Computing Efficiently in QLDPC Codes

Alexander J. Malcolm,[1] Andrew N. Glaudell,[1] Patricio Fuentes,[1] Daryus Chandra,[1] Alexis Schotte,[1] Colby DeLisle,[1] Rafael Haenel,[1] Amir Ebrahimi,[1] Joschka Roffe,[1,2] Armanda O. Quintavalle,[1,3] Stefanie J. Beale,[1] Nicholas R. Lee-Hone,[1] and Stephanie Simmons[1]

[1] *Photonic Inc.*
[2] *University of Edinburgh, United Kingdom*
[3] *Freie Universität Berlin, Germany*

(Dated: February 11, 2025)

It is the prevailing belief that quantum error correcting techniques will be required to build a utility-scale quantum computer able to perform computations that are out of reach of classical computers. The quantum error correcting codes that have been most extensively studied and therefore highly optimized, surface codes, are extremely resource intensive in terms of the number of physical qubits needed. A promising alternative, quantum low-density parity check (QLDPC) codes, has been proposed more recently. These codes are much less resource intensive, requiring up to 10x fewer physical qubits per logical qubit than practical surface code implementations. A successful application of QLDPC codes would therefore drastically reduce the timeline to reaching quantum computers that can run algorithms with proven exponential speedups like Shor's algorithm and QPE. However to date QLDPC codes have been predominantly studied in the context of quantum memories; there has been no known method for implementing arbitrary logical Clifford operators in a QLDPC code proven efficient in terms of circuit depth. In combination with known methods for implementing T gates, an efficient implementation of the Clifford group unlocks resource-efficient universal quantum computation. In this paper, we introduce a new family of QLDPC codes that enable efficient compilation of the full Clifford group via transversal operations. Our construction executes any m-qubit Clifford operation in at most $O(m)$ syndrome extraction rounds, significantly surpassing state-of-the-art lattice surgery methods. We run circuit-level simulations of depth-126 logical circuits to show that logical operations in our QLDPC codes attains near-memory performance. These results demonstrate that QLDPC codes are a viable means to reduce, by up to 10x, the resources required to implement all logical quantum algorithms, thereby unlocking a much reduced timeline to commercially valuable quantum computing.

I. INTRODUCTION

Quantum computers are poised to deliver the next major evolution in computational technology, with known applications in several high-impact sectors, including drug discovery, materials design, and cryptanalysis. However, by their nature, quantum technologies are highly susceptible to noise and are inherently incompatible with classical error correction techniques, leading to the development of unique quantum-specific error correction solutions. In quantum error correction (QEC), physical redundancy is introduced so that errors can be detected and corrected, ideally without harming the encoded information. QEC researchers have spent decades optimizing so-called "planar" QEC codes, such as the surface code. These codes have many positive attributes, including nearest-neighbour connectivity for syndrome extraction, competitive error thresholds, and a high degree of symmetry. However, the physical resource requirements of the surface code are of surface code error correction are so onerous that experts have assessed that with these codes quantum technologies capable of breaking RSA-2048 (a benchmark often used to assess commercial utility) are likely to only arrive decades in the future [1]. Until efficient QEC is unlocked at scale, commercial quantum applications will not deliver material value to society.

Tantalizingly, there are other QEC codes –known as quantum low-density parity check (QLDPC) codes [2]– that do not suffer from such high physical overheads. These codes have many of the same positive traits that planar codes do, leading to an explosion of interest in them over the past few years [3–11]. However, despite significant work in this direction [6, 9], it is not presently known how to compile depth-efficient logical operations in QLDPC codes. If provably efficient universal logical gate sets were to be found for QLDPC codes, the timeline for the availability of commercially relevant quantum computers could be brought in by years if not decades, owing to the reduced cost in physical qubit count.

In this paper, we propose QLDPC codes, *Subsystem HYpergraph Product Simplex (SHYPS) codes*, designed from the bottom up with logical operator implementation as the core consideration. We construct highly symmetric codes capable of implementing immense numbers of logical gates transversally [12], facilitating arbitrary logic with asymptotic circuit depths matching those for *unencoded* logic. This approach is generally compatible with single-shot error correction, providing benefits in reduced qubit-overhead, reduced decoding complexity, and faster logical clock speeds. We demonstrate this compatibility by performing the most advanced circuit-level logical simulation to date, demonstrating near-memory performance for a compiled circuit derived from a randomly sampled logical operator on 18 logical qubits.

II. LOGIC IN QLDPC CODES

The Clifford group is of particular interest for quantum computation as it provides a universal gate set when combined with any single non-Clifford operator. As there are known methods for implementing non-Clifford operations (e.g., T gates) fault-tolerantly using state injection [13], an efficient fault-tolerant implementation of the Clifford group for a given code is sufficient to unlock universal computation in that code.

To date, QLDPC codes have been predominantly studied in the context of quantum memories. Various techniques based on generalized lattice surgery [7, 10] have been proposed to rework logical Clifford implementations in planar codes for application in QLDPC codes. However, these methods often introduce substantial overheads in both time and space: operations are not guaranteed to be parallelizable, large auxiliary patches are required, and any single-shot properties [11] of the QLDPC code can be compromised, as lattice surgery does not inherently support them.

In the generalized lattice surgery framework, computation is performed via joint logical operator measurements. State-of-the-art schemes for QLDPC surgery [6] measure single weight-d logical operators with overheads of $O(d)$ in both space and time for a single logical cycle, where d is the code distance. A constant number of such measurements may then be combined to implement elementary Clifford gates such as CNOT, Hadamard and Phase. It may be possible that compiling methods will be developed to optimize depth-efficiency for these techniques. However, even assuming a high degree of parallelization commensurate with planar codes –which may or may not be possible in general– direct compilation with these elementary gates to implement a worst-case m-qubit Clifford operator requires $O(m)$ logical cycles, leading to a total depth of $O(md)$.

Recent research [8] suggests that logical Clifford execution based on transversal gates may avoid the $O(d)$ overhead for a single logical cycle, allowing syndrome information to accrue simultaneously with logical Clifford execution. This indicates that a QLDPC code family with sufficiently many transversal gates that also exhibits single-shot capabilities might execute arbitrary Clifford logic in lower depths with efficient decoding strategies. We generate a new family of highly symmetric QLDPC codes –*Subsystem HYpergraph Product Simplex* (SHYPS) codes– by combining the subsystem hypergraph product (SHP) code construction [14] with classical simplex codes [15, Ch. 1.9], and show that for these codes such low-depth logical Clifford implementations are indeed possible. The SHYPS code family permits implementations of elementary Clifford gates in $O(1)$ time with typically zero space overhead (see Table I). However, the chief figure of merit we analyze is the depth of a worst-case encoded Clifford.

Using new compilation techniques, we achieve a depth-efficient implementation of a worst-case m-qubit Clifford in only $4m(1 + o(1))$ logical cycles, each consisting of a single depth-1 circuit followed by a depth-6 syndrome extraction round. This is comparable to state-of-the-art depth-efficiency, $2m + O(log^2(m))$, for a purely unitary implementation of Cliffords acting on unencoded qubits [16]. The compilation strategy employed to achieve this low-depth implementation constructs circuits not as a sequence of elementary one- and two-qubit logical operations, but rather by leveraging many-qubit logical operations that occur naturally via low physical depth logical generators. Circuit-level simulations of these compiled circuits show near-memory performance (see Fig. 2), demonstrating the feasibility of time-efficient logical execution in QLDPC codes.

III. COMPILING AND COSTS

When computing in an error correction setting, logical operations are interleaved with rounds of syndrome extraction [17]. The operators are typically drawn from a subset of the full set of logical operations, and we will refer to elements of this subset as *logical generators*. A sequence of logical generators combine to synthesize a logical operation that the circuit performs. To compute efficiently in a QEC code, a generating set of (ideally low-depth) logical generators is needed. The size of this generating set relative to the space of logical operations it generates informs the worst-case number of generators needed to implement an arbitrary logical operation, and therefore the number of syndrome extraction steps that need to be interleaved between them. Since syndrome extraction is costly and lengthier computations are less desirable, the aim of compiling is to reduce the number of steps needed to implement a desired logical operator. This reduction can be achieved by constructing codes which have more native low-depth logical generators. We restrict attention to the space of the m-qubit Clifford group, and examine the notion of efficient compilation in that context to quantify the number of logical generators needed to compute efficiently.

The size of the m-qubit Clifford group scales as $2^{2m^2+O(m)}$ [18] (see also Section IV A), which tends to pose a problem for compiling with a number of logical generators that doesn't also grow at least exponentially in m. Consider the case where we work with γ fixed-depth logical generators of the m-qubit Clifford group. The set of all circuits composed of up to depth D of these logical generators cannot produce more than $(\gamma+1)^D$ unique Clifford operators. Ignoring the action of the Pauli group (all Paulis can be pushed to the end and implemented via a depth-1 circuit in stabilizer codes), this means we need D to be such that

$$(\gamma + 1)^D \geq 2^{2m^2+O(m)}$$

to possibly produce every Clifford operator. Rearranging, this lower bounds the required depth to attain an

arbitrary operation in the Clifford group:

$$D \geq \frac{2m^2 + m - 1}{\log_2(\gamma + 1)} =: D^*. \quad (1)$$

We can estimate that the fraction of total Clifford operators achievable with all depths $D < D^*$ is $1/(\gamma + 1)$. Since we generally consider cases where γ is a monotonically increasing function of m, asymptotically we expect $1/(\gamma + 1) \to 0$ so that *almost all* Cliffords require depth *at least* D^*.

