectures where physical SWAP operations can be implemented with low overhead, this allows for a wider range of fault-tolerant gates.

SWAP-transversal gates of a stabilizer code can be identified by looking at the automorphisms of a related binary linear code. For binary linear codes, the term *automorphism* denotes a permutation of the bits of a code that map a code to itself. The automorphisms of a code form a group under composition. Calderbank *et al.* [3] have extended the notion of automorphism to quantum codes by showing that quantum codes can be represented as classical codes over $GF(4)$. In this representation, automorphisms of the classical code correspond to physical-qubit permutations and single-qubit Clifford gates that preserve the code space and implement fault-tolerant logical operations.

In Refs. [4–6], automorphisms of the binary linear codes used in the construction of a Calderbank-Shor-Steane (CSS) code have been used to identify *fold-transversal*

*These authors contributed equally to this work.

†Contact author: hasan.sayginel.17@ucl.ac.uk

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

gates, which employ qubit permutations and Hadamard (H) gates on all physical qubits.

In this paper, we bridge the gap between the fold-transversal and $GF(4)$ methods by introducing a family of algorithms for finding fault-tolerant Clifford gate implementations for any stabilizer code based on symmetries of the code. By choosing different binary representations of stabilizer codes, we can control which single-qubit Clifford operations appear in the circuits implementing a logical gate. This generalizes the ZX dualities of Ref. [5] to non-CSS codes and a wider range of logical gates. We develop a rigorous approach for finding Clifford gates through code automorphisms that includes applying Pauli corrections to ensure that the stabilizer group is preserved and identifying logical actions and physical circuits.

We provide a package with an implementation of our algorithms in PYTHON using MAGMA [7] and the open-source BLISS [8] software package for the calculation of the automorphism group of a related classical linear code. The package is available in Ref. [9].

The structure of the paper is as follows. In Sec. II, we introduce the key theoretical concepts outlined above. We then discuss stabilizer codes, logical operators, and fault tolerance. In Sec. II B, we discuss the fault tolerance of the circuits that we obtain by separating them into three different categories. We highlight the close link between the hardware implementation of a SWAP gate and the fault-tolerant properties of our circuits.

In Sec. III, we introduce our algorithms. Our methods work by mapping a stabilizer code to a related binary linear code with constraints on the physical Clifford gates included in the circuits that we want to find. We first show how to obtain the logical operators consisting of physical SWAP and H gates for general stabilizer codes. These arise as a subgroup of the automorphisms of the classical linear code generated by the symplectic representation of the quantum code. In the case of CSS codes, we recover the ZX dualities from Ref. [5] but also find a wider range of operators. Similarly, we can fix any of the other two Pauli axes and obtain circuits consisting of physical SWAP and transversal S or $\sqrt{X}$ gates.

We then extend this method by using a three-block representation that recovers the form of Ref. [3] and allows us to find logical operators consisting of physical SWAP and arbitrary single-qubit Clifford gates. We expand on the method by finding Pauli phase corrections and determining the logical action of the circuit.

We then explain a number of technical points for the methods. In Sec. IV, we show how to apply Pauli corrections to automorphism gates to ensure that the signs of the stabilizer generators are preserved. In Sec. V, we show how to find the logical action of the gates. In Sec. VI, we show how to translate the results of the algorithms to a standard circuit form.

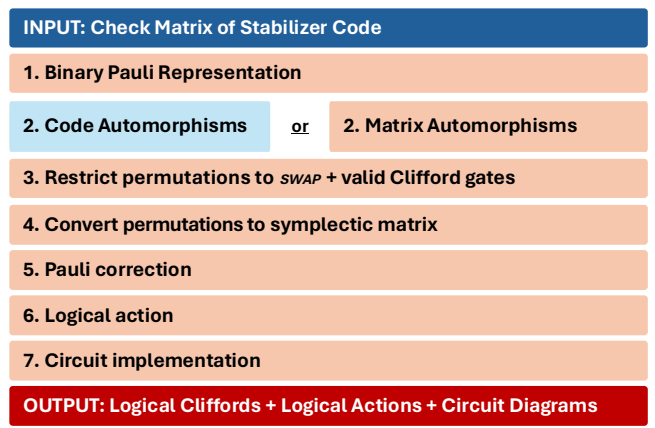


FIG. 1. An outline of our algorithms for finding logical Clifford operators via automorphisms of a related binary linear code. The step outlined in light blue (code automorphisms) is performed using a computational-algebra software package such as MAGMA. All other steps are performed in PYTHON.

In Sec. VII, we show how to use the embedded-code technique of Ref. [10] to map pairs of qubits onto additional columns of the related binary linear code. This allows us to represent physical controlled-NOT (CNOT) and controlled-Z (CZ) gates as automorphisms. We show that different embedding operators allow us to restrict the physical circuits that we find based on additional constraints. This can be used to find a wider range of Clifford logical actions and to restrict CNOT and CZ gates in accordance with the connectivity of the available hardware.

In Sec. VIII, we apply our techniques to various stabilizer codes. We first find fault-tolerant gates for the well-known $[[4, 2, 2]]$ and $[[5, 1, 3]]$ codes. We then analyze the automorphism groups of the best-known-distance single-logical-qubit stabilizer codes for up to 30 physical qubits from Ref. [11]. Finally, we calculate and analyze the automorphism groups of the bivariate bicycle codes of Ref. [12] and find additional automorphism gates compared to that work.

In Sec. IX, we discuss the computational complexity of our algorithms.

We outline our set of algorithms in Fig. 1: step 1 is explained in Sec. II for single-qubit gates and in Sec. VII for two-qubit gates. Steps 2–4 are explained in Sec. III. Step 5 is explained in Sec. IV. Step 6 is explained in Sec. V and step 7 is explained in Sec. VI.

II. BACKGROUND

A. Stabilizer codes

In this section, we introduce the key concepts of stabilizer codes and illustrate these using the well-known five-qubit code as an example. Stabilizer codes are defined by choosing a set $\mathbf{G}$ of *stabilizer generators*. These are a set of independent commuting Pauli operators on n qubits

that generate the *stabilizer group* denoted $\langle \mathbf{G} \rangle$. The *code space* is the subspace of states on n qubits $\mathcal{H}_2^n$ fixed by all elements of $\mathbf{G}$ —i.e., the set $\{|\psi\rangle \in \mathcal{H}_2^n : A|\psi\rangle = |\psi\rangle, \forall A \in \mathbf{G}\}$.

1. Symplectic representation of Pauli operators

In this work, we use various mappings of the stabilizer generators of a quantum code to a binary matrix, which can then be interpreted as the generator matrix of a binary linear code. Pauli operators on n qubits have a *symplectic representation* as binary vectors of length $2n$. Let $\mathbf{x}$ and $\mathbf{z}$ be binary vectors of length n . We define an *unsigned Pauli operator* as

$$X(\mathbf{x})Z(\mathbf{z}) := \prod_{0 \leq i < n} X_i^{\mathbf{x}[i]} \prod_{0 \leq i < n} Z_i^{\mathbf{z}[i]}. \quad (1)$$

We refer to the vector $\mathbf{x}$ as the *X component* and to $\mathbf{z}$ as the *Z component* of the operator. The operators of form $X(\mathbf{x})Z(\mathbf{z})$ generate the Pauli group on n qubits, up to global phases. The single-qubit Pauli operators have the following binary-string representations:

$$X \equiv (1|0); \quad Z \equiv (0|1); \quad Y = iXZ \equiv (1|1).$$

Note that in the case of the Y operator, the representation does not take into account the global phase i . Up to global phases, multiplication of Pauli operators corresponds to addition of the vector representations modulo 2.

We can check whether two Pauli operators commute via the *symplectic inner product* of their vector representations. The Pauli operators $X(\mathbf{x}_1)Z(\mathbf{z}_1)$ and $X(\mathbf{x}_2)Z(\mathbf{z}_2)$ commute if and only if

$$(\mathbf{x}_1, \mathbf{z}_1)\Omega_n(\mathbf{x}_2, \mathbf{z}_2)^T = \mathbf{z}_1 \cdot \mathbf{x}_2 + \mathbf{x}_1 \cdot \mathbf{z}_2 = 0 \pmod{2}, \quad (2)$$

where $\Omega_n := \begin{bmatrix} 0_{n \times n} & I_n \\ I_n & 0_{n \times n} \end{bmatrix}$ is the *binary symplectic form*. We will use Ω going forward rather than Ω_n because n is determined by the number of physical qubits used by the stabilizer code.

2. Logical Pauli operators

Pauli operators that commute with all elements of $\mathbf{G}$ but are not in the stabilizer group $\langle \mathbf{G} \rangle$ are called *nontrivial logical Pauli operators*. A generating set of the nontrivial logical Pauli operators can be determined using the method of Ref. [13, Sec. 10.5.7]. Using Gaussian elimination and permuting the qubits, we can write the check matrix G of a

stabilizer code in the following *standard form*:

$$G = \left[\begin{array}{ccc|cc} I & A_1 & A_2 & B & 0 & C_1 \\ 0 & 0 & 0 & D & I & C_2 \end{array} \right] \begin{array}{l} \} r \\ \} s \\ \} k \end{array} \quad (3)$$

The rank of the check matrix G determines the number of *logical qubits* $k := n - r - s$. We can construct a set of k independent *logical Z* operators and k independent *logical X* operators as follows:

$$L := \begin{bmatrix} L_X \\ L_Z \end{bmatrix} = \left[\begin{array}{ccc|cc} 0 & C_2^T & I & C_1^T & 0 & 0 \\ 0 & 0 & 0 & A_2^T & 0 & I \end{array} \right]. \quad (4)$$

Each row of L_Z anticommutes with the corresponding row of L_X and commutes with all other rows.

Example 1. The stabilizer group of the perfect $[[5, 1, 3]]$ quantum error-correcting code is generated by

$$\mathbf{G} = \langle XZZXI, IXZZX, XIXZZ, ZXIXZ \rangle. \quad (5)$$

The check matrix in symplectic form is

$$G := \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array} \right].$$

Calculating the standard form as in Eq. (3), we obtain

$$G = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix} = \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{array} \right]. \quad (6)$$

Thus its Pauli logical operator basis is

$$\begin{bmatrix} L_X \\ L_Z \end{bmatrix} = \left[\begin{array}{ccccc|ccccc} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{array} \right]. \quad (7)$$

3. Representation of Clifford operators as binary symplectic matrices

Clifford operators are unitaries that map Pauli operators to Pauli operators under conjugation and play a crucial role in quantum error correction. Our algorithms search for logical operators that are composed of physical Clifford gates and the logical action of which can also be described

as a Clifford unitary. The Clifford operators form a group denoted by C_2 , which has the following generators:

$$C_2 = \langle H, S, CZ \rangle,$$

where

$$H := \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad S := \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

$$CZ := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

Other important Clifford gates which we will use below are:

$$\text{CNOT} := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

$$\text{SWAP} := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\sqrt{X} := \frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}$$

$$C(X, X) := \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \end{bmatrix}.$$

Clifford unitaries can be represented as binary symplectic matrices up to a Pauli factor. *Binary symplectic matrices* are binary matrices U that preserve the symplectic form $U\Omega U^T = \Omega$. A binary symplectic matrix acting on the vector representation of a Pauli operator by right matrix multiplication has the same action as the related Clifford operator acting via conjugation. We illustrate this in the following example, which gives the symplectic matrix form of the S , CZ , and CNOT Clifford operators.

Example 2. The symplectic matrix

$$\left[\begin{array}{c|c} 1 & 1 \\ \hline 0 & 1 \end{array} \right]$$

acts on single-qubit Pauli operators as the Clifford operator $S := \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$. It is sufficient to determine the action of S on the basis $\{X, Z\}$ of Pauli operators on one qubit up to global

phases:

$$S : X \equiv (1|0) \mapsto Y \equiv (1|1),$$

$$S : Z \equiv (0|1) \mapsto Z \equiv (0|1).$$

This is equivalent to right multiplication by the matrix

$$\left[\begin{array}{c|c} 1 & 1 \\ \hline 0 & 1 \end{array} \right]$$

as claimed.

Similarly, the symplectic matrix

$$\left[\begin{array}{c|c} 0 & 1 \\ \hline 1 & 0 \end{array} \right]$$

swaps the X and Z components and so acts as an H operator.

The matrix

$$\left[\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right]$$

acts on two-qubit Paulis as an CZ_{01} operator because:

$$CZ_{01} : X_0 \equiv (10|00) \mapsto X_0 Z_1 \equiv (10|01)$$

$$CZ_{01} : X_1 \equiv (01|00) \mapsto X_1 Z_0 \equiv (01|10)$$

$$CZ_{01} : Z_0 \equiv (00|10) \mapsto Z_0 \equiv (00|10)$$

$$CZ_{01} : Z_1 \equiv (00|01) \mapsto Z_1 \equiv (00|01).$$

generalizing this example, symplectic matrices of form

$$\left[\begin{array}{c|c} I & Q \\ \hline 0 & I \end{array} \right]$$

where Q is a symmetric binary matrix correspond to a product of S_i operators (nonzero entries Q_{ii} along the diagonal) and CZ_{ij} operators (nonzero entries $Q_{ij} = Q_{ji}$).