Consider synthesizing m-qubit Clifford operators using depth-1 circuits of arbitrary one- and two-qubit Clifford gates. These circuits are fixed under conjugation by the Clifford subgroup $\langle S, H, \mathrm{SWAP}\rangle$, and a counting exercise demonstrates that there are $\gamma \leq e/\sqrt{\pi}\,(10m/e)^{m/2} - 1$ nontrivial depth-1 circuits, up to the right action of this subgroup. Substituting this expression into Eq. (1) yields an asymptotic worst-case (and average-case) depth lower bound of $D \geq 4m/\log_2(10m/e) + O(1)$. While there are indeed compiling algorithms that achieve depth $O(m/\log_2 m)$ asymptotically [19], the leading constants are impractically large [16]. Rather than accept these large overheads, researchers have derived alternative decompositions that achieve better depths in practice for realistic qubit counts with worst-case depths of $O(m)$ [12, 16, 20, 21].

We employ an alternative decomposition (see Supplementary Material XII), which focuses on minimizing the rounds of specific subsets of Clifford gates. Every Clifford operator can be written in the four-stage decomposition DZ−CX−DX−DZ(1) [22], where we have defined subsets of the Clifford group as follows:

- DZ $= \langle S, CZ\rangle$; Z-diagonal Cliffords,
- CX $= \langle CNOT\rangle$; The CNOT group,
- DX; X-diagonal Cliffords (Hadamard-rotated DZ),
- DZ(1); depth-1 Z-diagonal operations.

Combined with a few additional insights [16], any of the listed decompositions yield circuits with asymptotic depths of $2m+O(\log^2 m)$, which are state of the art for realistic qubit counts. Compiling worst case logical circuits on m patches of surface code with all-to-all connectivity for an arbitrary m-qubit logical Clifford will thus require roughly $2m$ logical cycles.

Having established via study of decompositions a worst-case depth for a logical Clifford in the surface code, we can now revisit Eq. (1) and consider the number of logical Clifford generators we require for a different error correcting code to achieve similar efficiency. Clearly, $\gamma \sim 2^{O(m)}$ is necessary to achieve a logical cycle depth of $O(m)$. However, note that because CSS codes always have transversal CNOT operations available as logical generators, when using large numbers of code blocks of a CSS code, we will always satisfy this requirement. This holds because the number of ways to pair up code blocks to apply cross-block operations scales exponentially in the number of code blocks (and hence the total number of logical qubits). Instead, we would like to capture the compiling behavior in code families at both the "few" code block scale and the "many" code block scale.

For b code blocks of an $[n, k, d]$ code [23], the *Clifford compiling ratio*

$$\frac{\text{depth of a worst case Clifford on } b \cdot k \text{ logical qubits}}{b \cdot k}$$

captures compiling properties for an arbitrary number of code blocks. Any code that achieves an $O(1)$ ratio is said to generate the logical Clifford group *efficiently*. A code family with parameters $[n(r), k(r), d(r)]$ for which every member generates the logical Clifford group efficiently is also said to have this feature. Any code family whose associated compiling is such that the Clifford compiling ratio scales with $k(r)$ is failing to keep pace with the surface code due to the depth of logical operations *within* a code block, whereas scaling with b is associated with overheads for compiling *between* code blocks.

Motivated by the advantages of computing in a code that has many fault-tolerant logical generators of low depth, we introduce the SHYPS code family in Section V. This family has $\gamma = 2^{O(k)}$ logical generators for a *single* code block, each implemented by a depth-1 physical circuit. Moreover, these logical generator implementations often require 0 additional qubits, rising to at most n for in-block CNOT operators where a scheme involving an auxiliary code block is used [24].

In addition to possessing a sufficient number of logical generators, the SHYPS codes achieve the desired $O(1)$ Clifford compiling ratio, with logical Clifford operators across b blocks implemented fault-tolerantly in depth at most $4bk(1 + o(1))$ (see Table I). Crucially, the depth of Clifford operations compiled in our SHYPS code framework remains independent of the code distance, compared to state-of-the-art lattice surgery methods where the depth scales as $O(md)$. For a moderately sized code with distance $d = 20$, an SHYPS-compiled CNOT gate would achieve an order-of-magnitude reduction in depth relative to the equivalent compiled using lattice surgery (4 vs. roughly 40). This example highlights that –in addition to reducing qubit overheads relative to the surface code– an SHYPS-based quantum computer would provide substantially faster clock-speeds at the logical level.

Logical Gate	**Time cost**	**Space cost**
CNOT (cross-block)	4	0
CNOT (in-block)	4	n
S (in-block)	6	0
CZ (cross/in-block)	4	0
H (in-block)	8	0
Arbitrary b-block Clifford	$4bk(1 + o(1))$	bn or 0 [24]

Table I. Time and space costs for logical Clifford operations of $SHYPS(r)$ codes ($r \geq 4$) with parameters $[n, k]$. For $r = 3$, the logical S and H gates have depths 9 and 11, respectively. Here, time corresponds to the number of complete syndrome extraction rounds.

IV. CLIFFORD OPERATORS AND AUTOMORPHISMS

A. Symplectic representations

The Clifford group $\mathcal{C}_n$ is a collection of unitary operators that maps the Pauli group $\mathcal{P}_n$ to itself, under conjugation. For example, the two-qubit controlled-not operator $CNOT_{i,j}$, the single-qubit phase gate S_i, and the single-qubit Hadamard gate H_i, are all Clifford operators, and in fact these suffice to generate the full group. When considering logical operators of codes, it is convenient to utilise the well-known symplectic representation of Clifford operators: by definition $\mathcal{P}_n$ is a normal subgroup of $\mathcal{C}_n$ and the quotient $\mathcal{C}_n/\mathcal{P}_n$ is isomorphic to $Sp_{2n}(2)$, the group of $2n \times 2n$ binary symplectic matrices [18, Thm. 15]. Hence each Clifford operator is, up to Pauli, specified by a unique element $g \in Sp_{2n}(2)$. That this representation ignores Pauli factors is of no concern, as any logical Pauli may be implemented transversally.

The following examples illustrate the symplectic representation of some common families of Clifford operators; note that by convention we assume that elements of $Sp_{2n}(2)$ act on row vectors from the right.

Example IV.1. **(CNOT circuits)** The collection of CNOT circuits $\langle CNOT_{i,j} : 1 \leq i, j \leq n \rangle$ have symplectic representations

$$\left\{ \begin{bmatrix} C & 0 \\ 0 & C^{-T} \end{bmatrix} : C \in GL_n(2) \right\},$$

where $GL_n(2)$ is the group of invertible $n \times n$ binary matrices.

Example IV.2. **(Diagonal Clifford operators)** The Clifford operators that act diagonally on the computational basis form an abelian group, generated by single-qubit phase gates S_i and the two-qubit controlled-Z gate $CZ_{i,j}$. They are represented by symplectic matrices of the form

$$\left\{ \begin{bmatrix} I_n & B \\ 0 & I_n \end{bmatrix} : B \in M_n(2), B^T = B \right\},$$

where the diagonal and off-diagonal entries of the symmetric matrix B, determine the presence of S and CZ gates, respectively.

B. Code automorphisms

Code automorphisms are permutations of the physical qubits that preserve the codespace. They are a promising foundation for computing in QLDPC codes as they can provide nontrivial logical operators implementable by low-depth SWAP circuits, or simply relabelling physical qubits. Moreover, combining automorphisms with additional transversal Clifford gates can give greater access to fault-tolerant logical operator implementations [3, 4, 9].

Let $\mathcal{C}$ be an $[n, k, d]$ CSS code with X- and Z-type gauge generators determined by matrices $G_X \in \mathbb{F}_2^{r_X \times n}$ and $G_Z \in \mathbb{F}_2^{r_Z \times n}$, respectively. The *(permutation) automorphism group* $\mathrm{Aut}(\mathcal{C})$ is the collection of permutations $\pi \in S_n$ that preserve the gauge generators, and therefore the codespace. I.e., $\pi \in \mathrm{Aut}(\mathcal{C})$ if there exist $g_{\pi,X} \in GL_{r_X}(2)$ and $g_{\pi,Z} \in GL_{r_Z}(2)$ such that $g_{\pi,X} G_X = G_X \pi$ and $g_{\pi,Z} G_Z = G_Z \pi$.

The logical operator implemented by a given $\pi \in \mathrm{Aut}(\mathcal{C})$ is determined by its action on the code's logical Paulis. In particular, as permutations preserve the X/Z-type of a Pauli operator, π implements a logical CNOT circuit [25, Thm. 2]. Furthermore, following [25], a larger set of fault-tolerant CNOT circuits across two copies of $\mathcal{C}$ may be derived by conjugating the target block of the standard transversal CNOT operator [26] by π (see Supplementary Materials Fig. 3).

The symplectic representation for this combined operator is given by

$$\begin{bmatrix} I_n & \pi & & \\ 0 & I_n & & \\ & & I_n & 0 \\ & & \pi^{-1} & I_n \end{bmatrix} \in Sp_{4n}(2), \tag{2}$$

where we identify $\pi \in \mathrm{Aut}(\mathcal{C})$ with the permutation matrix in $GL_n(2)$ whose (i, j)-th entry is 1 if $i = \pi(j)$, and zero otherwise. Note that as conjugation by π simply permutes the targets of the physical transversal CNOT, this is a fault-tolerant circuit of depth 1. Moreover, (2) implements a *cross-block* CNOT operator, with all controls in the first code block of k logical qubits, and all targets in the latter code block.

More recently, code automorphisms have been generalized to include qubit permutations that exchange vectors in G_X and G_Z. These so-called *ZX-dualities* lead to low-depth logical operator implementations involving qubit permutations and single qubit Hadamard gates [3]. Moreover, ZX-dualities allow for the construction of logical diagonal Clifford operators in the following manner: Let $\tau \in S_n$ be an involution ($\tau^2 = 1$) such that $G_X \tau = G_Z$, and suppose that $\pi \in \mathrm{Aut}(\mathcal{C})$ is such that $\pi\tau$ is also an involution. Then the physical diagonal Clifford operator given by

$$\begin{bmatrix} I_n & \pi\tau \\ 0 & I_n \end{bmatrix} \in Sp_{2n}(2), \tag{3}$$

is a depth-1 circuit that implements a logical diagonal Clifford operator up to Pauli correction (see [3, 4] and Supplementary Materials Lem. X.2). As there is always a Pauli operator with the appropriate (anti)commutation relations with the stabilizers and logical operators of the code to fix any logical/stabilizer action sign issues, we can ignore this subtlety [12].