The binary symplectic matrix

$$\left[\begin{array}{cc|cc} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{array} \right]$$

acts on two-qubit Paulis as an CNOT_{01} operator because

$$\text{CNOT}_{01} : X_0 \equiv (10|00) \mapsto X_0 X_1 \equiv (11|00),$$

$$\text{CNOT}_{01} : X_1 \equiv (01|00) \mapsto X_1 \equiv (01|10),$$

$$\text{CNOT}_{01} : Z_0 \equiv (00|10) \mapsto Z_0 \equiv (00|10),$$

$$\text{CNOT}_{01} : Z_1 \equiv (00|01) \mapsto Z_0 Z_1 \equiv (00|11).$$

Generalizing this example, symplectic matrices of form

$$\left[\begin{array}{c|c} C & 0 \\ \hline 0 & C^{-T} \end{array} \right],$$

where C is an invertible binary matrix and C^{-T} is the inverse of the transpose of C , correspond to a product of CNOT operators.

4. Encoding operator of a stabilizer code

Quantum information is encoded into a stabilizer code by using a Clifford operation called an *encoding operator*. We now show how to construct an encoding operator for a stabilizer code given in terms of a check matrix G in standard form and the logical Pauli generators L_X and L_Z . We first construct a set of *destabilizer generators* R , which have the property that each row of R anticommutes with the corresponding row of G but commutes with all other rows of G , L_Z , and L_X :

$$R := \begin{bmatrix} R_X \\ R_Z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & I & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 \end{bmatrix}. \quad (8)$$

We then write the stabilizers, destabilizers, and logical Paulis in *tableau format* [14] as follows:

$$\tau := \begin{bmatrix} G_X \\ G_Z \\ L_X \\ R_X \\ R_Z \\ L_Z \end{bmatrix} = \begin{bmatrix} I & A_1 & A_2 & B & 0 & C_1 \\ 0 & 0 & 0 & D & I & C_2 \\ 0 & C_2^T & I & C_1^T & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & A_2^T & 0 & I \end{bmatrix}. \quad (9)$$

By construction, τ is a *binary symplectic matrix* because the commutation relations of G , L_X , R , and L_Z are equivalent to the identity $\tau \Omega \tau^T = \Omega$.

The Clifford operator $\mathcal{C} : \mathcal{H}_2^n \rightarrow \mathcal{H}_2^n$ corresponding to the binary symplectic matrix τ maps the k -qubit *logical state* $|\psi\rangle$ to the n -qubit *encoded state* $|\bar{\psi}\rangle$ as follows:

$$|\bar{\psi}\rangle := \mathcal{C}(|+\rangle^{\otimes(n-k)} |\psi\rangle). \quad (10)$$

B. Fault-tolerant logical operators of stabilizer codes

The objective of this paper is to identify fault-tolerant logical Clifford operators of a given stabilizer code. In this section, we explain in more detail the important concepts of fault tolerance and logical operators of stabilizer codes.

Fault tolerance is the property of a system that ensures that errors do not propagate catastrophically and can be systematically suppressed. For instance, when applying a two-qubit CZ gate, a Pauli- X error on the control qubit can spread to the target qubit. On the other hand, applying a single-qubit S gate will not inherently spread Z errors. In

certain qubit architectures, SWAP gates can be implemented with no direct interaction between qubits, meaning that the spread of Pauli errors is also constrained. In order to quantify fault tolerance in quantum circuits, we often use the distance of the circuit, which is defined as the minimum number of faulty physical error locations that can introduce a logical error to the computation.

A *logical operator* acting on an $[[n, k]]$ stabilizer code is a unitary that maps the code subspace to itself. More specifically, a *Clifford logical operator* $\bar{U}$ is a logical operator that applies a Clifford operator U to the code subspace, transforming the logical basis L_X, L_Z .

Finding a Clifford operator that applies a desired logical action is relatively straightforward. For instance, the Clifford operator $\bar{U} := \mathcal{C}(U \otimes I) \mathcal{C}^\dagger$ is a logical U operator where $\mathcal{C}$ is the encoding operator of Sec. II A 4. There is no guarantee that $\bar{U}$ has a fault-tolerant implementation—e.g., it may require a high-depth circuit. In this work, we consider the following types of Clifford logical operators:

- (1) *single-qubit transversal circuits*—circuits with only single-qubit (Clifford) physical gates
- (2) *SWAP-transversal circuits*—circuits with single-qubit Clifford gates and SWAPs
- (3) *general Clifford circuits*—circuits with single-qubit and general two-qubit physical gates

The first category involves transversal circuits and is considered fault tolerant because an error in applying one of the single-qubit Cliffords does not spread errors to other qubits. In the case of diagonal transversal logical operators (i.e., those composed of a series of single-qubit rotations around the Z axis), efficient methods for finding these have been set out in Ref. [10]. The methods in this paper identify transversal logical operators composed of other single-qubit Cliffords, such as H and products of H and S gates.

Depending on the qubit architecture, SWAP-transversal circuits may also be considered fault tolerant. In some architectures, SWAP gates can be accomplished without any direct interaction between the qubits and no physical operation involving both qubits simultaneously, and hence any faults in the process lead to independent errors on the qubits. Examples of such architectures include ion-trap shuttle systems [15] and atom arrays [16]. In fixed-layout architectures (e.g., superconducting), SWAP gates can be made fault tolerant via the use of auxiliary qubits, which may have their states reset during the process [12].

If SWAP gates are viewed as a primitive instruction of our system, then similarly to Ref. [4], the circuits with transversal operations and qubit permutations are fault tolerant, as the second category collapses to the first. This can happen on, e.g., Rydberg-atom [17], ion-trap [18], or electron [19] qubit architectures.

Hence, the fault tolerance of operations consisting of single-qubit Clifford gates and SWAPs can vary depending on the architecture of the system but can be viewed as a special case between transversal and general circuits.

The third category includes general Clifford circuits, which can be of high depth, and there is no guarantee that these can be implemented in a fault-tolerant way. Many qubit architectures have connectivity constraints that limit which qubits can be acted upon by multiqubit gates. The embedded-code method outlined in Sec. VII allows for the search to be limited to gates that can be implemented within the connectivity constraints of the given device, though the circuits may still be of high depth. We restrict our search in relevant examples to circuits with few CNOT and CZ gates, to exclude trivial circuit implementations such as decoding, applying a unitary, and then reencoding.

These circuits are fault tolerant when no qubit within a code block is involved in more than one two-qubit gate. Low-depth circuits can potentially be implemented if they are pieceably fault tolerant. This concept has been introduced in Ref. [20] and used in Ref. [6], where circuits are partitioned and interleaved with error-correction rounds of a possibly different code to ensure that low-weight physical errors are corrected before propagating to logical errors. Furthermore, logical gates that have physical entangling two-qubit gates (i.e., CNOT, CZ, or their generalizations) can be fault tolerant but with a reduced circuit distance. Specifically, if the distance of the code is d , then r entangling two-qubit gates may reduce the circuit distance to $d - 2r$ [21]. Note that the additional resources and depth of these circuits can become quite large in this case. We only find automorphisms corresponding to physical circuits with entangling two-qubit gates using the embedded-code technique from Sec. VII.

C. Previous work

In Ref. [3], the authors have described a way to map quantum stabilizer codes to additive codes over $GF(4)$. Automorphisms over $GF(4)$, which are defined as permutations and multiplications by a field element, are then used to define equivalent codes. To facilitate the automorphism-group calculation, the authors provide a mapping of Pauli strings of length n corresponding to the stabilizer generators to binary strings of length $3n$. A method for calculating automorphism groups of linear codes over finite fields has been developed in Ref. [22]. Here, we aim to bring some intuition into why binary strings of length $3n$ are useful to represent Pauli operators by drawing parallels to the symplectic representation of stabilizer codes. We set out different mappings of Pauli operators to binary strings of length $2n$, which allow us to find specific types of circuits as code automorphisms. We also address issues such as Pauli corrections required to preserve the stabilizer

group in circuits, since the original $GF(4)$ representation in Ref. [3] only works up to a sign of ± 1 .

In Ref. [23], automorphism groups of codes are considered to be qubit permutations that preserve the code space. The authors consider qubit permutations that map X checks to X checks and Z checks to Z checks. They then show that a larger set of fault-tolerant gates can be derived by using two copies of a CSS code and transversal CNOT operators between the two blocks. Our method accommodates non-CSS stabilizer codes and arbitrary combinations of physical CNOT, H , S , and SWAP gates. In Ref. [24], the authors consider automorphism gates as a generalization of transversal logical operators. They show that a universal gate set cannot be generated using transversal or SWAP-transversal operators on a code. Furthermore, a condition is given for a particular logical action to be realized on a code via code automorphisms.

Recently, automorphisms have been used to find logical operators on quantum codes based on ZX dualities, in which the X checks of a CSS code are mapped to the Z checks by combining qubit permutations and transversal H gates. In Ref. [4], ZX dualities have been used to implement fault-tolerant logical gates on surface codes. The method has been extended to quantum low-density parity-check (qLDPC) CSS codes in Refs. [5,6], with the aim of implementing H logical operators made from physical H and SWAP gates. The authors also give a construction based on such a ZX duality to perform a phase-type logical operator made from physical S , $S^\dagger$, and CZ gates. Notably, the ZX -duality method has been referenced in Refs. [12,25] as a way to implement H gates on bivariate bicycle codes and the carbon code. Here, we extend the method to non-CSS codes and show how to find a broader group of fault-tolerant logical operators composed of physical qubit SWAPs and H gates. We refer to these as SWAP + H gates. We generalize the method to SWAP + S and SWAP + $\sqrt{X}$ gates and connect it to the general logical Clifford gates through the automorphisms method.

III. AUTOMORPHISMS OF STABILIZER CODES

In this section, we show how to find SWAP-transversal logical operators of a stabilizer code by mapping the stabilizer generators to a binary linear code, finding the permutation automorphisms of the binary linear code, and then mapping these back to valid logical Clifford operators. We start this section by using the two-block symplectic representation of the stabilizer generators to find logical operators composed of H and SWAP gates. This generalizes the ZX dualities of Refs. [5,6] to non-CSS codes. We then show how to generalize the two-block method to find logical operators composed of either S and SWAP gates or $\sqrt{X}$ and SWAP gates. Finally, we introduce a three-block representation that allows us to find logical operators

composed of single-qubit Cliffords in the group $\langle H, S \rangle$ and SWAP gates.

A. Logical operators composed of physical SWAP and Hadamard gates

We now show how to find all logical operators that can be implemented using physical SWAP and H gates. To do this, we use the two-block symplectic binary representation of Pauli operators outlined in Eq. (3):

$$G := [G_X | G_Z]. \quad (11)$$

We then calculate the permutation automorphisms of the binary linear code with generator matrix G the code words of which are $\langle G \rangle$, the span of the rows of G over $\mathbb{F}_2$. The *automorphism group* $\text{Aut}(\langle G \rangle)$ of the binary linear code with generator matrix G is the set of permutations of the bits of G that map each code word in $\langle G \rangle$ to a potentially different code word (i.e., permutations of the bits that result in a permutation of the code words); we refer to these as *code automorphisms*. A set of generators for $\text{Aut}(\langle G \rangle)$ can be found by using the MAGMA computational-algebra software package.

In some cases, code automorphisms do not correspond to valid Clifford circuits constructed from H and SWAP gates acting on the stabilizer code. To address this, we consider the intersection of $\text{Aut}(\langle G \rangle)$ with the automorphism group of the binary linear code $\langle B \rangle$ the generator matrix of which is

$$B := [I_n | I_n].$$

The automorphism group of $\langle B \rangle$ is generated by the following elements:

- (1) *H type*: $(i, n + i)$. This exchanges the X and the Z blocks of the classical linear code. It corresponds to a physical H gate on qubit i of the stabilizer code.
- (2) *SWAP type*: $(i, j)(n + i, n + j), i, j \leq n$. This exchanges the X and the Z components of indices i and j on the classical linear code. It corresponds to a physical SWAP gate between qubits i and j on the stabilizer code.

By restricting to automorphisms of $\langle B \rangle$, we ensure that code automorphisms correspond to a combination of H and SWAP gates acting on the stabilizer code. We give more details on the derivation of this result in Appendix A 1.

Example 3. We will now calculate the $H + \text{SWAP}$ circuits corresponding to automorphisms of the perfect $[[5, 1, 3]]$ code. We have found the standard form of its stabilizer generators and logical basis in Eqs. (6) and (7), respectively. We thus find the automorphism group of the linear code on 10 bits $\langle G \rangle$ with the above generator matrix

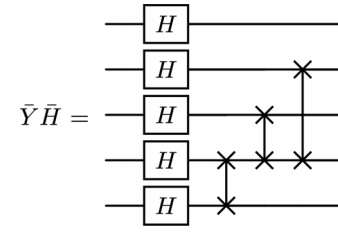


FIG. 2. The $H + \text{SWAP}$ gate of the $[[5, 1, 3]]$ code.

and intersect it with the automorphism group of the binary linear code on 10 bits $\langle B \rangle$ with generator matrix given by

$$B := [I_5 | I_5].$$

The group $\text{Aut}(\langle G \rangle) \cap \text{Aut}(\langle B \rangle)$ is of order 20 and is generated by three permutations. Two of these are qubit SWAPs, which act as logical identities. There is also a non-trivial logical operator consisting of H and SWAPs, which is shown in Fig. 2; this is similar to the circuit previously found in Ref. [20]. This operator has the following action on the logical operators of the code:

$$\begin{aligned} \bar{X} = ZIIZX &\mapsto XIZIX = -g_1\bar{Z}, \\ \bar{Z} = ZZZZZ &\mapsto XXXXX = -g_1g_2g_3g_4\bar{X}. \end{aligned}$$

As $\bar{X}$ and $\bar{Z}$ map to $-\bar{Z}$ and $-\bar{X}$, respectively, this operator corresponds to a logical $\bar{Y}\bar{H}$. We will describe how to find the logical action and adjust for the signs of mapped logical operators in more detail in Sec. IV.

In Ref. [5], the authors introduce the concept of ZX dualities for a class of CSS codes. ZX dualities map X checks of the code to Z checks and vice versa, and are implemented using H and SWAP operations. The above method generalizes the concept of a ZX duality to non-CSS codes and finds all Clifford logical operators that can be formed from SWAPs and H s, if such operators exist.

Furthermore, we can use alternative binary representations to find Clifford logical operators formed from SWAP and $S \equiv \sqrt{Z}$ gates. We do this by choosing the representation $[G_Z | G_X \oplus G_Z]$, where by $\oplus$ we denote the vector sum modulo 2. In this representation, swapping column i in the G_Z block with column $n + i$ in the $G_X \oplus G_Z$ block corresponds to applying an S_i gate. Similarly, we use the representation $[G_X | G_X \oplus G_Z]$ to find logical operators formed from SWAP and $\sqrt{X}$ gates.

B. Logical operators formed from physical single-qubit Clifford and SWAP gates

We have so far shown how to find logical operators formed from SWAPs and one of the single-qubit Cliffords. We now extend this to arbitrary combinations of single-qubit Cliffords and SWAPs. This is done by extending the

symplectic representation to a three-block representation as follows.

Starting from the symplectic representation of the stabilizer generators $G := [G_X | G_Z]$, we append an all-zero block with n columns and apply the binary transformation given by

$$\mathcal{E} := \begin{bmatrix} I & 0 & I \\ 0 & I & I \\ I & I & I \end{bmatrix},$$

so that

$$G_{\mathcal{E}} := [G_X | G_Z | 0] \quad \mathcal{E} = [G_X | G_Z | G_X \oplus G_Z].$$

Example 4. For the $[[5, 1, 3]]$ code of Example 1, the three-block binary representation of the stabilizer generators is as follows:

G_X	G_Z	$G_X \oplus G_Z$
10010	01100	11110
01001	00110	01111
10100	00011	10111
01010	10001	11011

To ensure that code automorphisms correspond to Clifford circuits on the stabilizer code, we take the intersection with the automorphism group of the (classical) linear code the generator matrix of which is

$$B := [I_n | I_n | I_n].$$

As can be seen, for this matrix, column swaps can correspond to different actions depending on their indices. Below, we will use the cycle notation for permutations (i, j, k) corresponding to the permutation that sends $i \mapsto j \mapsto k \mapsto i$. As before, n is the number of physical qubits and we let i be some integer such that $1 \leq i \leq n$. The indices of columns corresponding to qubit i are $i, n + i, 2n + i$ and the permutations correspond to the following gates:

- (1) *H type:* $(i, n + i)$ exchanges the X and the Z parts of the stabilizer and hence corresponds to a physical H gate on qubit i .
- (2) *S type:* $(n + i, 2n + i)$ exchanges the Z and the XZ parts of the stabilizer and corresponds to a physical $\sqrt{Z}$ gate (which is S up to Pauli correction based on its phase, as we will see in Sec. IV).
- (3) *SWAP gates:* $(i, j)(n + i, n + j)(2n + i, 2n + j)$ for $1 \leq j \leq n, j \neq i$ corresponds to the physical SWAP gate between qubits i and j .

We provide a more extensive description of this mapping in Appendix A 1 and show how to find the corresponding symplectic matrix representation in Appendix A 2.

TABLE I. The family of automorphism algorithms: we show how to find fault-tolerant logical operators formed from SWAP and various single-qubit Clifford gate sets. For a stabilizer code with check matrix $G := [G_X | G_Z]$, we choose a transformation matrix $\mathcal{E}$ to create a generator matrix $G_{\mathcal{E}}$ for a related binary linear code. We then find the automorphism group $\text{Aut}(\langle G_{\mathcal{E}} \rangle)$ using a computational-algebra system such as MAGMA. Restricting the automorphism group to $\text{Aut}(\langle G_{\mathcal{E}} \rangle) \cap \text{Aut}(\langle B \rangle)$ ensures that the logical operators use only the desired gate set.

Physical gate types	Binary representation	Transformation matrix $\mathcal{E}$	Intersect code B
1. $H + \text{SWAP}$	$[G_X G_Z]$	$\begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}$	$[I \ I]$
2. $S + \text{SWAP}$	$[G_Z G_X \oplus G_Z]$	$\begin{bmatrix} 0 & I \\ I & I \end{bmatrix}$	$[I \ I]$
3. $\sqrt{X} + \text{SWAP}$	$[G_X G_X \oplus G_Z]$	$\begin{bmatrix} I & I \\ 0 & I \end{bmatrix}$	$[I \ I]$
4. $\langle H, S \rangle + \text{SWAP}$	$[G_X G_Z G_X \oplus G_Z]$	$\begin{bmatrix} I & 0 & I \\ 0 & I & I \\ I & I & I \end{bmatrix}$	$[I \ I \ I]$

Each automorphism, expressed as a column permutation matrix P , is associated with a symplectic matrix $\bar{U}_S$ that we calculate as follows. By examining each of the allowed gates, we find that for $\bar{U}_S$ a binary symplectic matrix and W an invertible binary matrix that acts only on the XZ block of the stabilizer generators:

$$\mathcal{E} P \mathcal{E}^{-1} = U_S \oplus W. \quad (12)$$

For simplicity, going forward, we identify the symplectic matrix $\bar{U}_S$ with the corresponding Clifford operator. To ensure that $\bar{U}_S$ is a valid logical operator on the code, we may need to apply a Pauli correction and this is described in Sec. IV.

In Table I, we highlight the different automorphism-finding algorithms based on the physical circuits that they allow us to find.

IV. PAULI CORRECTION TO ENSURE SIGNS OF STABILIZER GROUP ARE PRESERVED

In this section, we show how to apply a Pauli correction to ensure that the Clifford operators $\bar{U}_S$ of Sec. III B identified via code automorphisms preserve the phases of the stabilizer generators. This step is necessary because symplectic matrices correspond to Clifford operators modulo Pauli operators and phases.

The Clifford operator $\bar{U}_S$ preserves the stabilizer group but only up to a sign of ± 1 . This is because for a nontrivial Pauli operator A to be a valid stabilizer, it must have eigenvalues of ± 1 and so $A^2 = I$. Note that $-A$ is also a valid stabilizer with eigenvalues ± 1 , but that $\pm iA$ is not a valid stabilizer as it has eigenvalues $\pm i$. Conjugation by a

Clifford operator U results in an operator that squares to I because

$$(UAU^\dagger)^2 = UAU^\dagger UAU^\dagger = UA^2U^\dagger = I. \quad (13)$$

Accordingly, the Clifford operator associated with $\bar{U}_S$ maps stabilizer generators to elements of the stabilizer group up to a sign of ± 1 but never $\pm i$.

For each stabilizer generator G_i with destabilizer R_i , we perform the following steps. We first determine the phase applied to each stabilizer generator by $\bar{U}_S$ so that for some binary vectors $\mathbf{x}, \mathbf{z}$ and integer u modulo 4,

$$\bar{U}_S G_i \bar{U}_S^\dagger = i^u X(\mathbf{x})Z(\mathbf{z}). \quad (14)$$

If $\bar{U}_S$ is a valid logical operator, each G_i should be mapped to a product of stabilizer generators. As a result, the binary vector $(\mathbf{x}|\mathbf{z})$ is a linear combination of the rows of the check matrix G modulo 2 and for some binary vector $\mathbf{g}$ of length $n - k$, we have

$$(\mathbf{x}|\mathbf{z}) = \mathbf{g}G. \quad (15)$$

Determining $\mathbf{g}$ allows us write the mapped operator as a product of the stabilizer generators so that for some integer v modulo 4,

$$\prod_{0 \leq j < n-k} G_j^{\mathbf{g}[j]} = i^v X(\mathbf{x})Z(\mathbf{z}). \quad (16)$$

If the phase applied by $\bar{U}_S$ is different to the phase of the product of stabilizer generators, we apply the destabilizer R_i to the Pauli correction. We now examine these steps in more detail.

A. Phase applied by a Clifford circuit

The logical operators found by the methods of Sec. III B are SWAP-transversal circuits. As SWAP gates do not apply any phase, the phase applied by the circuit $\bar{U}_S$ is determined by the transversal single-qubit Clifford gates.

Single-qubit Clifford gates can apply a phase of -1 in the case of the following Pauli operators:

$$HYH = -Y, \quad (17)$$

$$SYS^\dagger = -X, \quad (18)$$

$$\sqrt{X}Z\sqrt{X}^\dagger = -Y. \quad (19)$$

We capture the phase applied by single-qubit Cliffords by prepending an integer modulo 4 to the three-block binary representation (Sec. III B) of single-qubit Paulis so that $(p, a, b, c) := (p, x, z, x \oplus z)$. There are six single-qubit Clifford operators (up to Pauli factors and phases) and the update rules for their action on the single-qubit Pauli operators are set out in Table II. The update rules allow us to calculate the phase p of $\bar{U}_S G_i \bar{U}_S^\dagger$ in Eq. (14) by computing a dot product on the three-block representation.

Example 5. The action of $\Gamma := HS^\dagger$ on single-qubit Paulis is as follows:

$$\Gamma(X) = \Gamma(0, 1, 0, 1) = (1, 1, 1, 0) = iXZ = Y,$$

$$\Gamma(Y) = \Gamma(1, 1, 1, 0) = (1 - 1, 0, 1, 1) = Z,$$

$$\Gamma(Z) = \Gamma(0, 0, 1, 1) = (1 - 1, 1, 0, 1) = X.$$

B. Phase of mapped stabilizer

A valid logical operator maps each stabilizer generator to an element of the stabilizer group and this may be a product of various different stabilizer generators. We can express any Pauli operator as a product of stabilizer, destabilizer, and logical generators by using the tableau representation τ for the stabilizer code as set out in Sec. II A 4. For a stabilizer code with stabilizer, destabilizer, and X - and Z -logical generators in binary form G , R , L_X , and L_Z , respectively, the tableau τ is a symplectic matrix of form:

$$\tau := \begin{bmatrix} G \\ L_X \\ R \\ L_Z \end{bmatrix}. \quad (20)$$

TABLE II. Single-qubit Clifford gates and their action on phase + three-block representation.

Clifford gate	$\mathbf{X}$ (0, 1, 0, 1)	$\mathbf{Y}$ (1, 1, 1, 0)	$\mathbf{Z}$ (0, 0, 1, 1)	Action on (p, a, b, c)
I	X	Y	Z	(p, a, b, c)
H	Z	$-Y$	X	$(p + c - a - b, b, a, c)$
$S := \sqrt{Z}$	Y	$-X$	Z	$(p + a, a, c, b)$
$\sqrt{X} := HSH$	X	Z	$-Y$	$(p - b, c, b, a)$
$\Gamma := HS^\dagger$	Y	Z	X	$(p + c - b, c, a, b)$
$\Gamma^\dagger := SH$	Z	X	Y	$(p + c - a, b, c, a)$

As τ is a symplectic matrix, its inverse is given by $\Omega\tau^T\Omega$, because

$$\tau\Omega\tau^T\Omega = \Omega^2 = I. \quad (21)$$

Hence an unsigned Pauli operator $X(\mathbf{x})Z(\mathbf{z})$ can be expressed as a product of stabilizer, destabilizer, and logical Pauli generators by calculating the vector $\mathbf{b}$ as follows:

$$\mathbf{b} := (\mathbf{x}|\mathbf{z})\Omega\tau^T\Omega \quad (22)$$

$$:= (\mathbf{g}|\mathbf{a}_X|\mathbf{r}|\mathbf{a}_Z), \quad (23)$$

where $\mathbf{g}$ and $\mathbf{r}$ are binary vectors of length $n - k$ and $\mathbf{a}_X$ and $\mathbf{a}_Z$ are of length k , so that

$$(\mathbf{x}|\mathbf{z}) = \mathbf{b}\tau = \mathbf{g}G + \mathbf{a}_XL_X + \mathbf{r}R + \mathbf{a}_ZL_Z. \quad (24)$$

As we require that $(\mathbf{x}|\mathbf{z})$ is in the span of the stabilizer generators G , the vectors $\mathbf{a}_X$, $\mathbf{a}_Z$, and $\mathbf{r}$ should be zero, so that $(\mathbf{x}|\mathbf{z}) = \mathbf{g}G$. The phase v of Eq. (16) results from the product of stabilizers specified by the nonzero entries of $\mathbf{g}$.