The requirement that $\pi\tau$ is an involution guarantees that the upper-right block of (3) is symmetric, and thus corresponds to a valid diagonal Clifford operator. This generally restricts the number of automorphisms that may be leveraged to produce valid logical operators. Crucially, this is insignificant for the SHYPS codes we intro-

duce in Section V, where we have sufficient symmetry to efficiently implement all diagonal operators in a code block.

V. CODE CONSTRUCTIONS AND LOGICAL OPERATORS

The constructions (2) and (3) provide a framework for implementing logical Clifford operators with fault-tolerant, low-depth circuits. However the number of such operators that exist for a given subsystem CSS code $\mathcal{C}$ clearly scales with the size of $\mathrm{Aut}(\mathcal{C})$. This motivates a search for quantum codes with high degrees of permutation symmetry, to achieve the number of fixed-depth Clifford generators necessary for efficient compilation. There are many methods for constructing quantum codes [27–33], but in this work we focus on a subsystem hypergraph product construction that allows us to leverage known highly symmetric classical codes, to produce quantum code automorphisms: Let H_i be parity check matrix matrices for two classical (n_i, k_i, d_i)-codes with $i = 1, 2$ and codespace $\ker H_i$. The *subsystem hypergraph product code* [14] (SHP), denoted $SHP(H_1, H_2)$, is the subsystem CSS code with gauge generators

$$G_X = (H_1 \otimes I_{n_2}),\ G_Z = (I_{n_1} \otimes H_2),$$

and parameters $[n_1 n_2, k_1 k_2, \min(d_1, d_2)]$ [14, 3.B].

Now the classical codes $\ker H_i$ have analogously defined automorphism groups, and crucially these *lift* to distinct automorphisms of $SHP(H_1, H_2)$.

Lemma V.1. *Let* $(\sigma_1, \sigma_2) \in \mathrm{Aut}(\ker H_1) \times \mathrm{Aut}(\ker H_2)$. *Then* $\sigma_1 \otimes \sigma_2 \in \mathrm{Aut}(SHP(H_1, H_2))$.

To capitalise on this, we pair the SHP construction with the highly symmetric classical simplex codes, referring to these as *subsystem hypergraph product simplex* (SHYPS) codes. A complete description of the SHYPS family (parameterized by integers $r \geq 3$) is given in Supplementary Materials VIII E but we note here that each instance, denoted $SHYPS(r)$ has parameters

$$[n(r), k(r), d(r)] = [(2^r - 1)^2, r^2, 2^{r-1}].$$

Moreover, this is a QLDPC code family, as each $SHYPS(r)$ has weight-3 gauge generators. An immediate corollary of Lemma V.1 and [15, Ch. 8.5] is that

$$|\mathrm{Aut}(SHYPS(r))| \geq |GL_r(2)|^2 = O(2^{2r^2}),$$

which grows exponentially in the number of logical qubits $k(r) = r^2$, as required in Section III. By utilising these automorphisms with the constructions outlined in Section IV B we are able to efficiently generate all CNOT and diagonal Clifford operators in the SHYPS codes.

The logical action of operators (2) and (3) may be characterized explicitly: For any pair $g_1, g_2 \in GL_r(2)$ there exists a corresponding automorphism $\sigma_1 \otimes \sigma_2 \in \mathrm{Aut}(SHYPS(r))$ such that the logical cross-block CNOT operator

$$\begin{bmatrix} I_k & g_1 \otimes g_2 & & \\ 0 & I_k & & \\ & & I_k & 0 \\ & & g_1^{-T} \otimes g_2^{-T} & I_k \end{bmatrix} \in Sp_{4k}(2), \tag{4}$$

is implemented by the depth-1 physical circuit of type (2). Furthermore, arbitrary logical CNOT circuits on b blocks of k logical qubits can be constructed from a sequence of $2bk(1 + o(1))$ such operators (see Supplementary Materials Thm. IX.3 and Cor. IX.21).

To characterize the logical action of diagonal operators (3) we first observe that the physical qubits of $SHYPS(r)$ may be naturally arranged in an $2^r - 1 \times 2^r - 1$ array such that the reflection across the diagonal is a ZX-duality exchanging G_X and G_Z (see Supplementary Materials Lem. X.2). We similarly arrange the logical qubits in an $r \times r$ array, and denote the reflection that exchanges rows and columns by τ. Then for all $g \in GL_r(2)$, there exists $\sigma \otimes \sigma^T \in \mathrm{Aut}(SHYPS(r))$ such that the logical diagonal Clifford operator

$$\begin{bmatrix} I & (g \otimes g^T) \cdot \tau \\ 0 & I \end{bmatrix} \tag{5}$$

is implemented by a corresponding generator of type (3). The operators (5) have depth 1 and are fault-tolerant with circuit distance equal to the code distance. Moreover, they are alone sufficient to generate all logical diagonal Clifford operators on an SHYPS code block in depth at most $k(1+o(1))$ (Supplementary Materials Thm. X.6).

For generation of the full logical Clifford group, we note that $SHYPS(r)$ possesses a *Hadamard type* [3] fold-transversal gate whereby the logical all-qubit Hadamard operator (up to logical SWAP) is implemented fault-tolerantly (see Supplementary Materials Lem. X.19). By then applying a Clifford decomposition as discussed in Section III (see also Supplementary Materials XII), we bound the cost of implementing an arbitrary Clifford operator:

Theorem V.2. *Let* $r \geq 3$. *An arbitrary logical Clifford operator on* b *blocks of the* $SHYPS(r)$ *code may be implemented fault-tolerantly in depth* $4bk(r)(1 + o(1))$.

In particular, the SHYPS codes achieve the desired $O(1)$ Clifford compiling ratio. Moreover this bound is competitive with best known depths of $2bk + O(log^2(bk))$ for compiling Cliffords on bk unencoded qubits [16]. We achieve further reductions in overhead for logical permutations and arbitrary Hadamard circuits (see Supplementary Materials Tables II, III and IV for details).

VI. PERFORMANCE OF THE SHYPS CODE

We present numerical simulations to evaluate the circuit-level noise performance of the SHYPS code family. Two types of simulations were performed. First, in

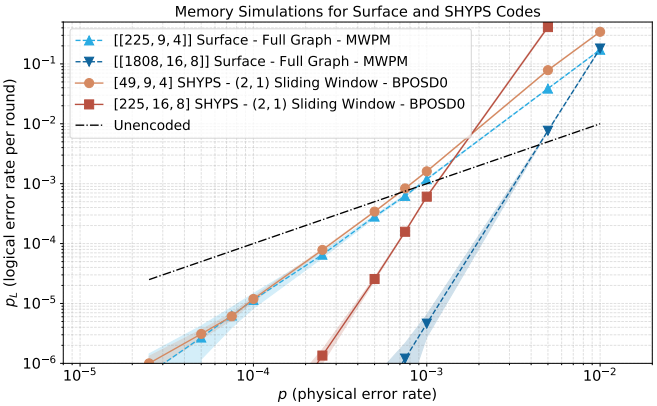


Figure 1. Simulation results for quantum memories under circuit-level noise for SHYPS and surface codes. For these simulations, only Z-type detectors are used.

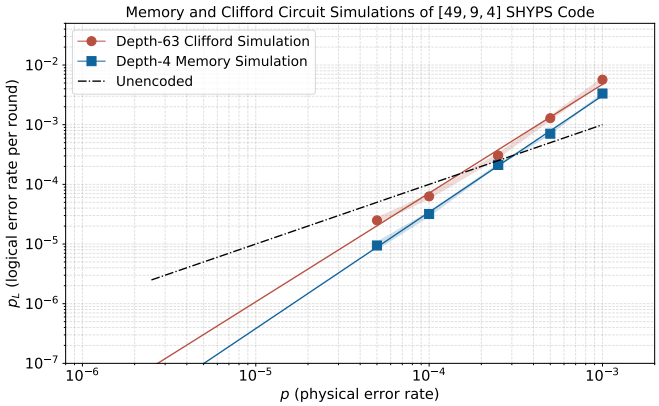


Figure 2. Simulation results for a depth-126 logical quantum circuit and a memory on two code blocks of the $[49, 9, 4]$ SHYPS code. Z- and X-type detectors are used in both cases.

section VI A, memory simulations for two different instances of SHYPS codes are benchmarked against comparably scaled surface codes. Second, in section VI B, we present a circuit-level noise logic simulation on two blocks of the $[49, 9, 4]$ SHYPS code. The logical operators are randomly sampled from the 18 qubit Clifford group and synthesized into low-depth generators using the techniques explained in Section V. For a more detailed treatment of the simulations presented in this section, see Supplementary Materials XIII.

A. Memory Simulation

Figure 1 shows the normalized logical error rate p_L from memory simulations with d syndrome extraction rounds. The logical error rates for the scaled surface codes are adjusted according to the number of logical qubits [34] to demonstrate that the $[49, 9, 4]$ SHYPS code is competitive with the surface code of the same distance while requiring fewer physical qubits per logical qubit. The pseudo-threshold of the $[49, 9, 4]$ SHYPS code matches that of the $[[225, 9, 4]]$ scaled distance-4 surface code, approximately at $p = 0.08\%$, in addition to using only 1/5 of the physical qubits. In addition, the $[225, 16, 8]$ SHYPS code outperforms the $[[225, 9, 4]]$ surface code both in pseudo-threshold and error correction slope for an equivalent number of physical qubits. Performance of the $[225, 16, 8]$ SHYPS code is comparable to that of the $[[1808, 16, 8]]$ scaled distance-8 surface code, albeit with a lower pseudo-threshold. We anticipate that a decoder specifically tailored to SHYPS codes would further improve the pseudo-threshold.