C. Application of destabilizer

We now show that applying a destabilizer has the desired effect of updating signs for mapped stabilizer generators. Assume that $u \neq v$ in Eqs. (14) and (16), so that

$$\bar{U}_S G_i \bar{U}_S^\dagger = - \prod_{0 \leq j < n-k} G_j^{\mathbf{g}[j]}. \quad (25)$$

If we instead conjugate G_i by $\bar{U}_S R_i$, where R_i is the destabilizer of G_i , we have

$$\bar{U}_S R_i G_i R_i \bar{U}_S^\dagger = \bar{U}_S (R_i G_i R_i) \bar{U}_S^\dagger \quad (26)$$

$$= -\bar{U}_S G_i \bar{U}_S^\dagger \quad (27)$$

$$= \prod_{0 \leq j < n-k} G_j^{\mathbf{g}[j]}. \quad (28)$$

Because R_i commutes with all other stabilizer generators, the sign of each mapped stabilizer generator can be updated independently. Accordingly, we choose the Pauli correction $\bar{U}_P$, which is the product of the destabilizers R_i , where the corresponding mapped stabilizer generator is an element of the stabilizer group up to a sign of -1 .

V. DETERMINING LOGICAL ACTION OF A CLIFFORD CIRCUIT

In this section, we show how to determine the logical action of a Clifford logical operator $\bar{U}_S$. We will use the tableau form of Sec. II A 4 to apply a Pauli correction to $\bar{U}_S$

and to describe the logical action as a $2k \times 2k$ symplectic matrix.

For each logical Pauli generator L_i , we determine the mapped operator $\bar{U}_S L_i \bar{U}_S^\dagger = i^u X(\mathbf{x})Z(\mathbf{z})$ as described in Sec. IV A. We then calculate the vector $\mathbf{b} = (\mathbf{g}|\mathbf{a}_X|\mathbf{r}|\mathbf{a}_Z)$ of Eq. (23). The vectors $\mathbf{a}_X$ and $\mathbf{a}_Z$ describe the product of logical X and Z generators, respectively, which appear in the mapped logical operator. Up to a Pauli operator, the logical action of $\bar{U}_S$ is given by the symplectic matrix U_{ACT} , the i th row of which is given by the length- $2k$ vector $(\mathbf{a}_X|\mathbf{a}_Z)$. A logical Y operator is represented as $XZ = -iY$, so we adjust the logical action by multiplying by the phase $i^{-\mathbf{a}_X \cdot \mathbf{a}_Z}$. The *logical-action phase adjustment* ensures that logical Pauli generators, which square to I , map to products of stabilizer and logical Pauli generators that also square to I . For example, this step ensures that a logical $\bar{S}$ maps $\bar{X}$ to $\bar{Y} = i\bar{X}\bar{Z}$ rather than $\bar{X}\bar{Z}$, which squares to $-I$. This also guarantees that the logical action is correct up to a sign of ± 1 .

We ensure that the sign of the logical action is correct by applying a Pauli correction using a similar method to that outlined in Sec. IV. This time, we use the vector $\mathbf{b}$ to calculate a product of stabilizer and logical generators, apply the logical-action adjustment phase $i^{-\mathbf{a}_X \cdot \mathbf{a}_Z}$, and check whether this has the same phase as the mapped logical operator. If not, we apply the destabilizer of the logical operator to update the sign. For a logical $\bar{X}_i$ corresponding to row i of L_X , the destabilizer is the anticommuting logical $\bar{Z}_i$ corresponding to row i of L_Z . As the logical generators commute with all other stabilizer and logical generators, this allows us to update the signs independently. A combined algorithm to calculate the Pauli correction $\bar{U}_P$ and the logical action U_{ACT} is set out in Algorithm 2.

VI. REPRESENTATION OF SYMPLECTIC MATRICES AS CIRCUITS

Both the physical implementations of logical operators and their logical actions have so far been presented as a symplectic matrix plus a Pauli correction. In this section, we show how we translate this representation into a circuit composed of single- and two-qubit Clifford gates.

In Theorem B1 in Appendix B, we show that any symplectic matrix can be written as $U = U_A U_B U_C U_H$, where

- (1) U_A represents a product of $C(X, X)$ and $\sqrt{X}$ operators
- (2) U_B is a product of CZ and S operators
- (3) U_C is a product of CNOT operators and
- (4) U_H is a product of single-qubit H operators

This leads to a natural decomposition of Clifford unitaries in six layers of gates as H -CNOT- S -CZ- $\sqrt{X}$ - $C(X, X)$.

VII. MULTIQUBIT PHYSICAL GATES VIA AUTOMORPHISMS OF EMBEDDED CODES

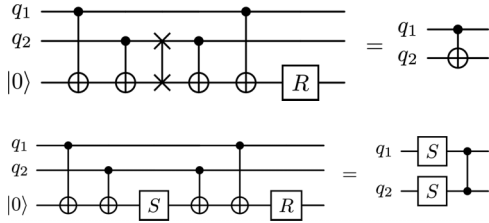
We have so far been working with SWAP-transversal logical Clifford operators that consist of physical SWAP and single-qubit Clifford operators. The embedded-code technique allows us to find a broader range of logical operators, including those that employ two-qubit CNOT and CZ physical gates. The resulting logical Clifford operators may be high depth and are not guaranteed to be fault tolerant. The technique does allow for restrictions on which qubits can interact via two-qubit gates and so may be useful for certain device architectures.

The embedded-code technique has been introduced in Ref. [10], as a method for finding logical operators of CSS codes involving multiqubit diagonal gates. In the present work, we generalize the method to non-CSS codes and to nondiagonal Clifford gates.

This method aims to describe the action of physical two-qubit entangling gates, such as CNOT and CZ, as an automorphism of a larger code. Specifically, the method works by embedding the original n qubits to a code with

$$n + \binom{n}{2}$$

qubits, where the extra $\binom{n}{2}$ qubits are initialized in the $|0\rangle$ state. We then make use of the following circuit identities:



where R is the reset instruction. Since we do not actually use the extra qubits, the operation is equivalent to a CNOT or CZ gate.

We construct an embedded code by defining a $m \times n$ binary matrix M the rows of which are of weight 2. Each row of M corresponds to a pair of qubits in the original code space. The connectivity constraints of a particular device can be encoded by only including in M rows where the corresponding two-qubit gates are available. We provide more details about the definition and calculation of the embedded code in Appendix D.

For a stabilizer code with check matrix $G := [G_X | G_Z]$, the check matrix of the embedded code on $n + m$ qubits is given by

$$G_V := \left[\begin{array}{cc|cc} G_X & G_X M^T & G_Z & 0 \\ 0 & 0 & M & I \end{array} \right]. \quad (29)$$

In Appendix D 4, we derive the relations between permutations of the columns of the check matrix of the embedded

TABLE III. Actions on the embedded code space and their corresponding physical gates on the original code. The notation $S_{(12)}$ indicates an S operator applied to the auxiliary qubit represented by the pair of physical qubits (1, 2).

Embedded code space	Original code space
$S_{(12)}$	$S_1 S_2 CZ_{12}$
$\text{SWAP}_{(12),2}$	CNOT_{12}

code and the corresponding single- and two-qubit Clifford operations on the original code space. We include the results in Table III. In Sec. VIII B, we show that we find a wider range of logical Clifford operators via the embedded-code method versus the methods of Sec. III B for the $[[4, 2, 2]]$ code.

VIII. EXAMPLES AND RESULTS

A. Five-qubit perfect code

We have applied the algorithms of Secs. III and IV to the five-qubit perfect code with code parameters $[[5, 1, 3]]$. This code has a well-known transversal logical $\Gamma = HS^\dagger$ gate [20]. Here, we find a SWAP-transversal circuit that implements the same logical gate up to a Pauli correction. The SWAP-transversal alternative in Fig. 3 uses fewer single-qubit physical Clifford gates. We illustrate the SWAP-transversal implementation of an S gate in Fig. 4.

Our results are available as an interactive notebook in Ref. [9].

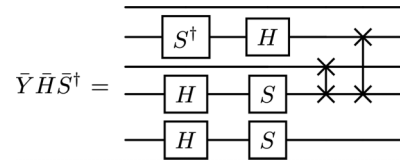


FIG. 3. The SWAP-transversal $\bar{\Gamma} = \bar{H}\bar{S}^\dagger$ gate of the $[[5, 1, 3]]$ code with a $\bar{Y}$ correction.

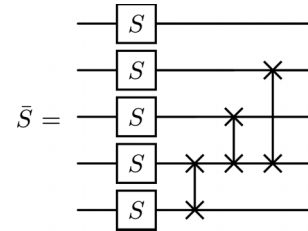


FIG. 4. The SWAP-transversal $\bar{S}$ gate of the $[[5, 1, 3]]$ code.

B. Four-qubit code

We now illustrate the three-block method of Sec. III B by applying it to the $[[4, 2, 2]]$ code, which has the following stabilizer generators and logical operators:

$$\mathbf{G} := \begin{bmatrix} XXXX \\ ZZZZ \end{bmatrix}, \quad \begin{bmatrix} L_X \\ L_Z \end{bmatrix} = \begin{bmatrix} XIIX \\ XIXI \\ ZIZI \\ ZIIZ \end{bmatrix}.$$

The resulting logical operators are shown in Table IV. Note that we do not obtain the full logical Clifford group because we do not have logical H and S addressing individual logical qubits.

TABLE IV. The SWAP-transversal logical Clifford gates of the $[[4, 2, 2]]$ code.

Gate	Circuit	Type
$\bar{H}_1 \bar{H}_2$	$H_1 H_2 H_3 H_4 \text{SWAP}_{3,4}$	Transversal
$\bar{CZ}$	$S_1^\dagger S_2^\dagger S_3 S_4$	Transversal
$\bar{\text{CNOT}}_{1,2}$	$\text{SWAP}_{2,4}$	SWAP-transversal
$\bar{\text{CNOT}}_{2,1}$	$\text{SWAP}_{2,3}$	SWAP-transversal
$\bar{\text{SWAP}}_{1,2}$	$\text{SWAP}_{3,4}$	SWAP-transversal

We do obtain a full set of logical Clifford operators for the $[[4, 2, 2]]$ code via the embedded-code technique of Sec. VII, though these are not SWAP-transversal and so are not necessarily fault tolerant. We choose the embedding matrix $\begin{bmatrix} I \\ M \end{bmatrix}$, where M is the binary matrix the rows of which are the binary vectors of length 4 and weight 2. We add $m := \binom{4}{2} = 6$ auxiliary qubits and the check matrix of the embedded code is as follows:

$$G_V := \left[\begin{array}{cc|cc} G_X & G_X M^T & G_Z & 0 \\ 0 & 0 & M & I \end{array} \right]$$

$$= \left[\begin{array}{cc|cc} 1111 & 000000 & 0000 & 000000 \\ 0000 & 000000 & 1111 & 000000 \\ 0000 & 000000 & 1100 & 100000 \\ 0000 & 000000 & 1010 & 010000 \\ 0000 & 000000 & 1001 & 001000 \\ 0000 & 000000 & 0110 & 000100 \\ 0000 & 000000 & 0101 & 000010 \\ 0000 & 000000 & 0011 & 000001 \end{array} \right]$$

We find circuits implementing S and $\sqrt{X}$ on each logical qubit using Algorithm 3, which searches for implementations of logical operators with a desired action. These circuits are shown in Table V.

TABLE V. The logical Clifford gates of the $[[4, 2, 2]]$ code via the embedded-code technique.

Gate	Circuit	Type
$\bar{S}_1$	$S_1 S_3 \text{CZ}_{1,3}$	General Clifford
$\bar{S}_2$	$S_1 S_4 \text{CZ}_{1,4}$	General Clifford
$\sqrt{X}_1$	$\sqrt{X}_1 \sqrt{X}_4 \text{C}(X, X)_{1,4}$	General Clifford
$\sqrt{X}_2$	$\sqrt{X}_1 \sqrt{X}_3 \text{C}(X, X)_{1,3}$	General Clifford

Our results are available as an interactive notebook in Ref. [9].

C. Best-known distance codes with $k = 1$

We have then applied the algorithms in Secs. III B and IV to best-known distance codes with $k = 1$ logical qubits from Ref. [11] on up to $n = 30$ qubits. Our results are presented in Table VI. We observe that most $k = 1$ codes have the full single-qubit Clifford group, achieved via SWAP-transversal gates. On the other hand, when restricting the logical-gate implementation to only transversal operations, we can only achieve a small subset of these gates. In Table VII, we list the transversal-gate implementations.

TABLE VI. The SWAP-transversal gates of $[[n, 1, d]]$ codes up to $n \leq 30$ physical qubits.

Logical gates	n
$\bar{I}$	4, 16, 28
$\bar{S}$	24
$\bar{S}\bar{H}$	10 – 15
$\langle \bar{S}, \bar{H} \rangle$	All other codes with $n \leq 30$

TABLE VII. The transversal gates of $[[n, 1, d]]$ codes up to $n \leq 30$ physical qubits.

Logical gates	n
$\bar{I}$	All other codes with $n \leq 30$
$\bar{S}$	2, 3
$\bar{S}\bar{H}$	5, 17, 25, 29
$\langle \bar{S}, \bar{H} \rangle$	7

Our results for $k = 1$ best-known distance codes, available in Ref. [9], indicate that small non-CSS codes, when combined with SWAP gates, can realize logical S and H gates with lower physical-qubit overhead than is possible using other codes, such as color codes, with comparable distance. These codes may thus offer practical advantages for quantum architectures with limited qubit resources.

D. Bivariate bicycle (BB) codes

The bivariate bicycle (BB) codes [12] have generated a lot of interest in the QEC community. Their construction is based on the generalized bicycle codes from Ref. [26] which is a refinement of the original bicycle construction in Ref. [27]. An important property of the codes introduced in Ref. [12] is that their logical qubits can be partitioned into two blocks of equal size, called the “unprimed” and “primed” blocks. The authors then identify two types of SWAP-transversal gates:

- (1) *Grid-translation automorphisms*: SWAPs along the Tanner graph that correspond to logical CNOTs between qubits in the same logical block.
- (2) *ZX dualities*: logical global H s and SWAPs between pairs of qubits across two logical blocks.