Most impressively, the SHYPS code simulations rely on a sliding window decoding approach with a small window size (2) and commit size (1). The use of a (2,1) sliding window decoder to achieve a high error correction performance is consistent with single-shot properties –the ability to decode based on a single syndrome extraction round between each logical operation.

B. Clifford Simulation

We now apply our decomposition and logical generator constructions in a simulation of a randomly sampled Clifford operator on 18 logical qubits in two code blocks of the $[49, 9, 4]$ SHYPS code, up to an in-block CNOT circuit. This prevents the need to use extra auxiliary code blocks to implement in-block CNOTs while ensuring all types of transversal operations appear in the simulation. Synthesizing the sampled Clifford using the $\mathrm{DZ} - \mathrm{CX} - \mathrm{DX} - \mathrm{DZ}(1)$ decomposition requires 63 fault-tolerant logical generators, and consequently a total of 126 logical generators to implement both the Clifford circuit and its inverse. The simulation proceeds as follows: initialize both code blocks in the encoded all-zero logical state; interleave each logical generator (both of the Clifford and its inverse) with gauge generator measurements; read out the state using a transversal Z measurement.

Figure 2 shows the normalized logical error rate p_L for the described simulation. Decoding with a $(3, 1)$ sliding window decoder attains beyond break-even performance with a pseudo-threshold of $p = 0.0185\%$. We also include the logical error rate per syndrome extraction round for a d-round SHYPS quantum memory simulation with 18 logical qubits. Simulation of the Clifford operator achieves near-memory error correction performance in terms of error suppression, demonstrating that logical operations can be executed efficiently and fault-tolerantly in SHYPS codes.

VII. CONCLUSION

QLDPC codes promise to reduce physical qubit overheads compared to existing surface codes. Despite their advantage as memories, it has been unclear whether any multi-logical qubit code (QLDPC or otherwise) could execute logical operations with as much parallelism as a surface code. Were this to remain unresolved, it may have crippled the speed of any QLDPC-code-based quantum computer when running highly-parallel circuits. By using a product construction of highly-symmetric classical simplex codes alongside novel compiling methods, we

have shown that the resultant SHYPS code family shatters this barrier, achieving the same asymptotic logical circuit depth as *unencoded* circuits under state-of-the-art algorithms. Remarkably, the resulting logical circuits retain strong fault-tolerance guarantees that are reflected in deep logical simulations showing near-memory performance even under circuit-level noise.

Two critical directions demand further exploration. Developing QEC codes with analogous properties but even better rates would enable further space improvements without making a time trade-off. More significantly, extending compiling parallelism to measurement parallelism would unlock for QLDPC codes every trick the surface code has at its disposal for reducing run-times via auxiliary code blocks. Without these advantages in parallelism, the reasons to consider using the surface code over QLDPC codes reduces to two factors: connectivity and simplicity. For any architecture where the necessary connectivity is achievable, it now seems all but certain: QLDPC codes are capable of driving down physical overheads *without* increasing time overheads, and, as a result of this work, appear to be the most compelling path to quantum computers that can perform commercially relevant algorithms.

[1] M. Mosca and M. Piani, *2022 Quantum Threat Timeline Report*, Tech. Rep.

[2] Note that throughout this paper we follow the convention of the field and implicitly exclude surface codes from consideration when discussing QLDPC codes, instead focusing on codes with higher encoding rate that meet the LDPC criteria.

[3] N. P. Breuckmann and S. Burton, Quantum **8**, 1372 (2024).

[4] J. N. Eberhardt and V. Steffan, Logical operators and fold-transversal gates of bivariate bicycle codes (2024), arXiv:2407.03973 [quant-ph].

[5] A. Gong, S. Cammerer, and J. M. Renes, Toward low-latency iterative decoding of QLDPC codes under circuit-level noise (2024), arXiv:2403.18901 [quant-ph].

[6] A. Cross, Z. He, P. Rall, and T. Yoder, Improved QLDPC Surgery: Logical Measurements and Bridging Codes (2024), arXiv:2407.18393 [quant-ph].

[7] A. Cowtan and S. Burton, Quantum **8**, 1344 (2024).

[8] H. Zhou, C. Zhao, M. Cain, D. Bluvstein, C. Duckering, H.-Y. Hu, S.-T. Wang, A. Kubica, and M. D. Lukin, Algorithmic fault tolerance for fast quantum computing (2024), arXiv:2406.17653 [quant-ph].

[9] A. O. Quintavalle, P. Webster, and M. Vasmer, Quantum **7**, 1153 (2023).

[10] L. Z. Cohen, I. H. Kim, S. D. Bartlett, and B. J. Brown, Science Advances **8**, eabn1717 (2022), 2110.10794.

[11] A. O. Quintavalle, M. Vasmer, J. Roffe, and E. T. Campbell, PRX Quantum **2**, 020340 (2021).

[12] H. Sayginel, S. Koutsioumpas, M. Webster, A. Rajput, and D. E. Browne, Fault-Tolerant Logical Clifford Gates from Code Automorphisms (2024), arXiv:2409.18175 [quant-ph].

[13] D. Litinski, Quantum **3**, 205 (2019).

[14] M. Li and T. J. Yoder, in *Proceedings of IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, 2020) pp. 109–119.

[15] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*, 2nd ed. (North-holland Publishing Company, 1978).

[16] D. Maslov and B. Zindorf, IEEE Transactions on Quantum Engineering **3**, 1 (2022).

[17] Syndrome extraction results inform error correction, and corrections are often tracked in software and folded into the operators applied later in the circuit. For the purposes of this discussion, we restrict attention to syndrome extraction as this is the piece that is costly in terms of operations applied to qubits.

[18] N. Rengaswamy, R. Calderbank, H. D. Pfister, and S. Kadhe, in *Proceedings of IEEE International Symposium on Information Theory (ISIT)* (IEEE, 2018) pp. 791–795.

[19] J. Jiang, X. Sun, S.-H. Teng, B. Wu, K. Wu, and J. Zhang, in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (SIAM, 2020) pp. 213–229.

[20] S. Bravyi and D. Maslov, IEEE Transactions on Information Theory **67**, 4546 (2021).

[21] R. Duncan, A. Kissinger, S. Perdrix, and J. Van De Wetering, Quantum **4**, 279 (2020).

[22] The order of DZ(1), CX, DX can be exchanged to any of its six permutations up to exchanging DZ(1) with DX(1) and/or moving those layers from the front to the back so that there are really 12 related decompositions. This freedom in reordering allows compilers to leverage interaction with neighbouring T gates effectively.

[23] Throughout this paper, we adopt the following notation for error-correcting codes to ensure clarity: (n, k, d) for classical codes, $[n, k, d]$ for subsystem codes, and $[[n, k, d]]$ for stabilizer codes.

[24] There exist methods to remove the need for this additional auxiliary block with the same asymptotic cost, but the actual circuit length tends to be larger in the low code block regime.

[25] M. Grassl and M. Roetteler, in *Proceedings of IEEE International Symposium on Information Theory (ISIT)* (IEEE, 2013) pp. 534–538.

[26] The physical transversal CNOT operator implements logical transversal CNOT in any subsystem CSS code [35, Sec. 5].

[27] A. R. Calderbank and P. W. Shor, Phys. Rev. A **54**, 1098 (1996).

[28] J.-P. Tillich and G. Zémor, IEEE Transactions on Information Theory **60**, 1193 (2013).

[29] P. Panteleev and G. Kalachev, IEEE Transactions on Information Theory **68**, 213 (2021).

[30] N. P. Breuckmann and J. N. Eberhardt, PRX Quantum **2**, 10.1103/prxquantum.2.040101 (2021).

[31] N. P. Breuckmann and J. N. Eberhardt, IEEE Transactions on Information Theory **67**, 6653 (2021).

[32] P. Panteleev and G. Kalachev, in *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Com-*

puting (ACM, 2022) pp. 375–388.
[33] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, Nature **627**, 778–782 (2024).
[34] Note that this is equivalent to considering multiple distance-d surface code patches required to provide the same number of logical qubits as the distance d SHYPS code.
[35] P. W. Shor, Fault-tolerant quantum computation (1997), arXiv:quant-ph/9605011 [quant-ph].

END NOTES

Acknowledgements

We thank Polina Bychkova, Zach Schaller, Bogdan Reznychenko, and Kyle Wamer for their contributions to the development of simulation infrastructure.

Author contributions

A.J.M. and A.N.G. designed the quantum codes studied in this manuscript. The depth costing of logical operators was done by A.J.M., A.S. and A.N.G.. D.C., P.F., J.R. and A.O.Q. designed the sliding window decoder and optimized decoder parameters used in the circuit-level numerical simulations. P.F., D.C., J.R., A.O.Q., and A.N.G. performed the numerical simulations and post-processed the resulting data. The software used for these simulations was designed and written by C.D., R.H., A.E., P.F., J.R., and D.C.. A.J.M., A.N.G., P.F., D.C., A.S., J.R., A.O.Q., S.J.B., N.R.L.-H. and S.S. contributed to writing and editing the manuscript.

Competing interests

US Provisional Patent Applications 63/670,626 and 63/670,620 (filed on 12 July 2024, naming A.J.M. and A.N.G. as co-inventors), and US Provisional Patent Application 63/720,973 (filed on November 15, 2024, naming A.N.G. and A.S as co-inventors) contain technical aspects from this paper.

Additional information

Supplementary Information is available for this paper. Correspondence and requests for materials should be addressed to Stephanie Simmons at ssimmons@photonic.com.

Supplementary Material

CONTENTS

The supplementary material is broken into 3 parts. First, in Section VIII we survey the necessary background information to introduce our new code family, the *subsystem hypergraph product simplex codes* (SHYPS) codes. In particular, we discuss the notion of code automorphisms, and how the subsystem hypergraph product construction yields highly symmetric quantum codes, from well chosen classical inputs.