In Ref. [28], the authors identify a different subfamily of generalized bicycle codes. For this subfamily, the authors construct a basis of logical Pauli operators and describe SWAP-transversal gates in this basis. They identify an additional symmetry and a CZ-type gate based on a ZX duality similar to the construction of Ref. [4] for the toric code. Here, we use our methods to find new logical Clifford gates for the BB codes of Ref. [12].

Using the method of Sec. III A, we have found $H + \text{SWAP}$ logical operators for the codes of Ref. [12] for up to $n = 360$ qubits. This allows us to identify both grid translation and ZX dualities for the relevant codes. Furthermore, by allowing unrestricted physical SWAP operations, we find additional generators of the automorphism group. These typically implement a logical CNOT circuit between the primed and unprimed logical blocks (i.e., without any logical H gate). The logical action of one of these is shown in Fig. 5 for the $[[108, 8, 10]]$ code. In Table VIII, we list the orders of the automorphism and logical-action groups that we find and compare these to the results of Refs. [12,29].

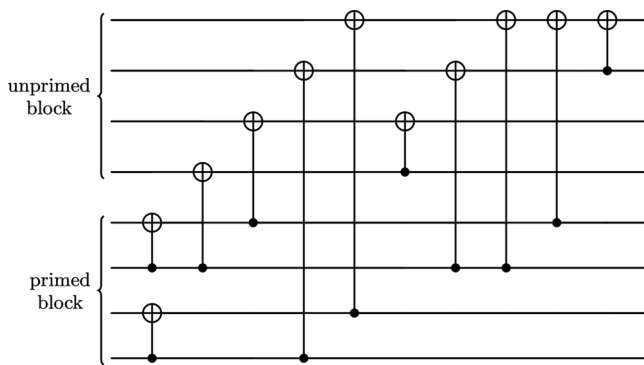


FIG. 5. The logical action of the additional automorphism-group generator for the $[[108, 8, 10]]$ BB code as a CNOT circuit.

TABLE VIII. The orders of the automorphism and logical-action groups of the BB codes of Ref. [12]. The order of the automorphism groups that we find is denoted $|\text{Aut}(\mathcal{G})|$. The order of the group generated by the grid-translation automorphisms and the ZX duality identified in Ref. [12] is denoted Aut_{IBM} and is the same as the number of physical qubits of the code. The size of the logical-action group that we find is denoted $|\Lambda|$. Where available, we compare this to the size of the logical-action group found in Ref. [12], which is denoted $|\Lambda_{\text{IBM}}|$.

Code	$ \text{Aut}(\mathcal{G}) $	$ \text{Aut}_{\text{IBM}} $	$ \Lambda $	$ \Lambda_{\text{IBM}} $
$[[72, 12, 6]]$	864	72	864	72
$[[90, 8, 10]]$	360	90	72	...
$[[108, 8, 10]]$	216	108	36	...
$[[144, 12, 12]]$	288	144	144	72
$[[288, 12, 18]]$	1728	288	432	...
$[[360, 12, \leq 24]]$	720	360	144	...

Our results for BB codes are available as an interactive notebook in Ref. [9].

IX. COMPUTATIONAL COMPLEXITY OF FINDING AUTOMORPHISM GROUPS

In this section, we discuss the computational complexity of finding automorphism gates using our methods. The most computationally complex part is finding the automorphism groups of the related binary linear code. Finding code automorphisms scales exponentially in the code dimension and so is not practical in all cases. For larger codes, we instead calculate the matrix automorphisms of the two- or three-block binary representations of the stabilizer generators. Matrix automorphisms can be found by mapping to a graph-isomorphism problem. While graph isomorphism is believed to be an NP-intermediate problem, a quasipolynomial run-time algorithm has been published [30,31]. In our work, we use the open-source BLISS software package for determining graph isomorphism [8] and this works well in practice, even for large graphs.

A. Code automorphisms

In Sec. III, we have considered *code automorphisms* of a binary linear code with generator matrix G . These are permutations of the bits of the code that result in a permutation of the code words. The code words $\langle G \rangle$ are the span of G over $\mathbb{F}_2$ and so we write $\text{Aut}(\langle G \rangle)$ for the group of automorphisms of the code with generator matrix G .

Finding code automorphisms is of exponential complexity in the worst case. The complexity of Leon’s algorithm is exponential in the code dimension [22]. Sendrier and Skersys’s algorithm has improved efficiency but is still exponential in the dimension of the hull of the code (i.e., the intersection of the code and its dual [32]).

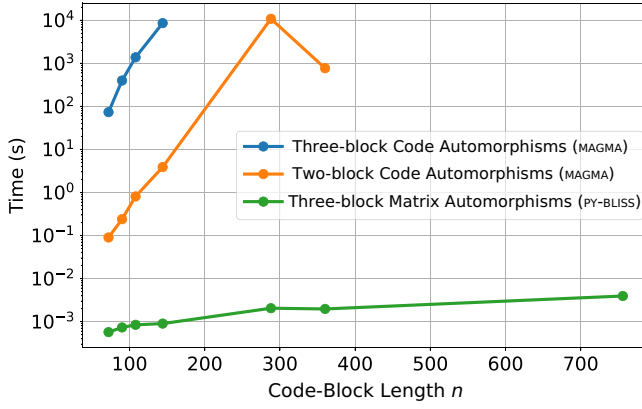


FIG. 6. The time analysis for computing automorphisms of BB codes. The blue line shows the time taken to compute code automorphisms using MAGMA for the three-block representation in Table I, row 4. The orange line shows the time taken to compute code automorphisms using MAGMA for the two-block representation in Table I, row 1. The green line shows the time taken to compute matrix automorphisms for the three-block representation in Table I, row 4. Note that the group orders found via each representation are the same for all BB codes.

To calculate code automorphisms, we use the proprietary MAGMA computational-algebra software package [7], which has an implementation of Leon’s algorithm. For large codes, the computation time of the MAGMA implementation becomes impractical—e.g., we have only been able to use MAGMA for BB codes on up to $n = 144$ qubits using the three-block binary linear code described in Sec. III B and up to $n = 360$ using the two-block ZX-duality representation of Sec. III A, as can be seen in Fig. 6.

B. Matrix automorphisms

For larger codes, we instead consider *matrix automorphisms*. The matrix automorphisms $\text{Aut}(M)$ of the matrix M are permutations of the columns of the matrix that result in a permutation of the rows of M . If M has the same span as the generator matrix G , then $\text{Aut}(M)$ is a subgroup of $\text{Aut}(\langle G \rangle)$. On the other hand, if M is the matrix the rows of which are the code words $\langle G \rangle$, we have $\text{Aut}(M) = \text{Aut}(\langle G \rangle)$. Matrix automorphisms can be calculated by mapping to a graph-isomorphism problem. The graph-isomorphism problem is thought to be NP-intermediate and quasipolynomial complexity algorithms are available in the open-source BLISS and NAUTY software packages [8,33].

To map to a graph-isomorphism problem, we start with an $a \times b$ binary matrix M that has the same span as G . The rows of M need not be independent and may be an over-complete basis. We consider the automorphisms of the a bicolored graph $\mathcal{G}_M$ with $a + b$ vertices, defined as follows. Each row of M corresponds to a white vertex and each column to a black vertex. There is an edge between

white vertex i and black vertex j if and only if $M_{ij} = 1$. *Graph automorphisms* are permutations of the white vertices and/or black vertices that result in a permutation of the edges of the graph. A generating set of graph automorphisms can be calculated using the open-source NAUTY or BLISS software packages, even for very large graphs.

C. Comparison of matrix- and code-automorphism methods

Where M has the same span as G , the matrix-automorphism group of M is a subgroup of the automorphism group of the code with generator matrix G (i.e., $\text{Aut}(M) \subseteq \text{Aut}(\langle G \rangle)$). In general, this is a strict inclusion relationship but, in some cases, particularly where we use an over-complete set of stabilizer generators, the groups are equal.

As our first example, we consider the automorphisms of the $[[5, 1, 3]]$ code. We first consider the independent set of four stabilizer generators as set out in Example 1. Using a three-block representation (see Sec. III B), we find a trivial matrix-automorphism group. If we instead use an over-complete set of stabilizer generators (i.e., the five cyclic shifts of the check $XZZX7$), we find a matrix-automorphism group of order 20. This corresponds to cyclic shifts of the qubits that act trivially and the ZX duality of Example 3. Finally, if we consider all $2^4 = 16$ elements of the stabilizer group, we obtain a matrix-automorphism group of order 360, which matches the order of the code-automorphism group.

In the case of BB codes, matrix automorphisms may suffice to generate the entire code-automorphism group. BB codes have a canonical over-complete set of n stabilizer generators. The order of the matrix-automorphism group of the canonical stabilizers matches the order of the code-automorphism group for members of the bivariate-bicycle-code family of Ref. [12] for up to $n = 360$ qubits. We conjecture that the canonical stabilizer generators form a complete set of minimum-weight stabilizers for BB codes, though proving this is an open question. As automorphisms are weight preserving, the set of minimum-weight stabilizer generators is closed under the action of code automorphisms, so that any code automorphism is also an automorphism of the minimum-weight stabilizer generators. As illustrated in Fig. 6, matrix automorphisms of BB codes take significantly less time to find than code automorphisms.

X. CONCLUSIONS

In this work, we have presented a number of different algorithms to identify the logical Clifford gates implementable through code automorphisms. The key to our methods is mapping a stabilizer code to a binary linear code. We have first extended the methods for identifying fold-transversal $\text{SWAP} + H$ operations to non-CSS stabilizer codes. We have also generalized our method to

finding $\text{SWAP} + S$ and $\text{SWAP} + \sqrt{X}$ physical circuits. Furthermore, we have combined the above to find all single-qubit Clifford + SWAP circuits preserving the code space, using a three-block representation of the stabilizer code. These correspond to code automorphisms over $GF(4)$ as set out in Ref. [3], but we have more rigorously applied Pauli corrections to ensure that the signs of the stabilizer generators are preserved and to interpret the logical actions of the resulting gates. Our methods show some very interesting results when applied to the best-known distance codes with one logical qubit. Specifically, we find that most of them allow for SWAP-transversal implementations of the single-qubit Clifford group, far more than is achievable using only transversal gates. Finally, we have found that for the bivariate bicycle codes [12], there exist further automorphisms with interesting logical actions if we allow for arbitrary physical-qubit SWAPs. We have provided a package that implements all the aforementioned methods using PYTHON and either the computational-algebra software MAGMA or the open-source BLISS software package for computing automorphism groups.

Overall, our results show that SWAP gates can substantially enrich the logical gate sets implementable in stabilizer codes.

A. Open questions and future work

Several follow-up questions naturally arise. First, we observe that automorphisms play an increasingly important role in modern fault-tolerant protocols—particularly in lattice surgery and extractor systems [29,34], where they can reduce the auxiliary qubit overhead. Our automated framework discovers symmetries beyond those previously identified analytically, raising the question of whether these additional automorphisms can lead to further resource savings in such protocols. Second, connections to decoding are also compelling: recent work [35] has shown that codes with rich automorphism groups admit highly parallelizable, accurate, and efficient decoding algorithms. These observations underscore the practical value of symmetries in quantum error-correction codes and motivate further exploration into their role in both decoding and logical gate construction. Finally, we note that our proposed constructions may be especially well suited for experimental platforms with native SWAP gates and we hope this work can contribute to accelerating near-term demonstrations of fault-tolerant gates.

ACKNOWLEDGMENTS

H.S. is supported by the Engineering and Physical Sciences Research Council (EPSRC) (Grant No. EP/S021582/1). H.S. also acknowledges support from the National Physical Laboratory. M.W. and D.E.B. are supported by the EPSRC (Grant No. EP/W032635/1). A.R., M.W., and D.E.B. are supported by the EPSRC

(Grant No. EP/S005021/1). S.K. and D.E.B. are supported by the EPSRC (Grant No. EP/Y004620/1). S.K., M.W., and D.E.B. are supported by the EPSRC (Grant No. EP/T001062/1). We would like to thank Geoff Bailey, Simon Burton, Earl Campbell, Alastair Kay, and Heather Leitch for helpful discussions.

DATA AVAILABILITY

The algorithms introduced in this paper are implemented as a Python package and are openly available at [9].

APPENDIX A: CORRESPONDENCE OF PERMUTATIONS AND SYMPLECTIC MATRICES

We will now give a more detailed description of our approach to mapping automorphisms of a binary linear code to SWAP-transversal Clifford gates. We begin by showing a mapping of the circuits to automorphisms of a binary linear code. We then construct the mapping of these permutations into symplectic matrices using the transformation matrix.

1. Connection between SWAP-transversal operations and automorphism groups of binary cyclic block codes

We will now prove that there is a 1-to-1 correspondence between the automorphism group of the classical binary linear code of size $3n$ generated by the matrix:

$$B_n := [I_n | I_n | I_n]$$

and all circuits composed of H, S and SWAP gates on n qubits, modulo Pauli operators.