Next, in Sections IX-XI we demonstrate how automorphisms of the SHYPS codes can be leveraged to obtain low-depth implementations of logical Clifford operators. Taking each of the CNOT, diagonal, and Hadamard-SWAP families in turn, we produce fault-tolerant generators (typically physical depth 1), and demonstrate efficient compilation of arbitrary operators. Utilising a novel decomposition of Clifford operators, these results are combined in Section XII to yield an overarching bound on the depth of Clifford implementations in SHYPS codes. We refer the reader to Tables II, III and IV, for a detailed summary of results.

Lastly, in Section XIII we discuss syndrome extraction and decoding of the SHYPS codes, and we provide details about our numerical simulations.

VIII. MATHEMATICAL PRELIMINARIES AND CODE CONSTRUCTIONS

We begin the preliminaries section with a review of the Pauli and Clifford groups, including the binary symplectic representation and examples of key types of Clifford operations that will be the focus of later sections.

A. Review of Paulis and Cliffords

The *Pauli group* on n qubits is defined as

$$\begin{aligned}\mathcal{P}_n &:= \langle \mathrm{i}, X_j, Z_j \mid j \in \{1,\dots,n\}\rangle \\ &= \langle X_j, Y_j, Z_j \mid j \in \{1,\dots,n\}\rangle.\end{aligned}$$

For many applications, it is convenient to ignore global phases involved in Pauli operators and instead consider elements of the phaseless Pauli group, $\mathcal{P}_n/\langle \mathrm{i}\rangle$. The phaseless Pauli group is abelian and has order 4^n. Additionally, every non-trivial element has order 2, and hence $\mathcal{P}_n/\langle \mathrm{i}\rangle \cong \mathbb{F}_2^{2n}$. This isomorphism can be made explicit in the following manner: as $XZ = \mathrm{i}Y$, any $P \in \mathcal{P}_n/\langle \mathrm{i}\rangle$ may be written uniquely as

$$\prod_{i=1}^{n} X_i^{u_i} Z_i^{v_i} =: X^u Z^v,$$

where $u = (u_1,\dots,u_n) \in \mathbb{F}_2^n$, and v is defined analogously. The combined vector $(u \mid v) \in \mathbb{F}_2^{2n}$ is known as the *binary symplectic representation* of P. We equip $\mathbb{F}_2^{2n}$ with a symplectic form $\omega : \mathbb{F}_2^{2n} \times \mathbb{F}_2^{2n} \longrightarrow \mathbb{F}_2$ such that for $(u \mid v), (s \mid t) \in \mathbb{F}_2^{2n}$,

$$(u \mid v), (s \mid t) \mapsto ut^T + sv^T.$$

This form captures the (anti-)commutativity of Paulis (that is lost when global phases are ignored), as $X^u Z^v$ and $X^s Z^t$ anti-commute if and only if

$$\omega((u \mid v), (s \mid t)) = 1.$$

For example in $\mathcal{P}_2$, the element X_1 anti-commutes with $Y_1 Z_2$, and we check that

$$\begin{aligned} X_1 &\mapsto (1,0,0,0) \\ Y_1 Z_2 \sim X_1 Z_1 \cdot Z_2 &\mapsto (1,0,1,1),\end{aligned}$$

and

$$\omega((1,0,0,0),(1,0,1,1)) = 1.$$

The *Clifford group* on n qubits, denoted Clif_n, is the normalizer of the n-qubit Pauli group within $U(n)$. I.e.,

$$\begin{aligned}\mathrm{Clif}_n &:= N_{U(n)}(\mathcal{P}_n) \\ &= \left\{ g \in U(n) \,:\, gPg^{-1} \in \mathcal{P}_n \text{ for all } P \in \mathcal{P}_n \right\}.\end{aligned}$$

For example, the two-qubit controlled-not operator $CNOT_{i,j}$, the single-qubit phase gate S_i, and the single-qubit Hadamard gate H_i, are all Clifford operators. Further examples include all gates of the form $e^{\mathrm{i}\theta} I$, but as the global phase is typically unimportant we restrict our attention to

$$\mathcal{C}_n := \mathrm{Clif}_n / \{e^{\mathrm{i}\theta}\} \times \left\langle \frac{1+\mathrm{i}}{\sqrt{2}} \right\rangle.$$

We likewise refer to this as the Clifford group, and note that

$$\mathcal{C}_n = \left\langle CNOT_{i,j}, S_i, H_i \,:\, 1 \le i,j \le n \right\rangle.$$

The action of a Clifford operator via conjugation corresponds to a linear transformation of $\mathbb{F}_2^{2n}$ in the binary symplectic representation. Moreover, as conjugation preserves the (anti-)commutation of of Pauli operators, the corresponding linear transformations preserve the symplectic form. The collection of such linear transformations is known as the *symplectic group* and is denoted $Sp_{2n}(2)$. Finally, observe that $\mathcal{P}_n \le \mathcal{C}_n$ but that conjugation by a Pauli operator induces at most a change of sign, which is ignored by the binary symplectic representation. Taking the quotient by this trivial action, we see that $\mathcal{C}_n/\mathcal{P}_n \cong Sp_{2n}(2)$ [1, Thm. 15].

The representation of Clifford operators by $2n \times 2n$ binary matrices is key to the efficient simulation of stabilizer circuits [2]. Moreover, as we'll demonstrate, it is a useful framework for synthesising efficient implementations of logical operators in quantum codes (see also [1]).

We conclude this section with an explicit construction of some symplectic representations for Clifford operators. Following [2], we assume that matrices in $Sp_{2n}(2)$ act on row vectors from the right. I.e., given $P \in \mathcal{P}_n$ with binary symplectic representation $(u \mid v)$, and $g \in \mathcal{C}_n/\mathcal{P}_n$ with binary symplectic representation G, we have $gPg^{-1} \longleftrightarrow (u \mid v)G$. In particular, the images of the Pauli basis X_i, Z_i, are given by the ith and $(i+n)$th rows of G, respectively.

Example VIII.1. **(CNOT circuits)** As CNOT circuits map X-type Paulis to other X-type Paulis (and similar for Z-type), in $Sp_{2n}(2)$ they have the form

$$\left\{ \begin{pmatrix} C & 0 \\ 0 & C^{-T} \end{pmatrix} \,:\, C \in GL_n(2) \right\}.$$

For simplicity we typically describe such operators solely by the matrix C.

For example, $CNOT_{1,2} \in \mathcal{C}_3$ has C defined as follows

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \in GL_3(2).$$

Example VIII.2. **(Diagonal Clifford operators)** The Clifford operators that act diagonally on the computational basis form an abelian group, generated by single-qubit phase gates S_i and the two-qubit controlled-Z gate $CZ_{i,j}$. Hence, modulo Paulis, they are represented by symplectic matrices of the form

$$\left\{ \begin{pmatrix} I & B \\ 0 & I \end{pmatrix} \,:\, B \in M_k(2), B^T = B \right\},$$

where the diagonal and off-diagonal entries of the symmetric matrix B determine the presence of S and CZ gates, respectively.

For example, $S_1 \cdot CZ_{1,2} \in \mathcal{C}_2$ corresponds to

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \in Sp_4(2).$$

And the action of this example on a Pauli, X_1,

$$(S_1 \cdot CZ_{1,2}) \cdot X_1 \cdot (S_1 \cdot CZ_{1,2})^{-1} = Y_1 Z_2$$

corresponds to

$$(1,0,0,0) \cdot \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = (1,0,1,1).$$

Example VIII.3. **(Hadamard circuits)** In $Sp_{2n}(2)$, Hadamard circuits $H^v = \prod H_i^{v_i}$ have the form

$$\{\begin{pmatrix} I_n + \mathrm{diag}(v) & \mathrm{diag}(v) \\ \mathrm{diag}(v) & I_n + \mathrm{diag}(v) \end{pmatrix} \,:\, v \in \mathbb{F}_2^n\},$$

where $\mathrm{diag}(v)$ is the diagonal matrix with entries $v_1, \ldots, v_n$.

For example, the transversal Hadamard operator $H^{\otimes 3} = H^{(1,1,1)} \in \mathcal{C}_3$ corresponds to

$$\begin{pmatrix} 0 & I_3 \\ I_3 & 0 \end{pmatrix} \in Sp_6(2).$$

B. Subsystem codes

A quantum stabilizer code on n physical qubits is the common $+1$ eigenspace of a chosen abelian subgroup $\mathcal{S} \leq \mathcal{P}_n$ with $-I \notin \mathcal{S}$. The subgroup $\mathcal{S}$ is known as the *stabilizer group*, and moreover if $\mathcal{S}$ admits a set of generators that are either X-type or Z-type Pauli strings, then the code is called *CSS* [3]. Quantum subsystem codes are the natural generalisation of stabilizer codes, in that they are defined with respect to a generic subgroup $\mathcal{G} \leq \mathcal{P}_n$ known as the *gauge group* [4]. Moreover, subsystem codes are typically interpreted as the subsystem of a larger stabilizer code whereby a subset of the logical qubits are chosen to not store information and the action of the corresponding logical operators is ignored. More formally, given gauge group $\mathcal{G}$, the corresponding stabilizer group $\mathcal{S}$ is the centre of $\mathcal{G}$ modulo phases

$$\langle \mathcal{S}, iI \rangle = Z(\mathcal{G}) := C_{\mathcal{P}_n}(\mathcal{G}) \cap \mathcal{G}.$$

Phases are purposefully excluded (in particular, $-I \notin \mathcal{S}$) to ensure the fixed point space of $\mathcal{S}$ is nontrivial, and said space decomposes into a tensor product $\mathcal{C}_\mathcal{L} \otimes \mathcal{C}_\mathcal{G}$, where elements of $\mathcal{G} \backslash \mathcal{S}$ fix only $\mathcal{C}_\mathcal{L}$. The subspaces $\mathcal{C}_\mathcal{L}$ and $\mathcal{C}_\mathcal{G}$ are said to contain the logical qubits, and gauge qubits, respectively.

The logical operators of the subsystem code are differentiated into two types, based on their action on $\mathcal{C}_\mathcal{L} \otimes \mathcal{C}_\mathcal{G}$: those given by $C_{\mathcal{P}_n}(\mathcal{G}) \backslash \mathcal{G}$ that act nontrivially only on $\mathcal{C}_\mathcal{L}$ are known as *bare* logical operators. Whereas operators acting nontrivially on both $\mathcal{C}_\mathcal{L}$ and $\mathcal{C}_\mathcal{G}$ are known as *dressed* logical operators, and are given by $C_{\mathcal{P}_n}(\mathcal{S}) \backslash \mathcal{G}$. Note that a dressed logical operator is a bare logical operator multiplied by an element of $\mathcal{G} \backslash \mathcal{S}$. The minimum distance d of the subsystem code is the minimal weight Pauli that acts nontrivially on the logical qubits $\mathcal{C}_\mathcal{L}$, i.e., the minimum weight of a dressed logical operator

$$d = \min\{|P| \,:\, P \in C_{\mathcal{P}_n}(\mathcal{S}) \backslash \mathcal{G}\}.$$

We say that a subsystem code is an $[n, k, d]$ code if it uses n physical qubits to encode k logical qubits with distance d. The notation $[n, k, g, d]$ that additionally indicates the number of gauge qubits g, is also commonplace (but unused throughout this paper).

The advantages of subsystem codes are most evident when the gauge group $\mathcal{G}$ has a generating set composed of low-weight operators but the stabilizer group $\mathcal{S}$ consists of high-weight Paulis. Measuring the former operators requires circuits of lower depth (therefore reducing computational overheads), and the measurement results can be aggregated to infer the stabilizer eigenvalues. In fact, for the codes considered here, the difference in the stabilizer weights and gauge operator weights is a factor of the code distance (see Section XIII for more details). We note that it is exactly the measurement of operators in $\mathcal{G} \backslash \mathcal{S}$ that act nontrivially on $\mathcal{C}_\mathcal{G}$, that prevents the gauge qubits from storing information during computation, as these measurements impact the state of $\mathcal{C}_\mathcal{G}$.

In this work we restrict our attention to $[n, k, d]$-subsystem codes that are also of CSS type, with X- and Z-type gauge generators determined by matrices $G_X \in \mathbb{F}_2^{r_X \times n}$ and $G_Z \in \mathbb{F}_2^{r_Z \times n}$, respectively. Here each vector $v \in \mathbb{F}_2^n \cap \mathrm{RowSpace}(G_X)$ denotes an X-type gauge operator X^v, with Z-type gauge generators similarly defined. The associated stabilizers will also be of CSS type, and denoted by S_X, S_Z.

C. Automorphisms of codes

Code automorphisms are a promising foundation for computing in QLDPC codes as they can provide nontrivial logical operators implementable by permuting, or in practice simply relabelling, physical qubits. In this section, we review permutation automorphisms of classical and quantum codes, which will serve as the backbone of our logical operation constructions.

Let's first set some notation: given a permutation $\sigma \in S_n$ of the symbols $\{1, 2, \ldots, n\}$ in cycle notation, we identify σ with the permutation matrix whose (i, j)th

entry is 1 if $i = \sigma(j)$, and zero otherwise. For example

$$(1,2,3)(4,5) \in S_5 \mapsto \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

The permutation defined above maps the initial indices $\{1,2,3,4,5\}$ to $\{2,3,1,5,4\}$. So given a standard basis vector $e_i \in \mathbb{F}_2^n$, σ acts on row vectors on the right as $e_i\sigma = e_{\sigma^{-1}(i)}$, and on column vectors on the left as $\sigma e_i^T = e_{\sigma(i)}^T$.

Definition VIII.4. Let C be an (n,k,d)-classical linear code described by a generator matrix $G \in \mathbb{F}_2^{k\times n}$. Then the *(permutation) automorphism group* $\mathrm{Aut}(C)$ is the collection of permutations $\sigma \in S_n$ that preserve the codespace. I.e., $\sigma \in \mathrm{Aut}(C)$ if for all $c = (c_1, \ldots, c_n) \in C$,

$$c\sigma = (c_{\sigma(1)}, \ldots, c_{\sigma(n)}) \in C. \tag{6}$$

As C is linear, it suffices to check (6) on the basis given by the rows of G. Hence $\sigma \in \mathrm{Aut}(C)$ if and only if there exists a corresponding $g_\sigma \in GL_k(2)$ such that $G\sigma = g_\sigma G$ [5, Lem. 8.12]. In particular, g_σ represents the invertible linear transformation of the k logical bits, induced by the permutation.

The parity checks of the code C are similarly transformed by automorphisms: letting $C^\perp$ denote the dual classical code, we have $\mathrm{Aut}(C) = \mathrm{Aut}(C^\perp)$ [5, Sec. 8.5]. So given a parity check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ for C we have equivalently that $\sigma \in \mathrm{Aut}(C)$ if there exists $h_\sigma \in GL_{n-k}(2)$ such that $H\sigma = h_\sigma H$.

This definition extends naturally to quantum codes:

Definition VIII.5. Let C be an $[n,k,d]$ (CSS) subsystem code with X and Z type gauge generators determined by $G_X \in \mathbb{F}_2^{r_X\times n}$ and $G_Z \in \mathbb{F}_2^{r_Z\times n}$, respectively. Then $\mathrm{Aut}(C)$ consists of permutations that preserve the gauge generators, i.e., $\sigma \in S_n$ such that

$$g_{\sigma,X}G_X = G_X\sigma, \;\; g_{\sigma,Z}G_Z = G_Z\sigma,$$

for some $g_{\sigma,X} \in GL_{r_X}(2)$ and $g_{\sigma,Z} \in GL_{r_Z}(2)$.

The logical operator implemented by $\sigma \in \mathrm{Aut}(C)$ is determined by the permutation action on the code's logical Pauli operators. In particular, σ always gives rise to a permutation of the logical computational basis states of C corresponding to a logical CNOT circuit [6, Thm. 2].

In later sections we outline quantum constructions utilising classical codes, and consequently how classical automorphisms may be leveraged to produce quantum logical operators. In these instances, the logical CNOT implemented by the quantum code automorphism is a function of the associated classical linear transformations. This motivates an investigation of classical codes with high degrees of symmetry.

D. Classical simplex codes

Let $r \geq 3$ and define $n_r = 2^r - 1$, $d_r = 2^{r-1}$. The classical simplex codes, denoted $C(r)$ are a family of (n_r, r, d_r)-linear codes, that are dual to the well-known Hamming codes. More specifically, we consider $C(r)$ with respect to a particular choice of parity check matrix: for each $3 \leq r < 500$ [7], there exists a three term polynomial $h(x) = 1 + x^a + x^b \in \mathbb{F}_2[x]/\langle x^{n_r} - 1\rangle$ such that $\gcd(h(x), x^{n_r} - 1)$ is a primitive polynomial of degree r [8]. Then the $n_r \times n_r$ matrix

$$H = \begin{pmatrix} h(x) \\ xh(x) \\ \vdots \\ x^{n_r-1}h(x) \end{pmatrix}$$

is a parity check matrix (PCM) for $C(r)$ [5, Lem. 7.5], where here we adopt the usual polynomial notation for cyclic matrices

$$\sum_{i=0}^{n_r-1} a_i x^i \mod x^{n_r} - 1 \mapsto (a_0, a_1, \ldots, a_{n_r-1}) \in \mathbb{F}_2^{n_r}.$$

Note there are many alternative choices for the PCM of $C(r)$; in fact H chosen here has greater than $n_r - r$ rows and so this description contains redundancy. However what the choice above guarantees is that each row and column of H has weight 3, leading to low-weight gauge generators and optimal syndrome extraction scheduling, of an associated quantum code (see Section XIII).

The simplex codes are examples of highly symmetric classical codes with large automorphism groups.

Lemma VIII.6. *The automorphism group of the simplex codes are as follows:*

$$\mathrm{Aut}(C(r)) \cong GL_r(2).$$

Intuitively, Lemma VIII.6 means that each invertible linear transformation $g \in GL_r(2)$ of the r logical bits, is implemented by a distinct permutation σ of the n_r physical bits [5, Ch. 8.5].

Example VIII.7. Let $r = 3$. The polynomial $h(x) = 1 + x^2 + x^3$ is primitive and hence the overcomplete parity check matrix

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

defines the $(7,3,4)$-simplex code. A basis for $C(3)$ is

given by the rows of generator matrix

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}, \tag{7}$$

and we observe that

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot G = G \cdot (2,4)(5,6).$$

I.e., the bit permutation $(2,4)(5,6) \in \mathrm{Aut}(C(3))$ induces the linear transformation

$$(v_1, v_2, v_3) \mapsto (v_1 + v_2, v_2, v_3)$$

on the 3 logical bits $(v_1, v_2, v_3) \in \mathbb{F}_2^3$.

In the remainder of this work, we assume that all parity check matrices $H \in \mathbb{F}_2^{n_r \times n_r}$ for the classical simplex code are taken as above.

E. Subsystem hypergraph product simplex (SHYPS) codes

Here we describe our main quantum code construction, namely the *subsystem hypergraph product simplex code*, in greater detail.