We start by finding a description of the automorphism group of the binary linear code $\langle B_n \rangle$ generated by B_n . Let e_i be the binary vector of length n with 1 in position i and 0 everywhere else. The generator matrix of $\langle B_n \rangle$ can be written as:

$$g_i := (e_i | e_i | e_i), \forall 1 \leq i \leq n. \quad (\text{A1})$$

Any code word $b \in \langle B_n \rangle$ will thus be of the form

$$\begin{aligned} b &= \sum_{i=1}^n a_i g_i \\ &= (a_1, a_2, \dots, a_n, a_1, a_2, \dots, a_n, a_1, a_2, \dots, a_n), \end{aligned} \quad (\text{A2})$$

for some $a_i \in \{0, 1\}$.

Theorem A1. The automorphism group of the binary linear code generated by B_n , $\text{Aut}(\langle B_n \rangle)$ is generated by permutations of the following three types:

- (1) $(i, n+i), \forall 1 \leq i \leq n$
- (2) $(n+i, 2n+i), \forall 1 \leq i \leq n$

$$(3) (i, j)(n + i, n + j)(2n + i, 2n + j), \forall 1 \leq i < j \leq n$$

and its size is $|\text{Aut}(\langle B_n \rangle)| = 6^n n!$

Proof. Let P be an automorphism of $\langle B_n \rangle$ ie $P \in \text{Aut}(\langle B_n \rangle)$. As we have described in Sec. III, P will be a permutation of the $3n$ bits, $P \in S_{3n}$, that preserves the code space. We check the action of P on a generator g_i . As this is a binary vector of length $3n$ with three nonzero coordinates, $P(g_i)$ will also be a binary vector of length $3n$ with three nonzero coordinates. There are two cases, depending on where the mapped nonzero coordinates will be: we can either have more than two nonzero elements in one of the three n -tuples or exactly one nonzero element in each n -tuple.

Specifically, we have the following cases:

- (1) $P(g_i) = (e_j + e_k | e_l | 0)$ or $P(g_i) = (e_j + e_k + e_l | 0 | 0)$, for some $1 \leq j, k, l \leq n$. This is not a valid code word of $\langle B_n \rangle$, as it is not of the form given in Eq. (A2)—and similarly if we have at least two nonzero elements in the second or third tuple.
- (2) $P(g_i) = (e_j | e_k | e_l)$, for some $1 \leq j, k, l \leq n$. If $j \neq k$. Then, matching the form of Eq. (A2), from the first n bits we obtain that $a_j = 1$ and $a_k = 0$, while from the second n bits we find that $a_j = 0$ and $a_k = 1$. which is a contradiction—and similarly if $j \neq l$. Hence, we require $j = k = l$ and $P(g_i) = (e_j | e_j | e_j) = g_j$. This is a valid code word of $\langle B_n \rangle$ and hence P is a valid automorphism.

We see that all the automorphisms of $\langle B_n \rangle$ are of the form $P(g_i) = g_j$, for some $1 \leq i, j \leq n$ or, equivalently, $P : (e_i | e_i | e_i) \mapsto (e_j | e_j | e_j)$. Thus, for each i , P can swap column i with $n + i$ or $2n + i$, or column $n + i$ with $2n + i$, and then columns i with j , $n + i$ with $n + j$, and $2n + i$ with $2n + j$. This is equivalent to permuting each column to the same index in each of the three blocks and permuting n indices separately. Thus, the elements of $\text{Aut}(\langle B_n \rangle)$ are generated by the following permutations:

- (1) $(i, n + i), \forall 1 \leq i \leq n$
- (2) $(n + i, 2n + i), \forall 1 \leq i \leq n$
- (3) $(i, j)(n + i, n + j)(2n + i, 2n + j), \forall 1 \leq i < j \leq n$

Hence the order of the automorphism group of the binary linear code $\langle B_n \rangle$ is

$$|\text{Aut}(\langle B_n \rangle)| = |S_3|^n |S_n| = 6^n n!. \quad (\text{A3})$$

■

Let K be the set of all possible circuits composed of H s, S s, and SWAP s. These form a group, with the group operation being circuit composition.

We now introduce the map $\phi : \text{Aut}(\langle B_n \rangle) \mapsto K$, defined by

$$(i, n + i) \mapsto H_i, \quad \forall 1 \leq i \leq n, \quad (\text{A4})$$

$$(n + i, 2n + i) \mapsto S_i, \quad \forall 1 \leq i \leq n, \quad (\text{A5})$$

$$(i, j)(n + i, n + j)$$

$$(2n + i, 2n + j) \mapsto \text{SWAP}_{ij}, \quad \forall 1 \leq i \leq j \leq n. \quad (\text{A6})$$

This is a group isomorphism, as multiplying the permutations before applying the map has the same effect as applying ϕ and then composing the circuits.

Similarly, for the two-block representations discussed in Sec. III A, the automorphism group of

$$B := [I_n | I_n]$$

is generated by permutations of the type $(i, n + i)$ and $(i, j)(n + i, n + j)$ corresponding to a transversal Clifford gate, such as H_i for the symplectic binary representation, and SWAP_{ij} , respectively. By the same argument as in Theorem A1, the size of the group is then $|S_2|^n |S_n| = 2^n n!$.

2. Mapping of permutation automorphisms to symplectic matrices

We now show how each of the allowed permutation automorphisms translate to symplectic matrices under conjugation by the transformation matrix $\mathcal{E}$. For the three-block representation $[G_X | G_Z | G_X \oplus G_Z]$ of Sec. III B, the transformation matrix and its inverse are given by

$$\mathcal{E} := \begin{bmatrix} I & 0 & I \\ 0 & I & I \\ I & I & I \end{bmatrix}, \quad \mathcal{E}^{-1} := \begin{bmatrix} 0 & I & I \\ I & 0 & I \\ I & I & I \end{bmatrix}. \quad (\text{A7})$$

a. S -type permutation

On a single physical qubit, the permutation corresponding to an S gate as we defined it above in Eq. (A5) is $(2, 3)$ in cycle notation. Its matrix representation is thus given by

$$P_S := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \quad (\text{A8})$$

We see that right multiplication by this matrix swaps the Z and XZ components of the three-block binary representation of Sec. III B.

We calculate the corresponding symplectic matrix as follows:

$$U_S \oplus W_S := \mathcal{E} P_S \mathcal{E}^{-1} \quad (\text{A9})$$

$$= \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (\text{A10})$$

$$= \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (\text{A11})$$

$$= \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{A12})$$

$$= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \oplus [1]. \quad (\text{A13})$$

The symplectic matrix U_S corresponds to an S operator (see Example 2) and the action of W_S on the XZ block is trivial.

b. Hadamard-type permutation

The permutation matrix swapping the X and Z blocks on a single qubit is given by

$$P_H := \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (\text{A14})$$

Under conjugation by $\mathcal{E}$, we obtain the following symplectic matrix:

$$U_H \oplus W_H := \mathcal{E} P_H \mathcal{E}^{-1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \oplus [1]. \quad (\text{A15})$$

The symplectic matrix U_H corresponds to a single-qubit H operator (see Example 2).

c. SWAP-type permutation

Now consider a permutation of qubits given by an $n \times n$ permutation matrix Q . The corresponding permutation matrix in the three-block representation is given by

$$P_Q := \begin{bmatrix} Q & 0 & 0 \\ 0 & Q & 0 \\ 0 & 0 & Q \end{bmatrix}. \quad (\text{A16})$$

Under conjugation by $\mathcal{E}$, we obtain the following symplectic matrix:

$$U_Q \oplus W_Q := \mathcal{E} P_Q \mathcal{E}^{-1} = P_Q = \begin{bmatrix} Q & 0 \\ 0 & Q \end{bmatrix} \oplus [Q]. \quad (\text{A17})$$

The symplectic matrix U_Q corresponds to the permutation Q on the qubits. In this case, W_Q is not trivial and implements a corresponding permutation on the XZ block.

APPENDIX B: DECOMPOSITION OF SYMPLECTIC MATRIX INTO CIRCUIT FORM

In this appendix, we show how to decompose a symplectic matrix into a circuit involving H , CNOT, $C(X, X)$, $\sqrt{X}$, CZ, and S gates. We first set out our main theorem, which shows that such a decomposition exists, and then give an algorithm for obtaining the decomposition in practice.

1. Existence of circuit decomposition

In this section, we show that any symplectic matrix can be written as a product of four symplectic matrices—one corresponding to $C(X, X)$ and $\sqrt{X}$ gates, one corresponding to CZ and S gates, one corresponding to CNOT gates, and one corresponding to H gates.

Theorem B1 (Product form of symplectic matrix). Any symplectic matrix can be written in the form $U = U_A U_B U_C U_H$, where:

(1)

$$U_A := \left[\begin{array}{c|c} I & 0 \\ \hline A & I \end{array} \right],$$

for some symmetric $n \times n$ binary matrix A , is a set of $C(X, X)$ and $\sqrt{X}$ gates.

(2)

$$U_B := \left[\begin{array}{c|c} I & B \\ \hline 0 & I \end{array} \right],$$

for some symmetric $n \times n$ binary matrix B , is a set of CZ and S gates.

(3)

$$U_C := \left[\begin{array}{c|c} C & 0 \\ \hline 0 & C^{-T} \end{array} \right],$$

for some invertible $n \times n$ binary matrix C , is a set of CNOT gates.

(4)

$$U_H := \left[\begin{array}{c|c} \text{diag}(1 - \mathbf{h}) & \text{diag}(\mathbf{h}) \\ \hline \text{diag}(\mathbf{h}) & \text{diag}(1 - \mathbf{h}) \end{array} \right],$$

for some length n binary vector $\mathbf{h}$ corresponding to a set of H gates.

Proof. To solve for U_H , we follow the method outlined in Ref. [14, Lemma 6]. Let

$$U := \left[\begin{array}{c|c} C_0 & B_0 \\ \hline A_0 & C_1 \end{array} \right].$$

The rows of $[C_0|B_0]$ are independent, so if C_0 is of rank $r < n$, then, for some B_1 , C_2 , and B_2 of rank $n - r$, we have

the following reduced row echelon form (RREF)

$$\text{RREF}(C_0|B_0) = \left[\begin{array}{c|c} C_2 & B_1 \\ 0 & B_2 \end{array} \right]. \quad (\text{B1})$$

We now calculate $\text{RREF}(B_2)$ and form the binary vector $\mathbf{h}$ of length n , where the i th component of $\mathbf{h}$ is 1 if and only if i is a pivot when calculating $\text{RREF}(B_2)$. Following the reasoning of Ref. [14, Lemma 6], the upper-left-hand block of UU_H is invertible, so we set this to be C and form U_C . Applying the inverse

$$U_C^{-1} := \left[\begin{array}{c|c} C^{-1} & 0 \\ 0 & C^T \end{array} \right]$$

to the right-hand side of UU_H , for some A , B , and C_3 , we have

$$UU_H U_C^{-1} = \left[\begin{array}{c|c} I & B \\ A & C_3 \end{array} \right]. \quad (\text{B2})$$

This matrix is symplectic, so

$$\Omega = \left[\begin{array}{c|c} 0 & I \\ I & 0 \end{array} \right] = \left[\begin{array}{c|c} I & B \\ A & C_3 \end{array} \right] \Omega \left[\begin{array}{c|c} I & B \\ A & C_3 \end{array} \right]^T \quad (\text{B3})$$

$$= \left[\begin{array}{c|c} B & I \\ C_3 & A \end{array} \right] \left[\begin{array}{c|c} I & A^T \\ B^T & C_3^T \end{array} \right] \quad (\text{B4})$$

$$= \left[\begin{array}{c|c} B + B^T & BA^T + C_3^T \\ C_3 + AB^T & C_3 A^T + AC_3^T \end{array} \right]. \quad (\text{B5})$$

Hence $B + B^T = 0$ and so B is symmetric. The transpose of $UU_H U_C^{-1}$ is also symplectic and this implies that A is symmetric as well. We construct U_A and U_B , noting that these are self-inverse, and apply them to the right-hand side:

$$UU_H U_C^{-1} U_B U_A = \left[\begin{array}{c|c} I & B \\ A & C_3 \end{array} \right] \left[\begin{array}{c|c} I & B \\ 0 & I \end{array} \right] \left[\begin{array}{c|c} I & 0 \\ A & I \end{array} \right] \quad (\text{B6})$$

$$= \left[\begin{array}{c|c} I & B \\ A & C_3 \end{array} \right] \left[\begin{array}{c|c} I + BA & B \\ A & I \end{array} \right] \quad (\text{B7})$$

$$= \left[\begin{array}{c|c} I + BA + BA & B + B \\ A + ABA + C_3 A & AB + C_3 \end{array} \right] \quad (\text{B8})$$

$$= \left[\begin{array}{c|c} I & 0 \\ (I + AB + C_3)A & AB + C_3 \end{array} \right]. \quad (\text{B9})$$

This is a symplectic matrix and this implies that $AB + C_3 = I$ because, writing $D := (I + AB + C_3)A$ and

$$E := AB + C_3,$$

$$\Omega = \left[\begin{array}{c|c} 0 & I \\ I & 0 \end{array} \right] = \left[\begin{array}{c|c} I & 0 \\ D & E \end{array} \right] \Omega \left[\begin{array}{c|c} I & 0 \\ D & E \end{array} \right]^T \quad (\text{B10})$$

$$= \left[\begin{array}{c|c} 0 & I \\ E & D \end{array} \right] \left[\begin{array}{c|c} I & D^T \\ 0 & E^T \end{array} \right] \quad (\text{B11})$$

$$= \left[\begin{array}{c|c} 0 & E^T \\ E & ED^T + DE^T \end{array} \right]. \quad (\text{B12})$$

This in turn implies that $UU_H U_C^{-1} U_B U_A = I$ and so $U = U_A U_B U_C U_H$ as claimed. ■

2. Calculating the circuit decomposition

We now show how to efficiently calculate the circuit decomposition in practice for a symplectic matrix U . We first calculate $\mathbf{h}$ and U_H as described in Theorem B1, such that

$$UU_H = U_A U_B U_C \quad (\text{B13})$$

$$= \left[\begin{array}{c|c} I & 0 \\ A & I \end{array} \right] \left[\begin{array}{c|c} I & B \\ 0 & I \end{array} \right] \left[\begin{array}{c|c} C & 0 \\ 0 & C^{-T} \end{array} \right] \quad (\text{B14})$$

$$= \left[\begin{array}{c|c} I & B \\ A & I + AB \end{array} \right] \left[\begin{array}{c|c} C & 0 \\ 0 & C^{-T} \end{array} \right] \quad (\text{B15})$$

$$= \left[\begin{array}{c|c} C & BC^{-T} \\ AC & (I + AB)C^{-T} \end{array} \right]. \quad (\text{B16})$$

This matrix form gives C directly. We obtain A and B from the matrix form as follows:

$$A := (AC)C^{-1}, \quad (\text{B17})$$

$$B := (BC^{-T})C^T. \quad (\text{B18})$$

APPENDIX C: ALGORITHMS

In this appendix, we set out our key algorithms in more detail. In Algorithm 1, we describe our method for constructing a related binary linear code from a given stabilizer code, finding the automorphisms of the code and then constructing the corresponding symplectic matrices acting on the original code space. In Algorithm 2, we describe the algorithm for finding a Pauli correction to ensure that the phases of the stabilizer group are preserved. Finally, in Algorithm 3, we describe an algorithm for checking if a desired logical action can be implemented from a set of generating logical operators.