Definition VIII.8. Let $r \geq 3, n_r = 2^r - 1, d_r = 2^{r-1}$, and let H_r be the parity check matrix for the (n_r, r, d_r)-classical simplex code. Then the subsystem hypergraph product of two copies of H_r, denoted $SHYPS(r)$, is the subsystem CSS code with gauge generators

$$G_X = (H \otimes I_{n_r}),\ G_Z = (I_{n_r} \otimes H).$$

We call $SHYPS(r)$ the *subsystem hypergraph product simplex code.*

It's clear by definition that the row/column weights of G_X and G_Z match those of H, and so the $SHYPS(r)$ codes are a QLDPC code family with gauge generators of weight 3. Moreover, the gauge generators have a particular geometric structure: we arrange the n_r^2 physical qubits of $SHYPS(r)$ in an $n_r \times n_r$ array with row major ordering, such that for standard basis vectors $\langle e_1, \ldots, e_{n_r} \rangle = \mathbb{F}_2^{n_r}$, the vector $e_i \otimes e_j$ corresponds to the (i,j)th position of the $n_r \times n_r$ qubit array. For example, the first row of H given in Example VIII.7 is $r_1 = e_1 + e_3 + e_4$ and hence the first row $r_1 \otimes e_1 \in G_X$ indicates an X-type gauge generator supported on the vector

$$r_1 \otimes e_1 = e_1 \otimes e_1 + e_3 \otimes e_1 + e_4 \otimes e_1,$$

i.e., on the $(1,1), (3,1)$ and $(4,1)$ qubits of the array. It follows that gauge generators in G_X (respectively G_Z) are supported on single columns (respectively rows) of the qubit array.

We follow [9, Sec. 3B] to describe the parameters $[n, k, d]$ of $SHYPS(r)$: firstly recall from the above that $n = n_r^2$. Next, to calculate the number of encoded qubits k, observe that the the Pauli operators that commute with all gauge generators, are generated by

$$\mathcal{L}_X = (I_{n_r} \otimes G),\ \mathcal{L}_Z = (G \otimes I_{n_r}). \tag{8}$$

Here G is a chosen generator matrix for the classical simplex code (so in particular G is a matrix of rank r such that $HG^T = 0$).

The centre of the gauge group $\langle G_X, G_Z \rangle$ determines the stabilizers of a subsystem code. In particular, for $SHYPS(r)$ these are generated by

$$S_X = (H \otimes G),\ S_Z = (G \otimes H).$$

Finally, k is calculated by comparing the ranks of $\mathcal{L}_X$ and S_X:

$$\begin{aligned} k &= \operatorname{rank} \mathcal{L}_X - \operatorname{rank} S_X \\ &= n_r \cdot r - (n_r - r) \cdot r \\ &= r^2. \end{aligned}$$

More specifically, $\mathrm{RowSpan}(\mathcal{L}_X \backslash S_X)$ determines the space of logical X-operators, with logical Z-operators defined similarly. However, as indicated in Section VIII B, we consider the action of these operators up to multiplication by the gauge group. Hence the minimum distance d of the code is given by the minimum weight operator in $\langle \mathcal{L}_X, \mathcal{L}_Z \rangle \cdot \langle G_X, G_Z \rangle - \langle G_X, G_Z \rangle$. For subsystem hypergraph product codes, this is exactly the minimum distance of the involved classical codes, and hence $d(SHYPS(r)) = d_r = 2^{r-1}$ [9, Sec. 3B].

In summary, we have the following

Theorem VIII.9. *Let $r \geq 3$ and let H be the parity check matrix for the $(2^r - 1, r, 2^{r-1})$-classical simplex code described in Section VIII D. The subsystem hypergraph product simplex code $SHYPS(r)$ is an $[n, k, d]$-quantum subsystem code with gauge group generated by 3-qubit operators and*

$$\begin{aligned} n &= (2^r - 1)^2, \\ k &= r^2, \\ d &= 2^{r-1}. \end{aligned}$$

It's evident from the tensor product structure of (8) that like the physical qubits, the logical qubits of $SHYPS(r)$ may be arranged in an $r \times r$ array, such that logical operators have support on *lines* of qubits. In fact, recent work [10] demonstrates that a basis of logical operators may be chosen such that pairs of logical X/Z operators have supports intersecting in at most one qubit. The following result is an immediate application of [10, Thm. 1]:

Theorem VIII.10. *Let $r \geq 3$. There exists a generator matrix $G \in \mathbb{F}_2^{r\times n_r}$ for the classical simplex code and an ordered list of r bit indices $\pi(G) \subset [1..n_r]$ known as* pivots*, such that $PG^T = I_r$ for the* pivot matrix $P \in \mathbb{F}_2^{r\times n_r}$*: $P_{i,j} = 1$ if and only if $\pi(G)[i] = j$. Moreover, the matrices*

$$L_X = (P \otimes G),\ L_Z = (G \otimes P),$$

form a symplectic basis for the logical X/Z operators of $SHYPS(r)$.

The proof of [10, Thm. 1] is constructive and, as the name pivots suggests, relies on a modified version of the Gaussian elimination algorithm [10, Alg. 1]. This yields a so-called *strongly lower triangular* basis for the simplex code, represented by G, with rows $\{g_i\}_{i\in\pi(G)}$ indexed by the pivots. As P has row weights equal to one, we see that the matrix products $PG^T = I_r$ and $L_X L_Z^T = I_{r^2}$ hold not only over $\mathbb{F}_2$ but over $\mathbb{R}$. Hence, given pairs of pivots $(i,j),(k,l) \in \pi(G)^2$, the associated logical Paulis $\overline{X}_{i,j}, \overline{Z}_{k,l}$ given by basis vectors $e_i \otimes g_j \in L_X$ and $g_k \otimes e_l \in L_Z$ have intersecting support if and only if $(i,j) = (l,k)$. Moreover, this intersection is on the (i,j)th qubit of the array. From this point, we assume that G, and the logical operators of $SHYPS(r)$, are of the form above.

F. Lifting classical automorphisms

The geometric structure of $SHYPS(r)$ suggests a natural way to *lift* automorphisms of the simplex code by independently permuting either rows or columns of the qubit array. In this manner, we see that $SHYPS(r)$ inherits $|GL_r(2)|^2 = O(2^{2r^2})$ automorphisms from the two copies of the classical simplex code, and hence $\mathrm{Aut}(SHYPS(r))$ grows exponentially with the number of logical qubits $k = r^2$.

Lemma VIII.11. *Let $r \geq 3$ and C be the (n_r, r, d_r)-simplex code with automorphisms $\sigma_1, \sigma_2 \in \mathrm{Aut}(C)$. Then $\sigma_1 \otimes \sigma_2 \in \mathrm{Aut}(SHYPS(r))$. Furthermore,*

$$|\mathrm{Aut}(SHYPS(r))| \geq |GL_r(2)|^2.$$

Proof. Clearly $\sigma_1 \otimes \sigma_2$ is a permutation of the required number of qubits $n = n_r^2$ and so we need only check that it preserves the gauge generators G_X and G_Z. By definition of code automorphisms, for each $\sigma_i \in \mathrm{Aut}(C)$ there exists a corresponding $h_{\sigma_i} \in GL_{m_i}(2)$ such that $h_{\sigma_i} H = H\sigma_i$. Hence

$$\begin{aligned} G_X \cdot (\sigma_1 \otimes \sigma_2) &= (H\sigma_1 \otimes \sigma_2) \\ &= (h_{\sigma_1} H \otimes \sigma_2) \\ &= (h_{\sigma_1} \otimes \sigma_2) \cdot G_X, \end{aligned}$$

with G_Z similarly preserved. For the second claim, note that a tensor product of matrices $A \otimes B$ is the identity if and only if A and B are also identity. Hence each pair $\sigma_1, \sigma_2 \in \mathrm{Aut}(C_1) \times \mathrm{Aut}(C_2)$ produces a distinct $\sigma_1 \otimes \sigma_2 \in \mathrm{Aut}(SHYPS(r))$. □

Note that the above result naturally generalises to all subsystem hypergraph product codes, constructed from possibly distinct classical codes.

To determine the logical operator induced by the above automorphisms we examine the permutation action on the logical basis given by Theorem VIII.10.

Lemma VIII.12. *Let $r \geq 3$ and $C(r)$ be the (n_r, r, d_r)-simplex code with generator matrix G. Furthermore, assume that $\sigma_1, \sigma_2 \in \mathrm{Aut}(C)$ with corresponding linear transformations $g_{\sigma_1}, g_{\sigma_2} \in GL_r(2)$ such that $g_{\sigma_i} G = G\sigma_i$. Then $\sigma_1 \otimes \sigma_2 \in \mathrm{Aut}(SHYPS(r))$ induces the following action on the basis of logical operators*

$$L_Z \cdot (\sigma_1 \otimes \sigma_2) = (g_{\sigma_1} \otimes g_{\sigma_2}^{-T}) \cdot L_Z,$$

$$L_X \cdot (\sigma_1 \otimes \sigma_2) = (g_{\sigma_1}^{-T} \otimes g_{\sigma_2}) \cdot L_X.$$

Proof. For ease of presentation, let's consider the action of $\sigma \otimes I$ where $\sigma \in \mathrm{Aut}(C)$ (the general case follows identically). Firstly observe that

$$L_Z(\sigma \otimes I) = (G\sigma \otimes P) = (g_\sigma G \otimes P) = (g_\sigma \otimes I) L_Z.$$

The permutation action on the basis of X-logicals is then given by

$$L_X(\sigma \otimes I) = (P\sigma \otimes G),$$

where

$$\begin{aligned} I_{r^2} = L_X \cdot L_Z^T &= L_X(\sigma \otimes I) \cdot (\sigma^T \otimes I) L_Z^T \\ &= (P\sigma \otimes G) \cdot (L_Z(\sigma \otimes I))^T \\ &= (P\sigma \otimes G) \cdot (G^T \otimes P^T) \cdot (g_\sigma^T \otimes I). \end{aligned}$$

Collecting terms and noting that $A \otimes B = I_{r^2}$ if and only if both components are identity (recall also that $PG^T = I_r$), it follows that $P\sigma G^T = g_\sigma^{-T}$. Now H spans $\ker G$ and hence all solutions to the above are of the form

$$P\sigma = g_\sigma^{-T} P + AH,$$

for some $A \in M_{n_r}$. In summary, there exists A such that

$$\begin{aligned} L_X(\sigma \otimes I) &= P\sigma \otimes G \\ &= (g_\sigma^{-T} P + AH) \otimes G \\ &= (g_\sigma^{-T} \otimes I)(P \otimes G) + (A \otimes I_r)(H \otimes G) \\ &= (g_\sigma^{-T} \otimes I) L_X + (A \otimes I_r) S_X. \end{aligned}$$

That is, up to stabilizers (the exact stabilizer determined by A), $\sigma \otimes I$ has the desired action on L_X. □

As a logical Clifford operator is determined (up to a phase) by its action on the the basis of logical Paulis, Lemma VIII.12 demonstrates that the qubit permutation $\sigma_1 \otimes \sigma_2$ induces the logical CNOT circuit corresponding to $g_{\sigma_1}^{-T} \otimes g_{\sigma_2} \in GL_{r^2}(2)$.