 ALGORITHM 1. Automorphisms of a binary matrix representation of a stabilizer code.

Input:

1. Representation of the stabilizer generators as the binary matrix $G := \begin{bmatrix} G_X & | & G_Z \end{bmatrix}$; and
2. Transformation matrix $\mathcal{E}$ corresponding to either a 2 or 3 block binary representation.

Output: A set of symplectic matrices $\bar{U}_S$ which generate the relevant group of Clifford automorphisms.

Method:

1. Form the binary matrix $G_{\mathcal{E}} := G\mathcal{E}$ for 2-block representations or $G_{\mathcal{E}} := (G|0)\mathcal{E}$ for 3-block representations;
 2. Let $B := \begin{bmatrix} I & I \end{bmatrix}$ for 2-block representations or $B := \begin{bmatrix} I & I & I \end{bmatrix}$ for 3-block representations;
 3. Find the generators of $\text{Aut}(\langle G_{\mathcal{E}} \rangle) \cap \text{Aut}(\langle B \rangle)$ where $\langle B \rangle$ is the binary linear code with generator matrix B , using a computational algebra program e.g. MAGMA;
 4. Each generator of $\text{Aut}(\langle G_{\mathcal{E}} \rangle) \cap \text{Aut}(\langle B \rangle)$ is a column permutation and has a matrix representation P . For two-block representations, the symplectic matrix $\bar{U}_{SP} := \mathcal{E}P\mathcal{E}^{-1}$; for 3-block representations we have a direct sum $\mathcal{E}P\mathcal{E}^{-1} = \bar{U}_S \oplus W$;
 5. Return the symplectic matrices $\bar{U}_S$ for each generator P of $\text{Aut}(\langle G_{\mathcal{E}} \rangle) \cap \text{Aut}(\langle B \rangle)$.
-

ALGORITHM 2. The Pauli correction and logical action of a Clifford operator.

Input:

1. A tableau $\tau := \begin{bmatrix} G \\ L_X \\ R \\ L_Z \end{bmatrix}$ for an $[[n, k]]$ stabilizer code such that $\tau\Omega\tau = \Omega$ (see Section II A 4);
2. A Clifford circuit $\bar{U}_S$ acting on n qubits;

Output:

1. If $\bar{U}_S$ is not a logical operator, return FALSE.
2. Otherwise, return a product of Pauli operators $\bar{U}_P$ which ensures that the signs of the stabilizer generators are preserved by $\bar{U} := \bar{U}_S\bar{U}_P$, plus a $2k \times 2k$ symplectic matrix U_{ACT} which describes the logical action of $\bar{U}$.

Method:

1. Set $\bar{U}_P := I$ and set U_{ACT} to be an empty list;
2. For each row $(\mathbf{x}_i|\mathbf{z}_i)$ of τ corresponding to a stabilizer, logical Pauli generator $i^{p_i}X(\mathbf{x}_i)Z(\mathbf{z}_i)$, calculate the mapped operator $\bar{U}_S(i^{p_i}X(\mathbf{x}_i)Z(\mathbf{z}_i))\bar{U}_S^\dagger := i^uX(\mathbf{x}')Z(\mathbf{z}')$ using the update rules of Table II;
3. Let $\mathbf{b} := (\mathbf{x}'|\mathbf{z}')\Omega\tau^T\Omega = (\mathbf{g}|\mathbf{a}_X|\mathbf{r}|\mathbf{a}_Z)$ as in Equation (23). This is a binary string of length $2n$ such that that $(\mathbf{x}'|\mathbf{z}') = \mathbf{b}\tau = \mathbf{g}G + \mathbf{a}_XL_X + \mathbf{r}R + \mathbf{a}_ZL_Z$;
4. If $\mathbf{r}$ is non-zero, then the mapped operator is the product of at least one destabilizer and so $\bar{U}_S$ does not preserve the codespace. Return FALSE;
5. If row i corresponds to a stabilizer and $(\mathbf{a}_X|\mathbf{a}_Z)$ is non-zero, then the mapped operator is the product of at least one logical Pauli generator and $\bar{U}_S$ does not preserve stabilizers. Return FALSE;
6. If row i corresponds to a logical Pauli, append the row $(\mathbf{a}_X|\mathbf{a}_Z)$ to U_{ACT} . This is a vector of length $2k$ corresponding to the logical Pauli generators appearing in the mapped logical Pauli;
7. Calculate the product $i^{-\mathbf{a}_X \cdot \mathbf{a}_Z} \prod_{0 \leq j < 2n} (i^{p_j}X(\mathbf{x}_j)Z(\mathbf{z}_j))^{\mathbf{b}[j]} := i^vX(\mathbf{x}')Z(\mathbf{z}')$, noting the phase adjustment $i^{-\mathbf{a}_X \cdot \mathbf{a}_Z}$;
8. Let $h := (j + n) \bmod 2n$. If $u \neq v \bmod 4$, multiply $\bar{U}_P$ by the anti-commuting operator $X(\mathbf{x}_h)Z(\mathbf{z}_h)$;
9. Return $U_{\text{ACT}}, \bar{U}_P$.

 ALGORITHM 3. Determining whether a logical operator can be implemented through automorphisms of the code.

Input:

1. Desired Clifford Logical operator U acting on k logical qubits of the code.
2. Logical Clifford operators U_{ACT} implemented by the automorphism group generators of the chosen stabilizer code representation.

Output: If U is in the group of logical operators generated by the action of the physical circuits obtained through automorphisms of the chosen representation, return a physical circuit $\bar{U}$ implementing it. Otherwise return FALSE.

Method:

1. Write U as an element of Sp_{2k} , the group of $2k \times 2k$ symplectic matrices.
 2. For each generator $a_i, 1 \leq i \leq l$, of the automorphism group of the chosen stabilizer representation, obtain its logical action U_{ACT_i} with the correct phases from Algorithm 2. This is a $2k \times 2k$ symplectic matrix.
 3. Form the group $\Lambda := \langle U_{\text{ACT}_1}, \dots, U_{\text{ACT}_l} \rangle \subseteq \text{Sp}_{2k}$.
 4. Check whether $U \in \Lambda$. If not, return FALSE and terminate.
 5. If $U \in \Lambda$, find its decomposition in terms of generators U_{ACT_i} through a presentation of Λ .
 6. Run Algorithm 2 again to find any potential Pauli corrections $\bar{U}_P$ to the above decomposition of U .
 7. Return the composition of the physical circuits $\bar{U}_i$ corresponding to each U_{ACT_i} in the decomposition of U and any Pauli corrections $\bar{U}_P$. This circuit implements a logical U .
-

APPENDIX D: EMBEDDED CODES

The methods described in Sec. III are able to identify logical operators that use SWAP and single-qubit physical gates. The embedded-code technique allows us to identify a wider range of logical operators that also use two-qubit CNOT and CZ physical gates. In this appendix, we set out the key results for constructing embedded codes and finding logical Clifford operators via permutation automorphisms of a related binary linear code. The structure of this appendix is as follows. We first describe the embedding operator, which is a Clifford unitary used to embed a stabilizer code into a larger Hilbert space. We then describe how to calculate the stabilizer generators of the embedded code by applying the embedding operator to Pauli operators. Finally, we show how to interpret the automorphisms of the embedded code as logical operators in the original code space.

1. Embedding-operator definition

Let M be a $m \times n$ binary matrix and let $V := \begin{bmatrix} I \\ M \end{bmatrix}$. The embedding operator $\mathcal{E}_V$ on $n + m$ qubits is defined as follows. Let $\mathbf{v}_j$ be row of V and $j \geq n$. Let $i < \mathbf{v}_j$ denote that

i is in the support of $\mathbf{v}_j$. Define

$$U_j := \prod_{i < \mathbf{v}_j} \text{CNOT}_{ij} \quad \text{and} \quad (\text{D1})$$

$$\mathcal{E}_V := \prod_{n \leq j < n+m} U_j. \quad (\text{D2})$$

Note that $\mathcal{E}_V$ is a Clifford unitary with $\mathcal{E}_V^{-1} = \mathcal{E}_V$. All terms CNOT_{ij} in the U_j commute because control qubits are never updated by the relevant operators, and as a result the U_j also commute. We can, e.g., select V to be of form $V := \begin{bmatrix} I_n \\ M_2^n \end{bmatrix}$, where M_2^n is the matrix the $m := \binom{n}{2}$ rows of which are the binary strings of length n and weight 2. In the example below, we see that this corresponds to adding auxiliary qubits that represent pairs of our original qubits.

Example D1. Let $n := 2$,

$$V := \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix},$$

so that $\mathbf{v}_2 = 11$. In this case, $\mathcal{E}_V = U_2 = \text{CNOT}_{02}\text{CNOT}_{12}$. Here, we calculate the action of U_2 on a computational

basis vector of form $|x_0, x_1, 0\rangle$. We have

$$\begin{aligned} U_2 |x_0, x_1, 0\rangle &= \text{CNOT}_{02} \text{CNOT}_{12} |x_0, x_1, 0\rangle \\ &= \text{CNOT}_{02} |x_0, x_1, x_1\rangle \\ &= |x_0, x_1, x_0 \oplus x_1\rangle. \end{aligned}$$

Hence, we can think of qubit 2 as the “sum” of qubits 0 and 1, provided that we initialize qubit 2 in the $|0\rangle$ state.

2. Embedded-code definition

Let $\mathbf{G}$ be a set of independent commuting Pauli operators on n qubits that define a stabilizer code (see Ref. [36]). For a given binary matrix $V := \begin{bmatrix} I \\ M \end{bmatrix}$, we extend the stabilizer code defined by the stabilizer generators $\mathbf{G}$ on n qubits to a stabilizer code on $m + n > n$ qubits by adding stabilizers Z_j for $n \leq j < n + m$. Adding the stabilizers Z_j is equivalent to adding auxiliary qubits initialized in the state $|0\rangle$. The stabilizers of the *embedded code* are as follows:

$$\mathbf{G}_V := \{\mathcal{E}_V A \mathcal{E}_V : A \in \mathbf{G}\} \cup \{\mathcal{E}_V Z_j \mathcal{E}_V : n \leq j < n + m\}. \quad (\text{D3})$$

3. Action of embedding operator

We now look at the action under conjugation of the embedding operator $\mathcal{E}_V$ on Pauli operators—this allows us to explicitly calculate the stabilizer generators $\mathbf{G}_V$ of the embedded code.

For *Pauli-Z operators*, the action of U_j is trivial unless the Z operator acts on qubit j . This is because Z operators commute with CNOT operators, unless they act on the target of the CNOT operator:

$$U_j Z_k U_j = Z_k (Z(\mathbf{v}_j)^{\delta_{jk}}) = \begin{cases} Z_k Z(\mathbf{v}_j) : k = j, \\ Z_k : \text{otherwise.} \end{cases} \quad (\text{D4})$$

For *Pauli-X operators*, the action of U_j is trivial unless the X operator acts on a qubit in the support of $\mathbf{v}_j$. This is because X operators commute with CNOT operators, unless they act on the control of the CNOT operator:

$$U_j X_k U_j = X_k (X_j^{\mathbf{v}_j[k]}) = \begin{cases} X_k X_j : k < \mathbf{v}_j, \\ X_k : \text{otherwise.} \end{cases} \quad (\text{D5})$$

For a string of Pauli- X operators $X(\mathbf{x})$ for $\mathbf{x}$ a binary matrix of length n , the action of $\mathcal{E}_V$ is given by $\mathcal{E}_V(X(\mathbf{x})) =$

$X(\mathbf{x}V^T)$. To see this, we first note that

$$\begin{aligned} U_j X(\mathbf{x}) U_j &= \prod_{0 \leq k < n} (U_j X_k^{\mathbf{x}[k]} U_j) \\ &= \prod_{0 \leq k < n} (X_k^{\mathbf{x}[k]} X_j^{\mathbf{x}[k]\mathbf{v}_j[k]}) \\ &= X(\mathbf{x}) X_j^{\mathbf{x}\cdot\mathbf{v}_j} \end{aligned}$$

and so

$$\begin{aligned} \mathcal{E}_V(X(\mathbf{x})) &= \left(\prod_{n \leq j < n+m} U_j \right) X(\mathbf{x}) \left(\prod_{n \leq j < n+m} U_j \right) \\ &= X(\mathbf{x}) \prod_{n \leq j < n+m} X_j^{\mathbf{x}\cdot\mathbf{v}_j} \\ &= X(\mathbf{x}V^T). \end{aligned}$$

As a result, we can write the action of the embedding operator on computational basis elements as follows:

$$\mathcal{E}_V|\mathbf{x}\rangle = \mathcal{E}_V(X(\mathbf{x})|0\rangle) = |\mathbf{x}V^T\rangle.$$

Applying Eqs. (D5) and (D6) to Eq. (D3) yields the form of the check matrix of the embedded code as set out in Eq. (29):

$$G_V := \left[\begin{array}{cc|cc} G_X & G_X M^T & G_Z & 0 \\ 0 & 0 & M & I \end{array} \right]. \quad (\text{D6})$$

4. Automorphisms of embedded codes

We now show how to interpret the automorphisms of an embedded code as gates acting on the original code space. The automorphisms on the embedded code space are single-qubit Cliffords and SWAP gates and we show how each of these translate to the original code space. Let L_V be a Clifford automorphism of the embedded code so that the following diagram commutes:

$$\begin{array}{ccc} \mathcal{H}_2^n \otimes \mathcal{H}_2^m & \xrightarrow{\mathcal{E}_V} & \mathcal{H}_2^{n+m} \\ L \otimes W \downarrow & & \downarrow L_V \\ \mathcal{H}_2^n \otimes \mathcal{H}_2^m & \xrightarrow{\mathcal{E}_V} & \mathcal{H}_2^{n+m} \end{array}$$

In the above diagram, the operator W is some (possibly trivial) Clifford operator that preserves the group generated by Z_i for $n \leq i < n + m$. We can determine the operator L on the original code space by calculating $L \otimes W = \mathcal{E}_V L_V \mathcal{E}_V$. By construction, Z_j for $n \leq j < n + m$ are stabilizers of the original code space and any automorphism must preserve these stabilizers. We will find L by studying the action of $\mathcal{E}_V L_V \mathcal{E}_V$ on computational basis vectors and mapping the action back to a physical gate.

5. Phase operators on embedded code space

We consider the action of a phase operator $L_V := S_j$ on the embedded code space where $n \leq j < n + m$. Without loss of generality, let $U_j := \text{CNOT}_{0j} \text{CNOT}_{1j}$ and consider the action of $U_j S_j U_j$ on a computational basis state of form $|x_0, x_1, 0\rangle$:

$$L \otimes W |x_0, x_1, 0\rangle = U_j S_j U_j |x_0, x_1, 0\rangle \quad (\text{D7})$$

$$= U_j S_j |x_0, x_1, x_0 \oplus x_1\rangle \quad (\text{D8})$$

$$= i^{x_0 \oplus x_1} U_j |x_0, x_1, x_0 \oplus x_1\rangle \quad (\text{D9})$$

$$= i^{x_0 \oplus x_1} |x_0, x_1, 0\rangle. \quad (\text{D10})$$

The operator with this action is the phase-rotation operator $L = RS_{01} = S_0 S_1 CZ_{01}$, as defined in Ref. [10].

6. Hadamard operators on embedded code space

We now consider the action of an H operator $L_V := H_j$ on the embedded code space where $n \leq j < n + m$:

$$L \otimes W |x_0, x_1, 0\rangle = U_j H_j U_j |x_0, x_1, 0\rangle \quad (\text{D11})$$

$$= U_j H_j |x_0, x_1, x_0 \oplus x_1\rangle \quad (\text{D12})$$

$$= \frac{1}{\sqrt{2}} U_j (|x_0, x_1, 0\rangle + (-1)^{x_0 \oplus x_1} |x_0, x_1, 1\rangle) \quad (\text{D13})$$

$$= \frac{1}{\sqrt{2}} (|x_0, x_1, x_0 \oplus x_1\rangle + (-1)^{x_0 \oplus x_1} |x_0, x_1, 1 \oplus x_0 \oplus x_1\rangle). \quad (\text{D14})$$

There is no operator of the form $L \otimes W$ that produces this result, because it involves a superposition of states stabilized by $\pm Z_j$, so L_V cannot be a valid automorphism.

7. SWAP operators on embedded code space

We now consider the action of a SWAP operator $L_V := \text{SWAP}_{1j}$ on the embedded code space where $n \leq j < n + m$ and both 0, 1 are in the support of $\mathbf{v}_j$:

$$L \otimes W |x_0, x_1, 0\rangle = U_j \text{SWAP}_{1j} U_j |x_0, x_1, 0\rangle \quad (\text{D15})$$

$$= U_j \text{SWAP}_{1j} |x_0, x_1, x_0 \oplus x_1\rangle \quad (\text{D16})$$

$$= U_j |x_0, x_0 \oplus x_1, x_1\rangle \quad (\text{D17})$$

$$= |x_0, x_0 \oplus x_1, 0\rangle. \quad (\text{D18})$$

The operator on the original code space with this action is $L = \text{CNOT}_{01}$.

We may also encounter a situation in which the SWAP operator is between qubit j and a qubit k not in the support

of U_j . In this case, we see that

$$L \otimes W |x_0, x_1, x_k, 0\rangle = U_j \text{SWAP}_{kj} U_j |x_0, x_1, x_k, 0\rangle \quad (\text{D19})$$

$$= U_j \text{SWAP}_{kj} |x_0, x_1, x_k, x_0 \oplus x_1\rangle \quad (\text{D20})$$

$$= U_j |x_0, x_1, x_0 \oplus x_1, x_k\rangle$$

$$= |x_0, x_1, x_0 \oplus x_1, x_0 \oplus x_1 \oplus x_k\rangle. \quad (\text{D21})$$

For $L_V := \text{SWAP}_{kj}$ to be a valid automorphism and preserve the stabilizer Z_j , we require that $x_0 \oplus x_1 \oplus x_k = 0$ or $x_k = x_0 \oplus x_1$, in which case L acts as a logical identity on the original code space.

-
- [1] D. Gottesman, Theory of fault-tolerant quantum computation, *Phys. Rev. A* **57**, 127 (1998).
 - [2] B. Eastin and E. Knill, Restrictions on transversal encoded quantum gate sets, *Phys. Rev. Lett.* **102**, 110502 (2009).
 - [3] R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane, in *Proceedings of IEEE International Symposium on Information Theory* (IEEE, Ulm, Germany, 1997), p. 292.
 - [4] J. E. Moussa, Transversal Clifford gates on folded surface codes, *Phys. Rev. A* **94**, 042316 (2016).
 - [5] N. P. Breuckmann and S. Burton, Fold-transversal Clifford gates for quantum codes, *Quantum* **8**, 1372 (2024).
 - [6] A. O. Quintavalle, P. Webster, and M. Vasmer, Partitioning qubits in hypergraph product codes to implement logical gates, *Quantum* **7**, 1153 (2023).
 - [7] W. Bosma, J. Cannon, and C. Playoust, The MAGMA algebra system I: The user language, *J. Symb. Comput.* **24**, 235 (1997).
 - [8] T. Junttila and P. Kaski, in *Theory and Practice of Algorithms in (Computer) Systems*, edited by A. Marchetti-Spaccamela and M. Segal (Springer-Verlag, Berlin, 2011), pp. 151–162.
 - [9] See <https://github.com/hsayginel/autqec>. Our results are available as an interactive notebook at <https://github.com/hsayginel/autqec/tree/main/examples>. Results for $k = 1$ best-known distance codes are available at <https://github.com/hsayginel/autqec/tree/main/codetables> and results for bivariate bicycle codes are available as an interactive notebook at https://github.com/hsayginel/autqec/tree/main/examples/bivariate_bicycle_codes.
 - [10] M. A. Webster, A. O. Quintavalle, and S. D. Bartlett, Transversal diagonal logical operators for stabiliser codes, *New J. Phys.* **25**, 103018 (2023).
 - [11] M. Grassl, Bounds on the minimum distance of linear codes and quantum codes, Available online at <http://www.codetables.de> (2007).
 - [12] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, High-threshold and low-overhead fault-tolerant quantum memory, *Nature* **627**, 778 (2024).
 - [13] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition* (Cambridge University Press, Cambridge, 2010). ISBN: 9780511976667.

- [14] S. Aaronson and D. Gottesman, Improved simulation of stabilizer circuits, *Phys. Rev. A* **70**, 052328 (2004).
- [15] D. Schoenberger, S. Hillmich, M. Brandl, and R. Wille, Shuttling for scalable trapped-ion quantum computers, *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.* **44**, 2144 (2024).
- [16] S. J. Evered, D. Bluvstein, M. Kalinowski, S. Ebadi, T. Manovitz, H. Zhou, S. H. Li, A. A. Geim, T. T. Wang, N. Maskara, H. Levine, G. Semeghini, M. Greiner, V. Vuletić, and M. D. Lukin, High-fidelity parallel entangling gates on a neutral-atom quantum computer, *Nature* **622**, 268 (2023).
- [17] Y. Xiao, Y.-H. Kang, R.-H. Zheng, J. Song, Y.-H. Chen, and Y. Xia, Effective nonadiabatic holonomic swap gate with Rydberg atoms using invariant-based reverse engineering, *Phys. Rev. A* **109**, 062610 (2024).
- [18] D. Kielpinski, C. Monroe, and D. J. Wineland, Architecture for a large-scale ion-trap quantum computer, *Nature* **417**, 709 (2002).
- [19] T. A. Baart, M. Shafiei, T. Fujita, C. Reichl, W. Wegscheider, and L. M. K. Vandersypen, Single-spin CCD, *Nat. Nanotechnol.* **11**, 330 (2016).
- [20] T. J. Yoder, R. Takagi, and I. L. Chuang, Universal fault-tolerant gates on concatenated stabilizer codes, *Phys. Rev. X* **6**, 031039 (2016).
- [21] A. G. Manes and J. Claes, Distance-preserving stabilizer measurements in hypergraph product codes, *Quantum* **9**, 1618 (2025).
- [22] J. Leon, Computing automorphism groups of error-correcting codes, *IEEE Trans. Inf. Theory* **28**, 496 (1982).
- [23] M. Grassl and M. Roetteler, in *2013 IEEE International Symposium on Information Theory* (2013), pp. 534–538, iSSN: 2157-8117.
- [24] B. Zeng, A. Cross, and I. L. Chuang, Transversality versus universality for additive quantum codes, *IEEE Trans. Inf. Theory* **57**, 6272 (2011).
- [25] M. P. da Silva *et al.*, Demonstration of logical qubits and repeated error correction with better-than-physical error rates, [arXiv:2404.02280](https://arxiv.org/abs/2404.02280) [quant-ph].
- [26] A. A. Kovalev and L. P. Pryadko, Quantum Kronecker sum-product low-density parity-check codes with finite rate, *Phys. Rev. A* **88**, 012311 (2013).
- [27] D. MacKay, G. Mitchison, and P. McFadden, Sparse-graph codes for quantum error correction, *IEEE Trans. Inf. Theory* **50**, 2315 (2004).
- [28] J. N. Eberhardt and V. Steffan, Logical operators and fold-transversal gates of bivariate bicycle codes, *IEEE Trans. Inf. Theory* **71**, 1140 (2025).
- [29] A. Cross, Z. He, P. Rall, and T. Yoder, Linear-size ancilla systems for logical measurements in QLDPC codes, [arXiv:2407.18393](https://arxiv.org/abs/2407.18393) [quant-ph].
- [30] L. Babai, in *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16 (Association for Computing Machinery, New York, 2016), pp. 684–697.
- [31] L. Babai, in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019 (Association for Computing Machinery, New York, 2019), pp. 1237–1246.
- [32] N. Sendrier and G. Skersys, in *Proceedings. 2001 IEEE International Symposium on Information Theory (IEEE Cat. No. 01CH37252)* (2001), p. 13.
- [33] B. D. McKay and A. Piperno, Practical graph isomorphism, II, *J. Symb. Comput.* **60**, 94 (2014).
- [34] Z. He, A. Cowtan, D. J. Williamson, and T. J. Yoder, Extractors: QLDPC architectures for efficient Pauli-based computation, [arXiv:2503.10390](https://arxiv.org/abs/2503.10390) [quant-ph].
- [35] S. Koutsoumpas, H. Sayginel, M. A. Webster, and D. E. Browne, Automorphism ensemble decoding of quantum LDPC codes, [arXiv:2503.01738](https://arxiv.org/abs/2503.01738).
- [36] D. E. Gottesman, *Stabilizer Codes and Quantum Error Correction*, Ph.D. thesis, California Institute of Technology, Pasadena, 1997, <https://thesis.library.caltech.edu/2900/2/THESIS.pdf>.