IX. CNOT OPERATORS IN SHYPS CODES

In this section, the collection of automorphisms

$$\{\sigma_1 \otimes \sigma_2 \mid \sigma_i \in \mathrm{Aut}(C(r))\} \leq \mathrm{Aut}(SHYPS(r))$$

serves as the foundation for generating all logical CNOT operators in the SHYPS codes.

First, following [6], we show how all logical cross-block CNOT operators (with all controls in a first code block, and all targets in a second code block) may be attained by sequences of physical depth-1 circuits, that interleave the transversal CNOT operator with code automorphisms. In-block CNOT operators are then achieved by use of an auxiliary code block (see Section IX A). To demonstrate efficient compilation, we develop substantial linear algebra machinery for certain matrix decomposition problems – we expect these methods to be useful for logical operator compilation in many other code families.

A summary of depth bounds for a range of logical CNOT operators of SHYPS codes is available in Table II.

Notation: We adopt the notation discussed in Example VIII.1, where CNOT circuits on b blocks of $k = r^2$ logical qubits or $n = n_r^2$ physical qubits, are described by invertible matrices in $GL_{br^2}(2)$ or $GL_{bn_r^2}(2)$, respectively. The collection of all (not necessarily invertible) $n \times n$ binary matrices is denoted by $M_n(2)$ and $E_{i,j} \in M_n(2)$ denotes the matrix of all zeros except the (i,j) entry equal to 1. The collection of diagonal matrices are denoted $\mathrm{Diag}_n(2) \subset M_n(2)$.

Lemma IX.1. *Let $g_1, g_2 \in GL_r(2)$. Then the logical CNOT circuits $\begin{pmatrix} I & g_1 \otimes g_2 \\ 0 & I \end{pmatrix}, \begin{pmatrix} I & 0 \\ g_1 \otimes g_2 & I \end{pmatrix} \in GL_{2r^2}(2)$ on code $SHYPS(r)$ may be implemented by a depth-1 physical CNOT circuit.*

Proof. First recall that (like all CSS codes) the transversal $CNOT^{\otimes n_r^2}$ implements $\overline{CNOT}^{\otimes r^2}$ between two code blocks of $SHYPS(r)$. Independently, it follows from Lemmas VIII.6 and VIII.12 that there exist $\sigma_i \in S_{n_r}$, such that the physical CNOT circuit given by $\begin{pmatrix} I & 0 \\ 0 & \sigma_1 \otimes \sigma_2 \end{pmatrix} \in GL_{2n_r^2}(2)$ implements $\overline{\begin{pmatrix} I & 0 \\ 0 & g_1 \otimes g_2 \end{pmatrix}} \in GL_{2r^2}(2)$. Hence

$$\begin{pmatrix} I & 0 \\ 0 & \sigma_1^{-1} \otimes \sigma_2^{-1} \end{pmatrix} \cdot \begin{pmatrix} I & I \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} I & 0 \\ 0 & \sigma_1 \otimes \sigma_2 \end{pmatrix} = \begin{pmatrix} I & \sigma_1 \otimes \sigma_2 \\ 0 & I \end{pmatrix},$$

implements

$$\overline{\begin{pmatrix} I & 0 \\ 0 & g_1^{-1} \otimes g_2^{-1} \end{pmatrix}} \cdot \overline{\begin{pmatrix} I & I \\ 0 & I \end{pmatrix}} \cdot \overline{\begin{pmatrix} I & 0 \\ 0 & g_1 \otimes g_2 \end{pmatrix}} = \overline{\begin{pmatrix} I & g_1 \otimes g_2 \\ 0 & I \end{pmatrix}}.$$

But recall that $\pi = \sigma_1 \otimes \sigma_2$ is a permutation matrix, and in particular has row and column weights equal to one. Hence

$$\begin{pmatrix} I & \pi \\ 0 & I \end{pmatrix}$$

represents the depth-1 physical CNOT circuit $\prod_i CNOT_{i,\pi^{-1}(i)+n_r^2}$. The circuit diagram illustrating this conjugated CNOT operator is given by Fig. 3.

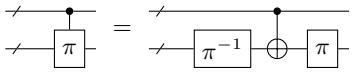


Figure 3. Physical implementation of a cross-block CNOT operator utilising $\pi \in \mathrm{Aut}(\mathcal{C})$.

The transposed logical CNOT circuit follows identically by exchanging the operations on the two code blocks. □

We now establish some notation for our generators, and drop the overline notation on the understanding that all operators are logical operators unless specified otherwise.

Definition IX.2. Denote the above collection of logical CNOT operators induced from classical automorphisms by

$$\mathcal{A} := \{\begin{pmatrix} I & g_1 \otimes g_2 \\ 0 & I \end{pmatrix} : g_i \in GL_r(2)\} \text{ and} \tag{9}$$

$$\mathcal{A}^T := \{\begin{pmatrix} I & 0 \\ g_1 \otimes g_2 & I \end{pmatrix} : g_i \in GL_r(2)\}. \tag{10}$$

A. Generating cross-block CNOT operators

The CNOT operators $\mathcal{A}$ and $\mathcal{A}^T$ are natural generalisations of $CNOT^{\otimes n_r^2}$ in that they have a transversal implementation on a pair of code blocks, and are thus inherently fault-tolerant. It is therefore highly desirable to use these circuits as generators for a larger class of CNOT operators. In this section we will see that they in fact suffice to efficiently generate all CNOT circuits between an arbitrary number of code blocks; a result that is highly unexpected in the context of generic quantum codes, and is derived from the particularly symmetric nature of the classical simplex codes.

Before proceeding, we state our first main result, which concerns cross-block CNOT operators in SHYPS codes

Theorem IX.3. *Let $A \in M_{r^2}(2)$. Then arbitrary cross-block CNOT operators given by*

$$\begin{pmatrix} I & A \\ 0 & I \end{pmatrix}, \begin{pmatrix} I & 0 \\ A & I \end{pmatrix} \in M_{2r^2}(2)$$

are implemented fault-tolerantly in $SHYPS(r)$ using at most $r^2 + r + 4$ generators from $\mathcal{A}$ and $\mathcal{A}^T$.

In particular, the implementation of cross-block CNOTs scales with the number of logical qubits $k = r^2$. In fact, by considering the size of the groups involved we see that this scaling is optimal up to a constant factor: as $|M_{r^2}(2)| = 2^{r^4}$ and $|\mathcal{A}| = |GL_r(2)|^2 = O(2^{2r^2})$,

$$M_{r^2}(2) = \mathcal{A}^l \implies |M_{r^2}(2)| \leq |\mathcal{A}|^l,$$

and hence $l \geq r^2/2$.

Theorem IX.3 is the foundation for much of the eficient, fault-tolerant computation in the SHYPS codes; it is from these operators that we build arbitrary CNOT operators, as well multi-block diagonal Clifford gates (in conjunction with a fold-transversal Hadmard gate – see Section XI).

To prove Theorem IX.3, first observe that the composition of cross-block CNOT operators in $\langle \mathcal{A} \rangle$ (and similarly in $\langle \mathcal{A}^T \rangle$) behaves like addition of the off-diagonal blocks

$$\begin{pmatrix} I & g_1 \otimes g_2 \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} I & h_1 \otimes h_2 \\ 0 & I \end{pmatrix} = \begin{pmatrix} I & g_1 \otimes g_2 + h_1 \otimes h_2 \\ 0 & I \end{pmatrix}.$$

Hence our Clifford generation problem becomes a question of showing that tensor products of the form $g_1 \otimes g_2$ efficiently generate the full matrix algebra, under addition. In summary, Theorem IX.3 is an immediate corollary of the following result

Theorem IX.4. *Let $A \in M_{r^2}(2)$. Then there exist w pairs of invertible matrices $g_{i_1}, g_{i_2} \in GL_r(2)$, for some $w \leq r^2 + r + 4$ such that $A = \sum_{i_1,i_2}^{w} g_{i_1} \otimes g_{i_2}$.*

The remainder of this section is devoted to the proof of Theorem IX.4 and so it is useful to begin with some remarks on our approach: First observe that the main challenge is that the matrices g_{i_1} etc. must be invertible. We first demonstrate that a decomposition of at most r^2 terms is easy without this invertibility condition, and then proceed to adapt this to invertible tensor products, while incurring minimal additional overhead. To this end we develop a number of matrix decomposition results that consider spanning sets in $GL_r(2)$, as well as the effect of adding specific invertible matrices such as permutations.

To track these results, we introduce the following *weight function*.

Definition IX.5. Let $A \in M_{r^2}(2)$. The *weight* of A, denoted $w(A)$, is the minimal number of pairs $(g_{i_1}, g_{i_2}) \in GL_r(2)^2$ such that $A = \sum_{i=1}^{w(A)} g_{i_1} \otimes g_{i_2}$.

First let's decompose our arbitrary matrix into a non-invertible tensor product

Lemma IX.6. *Let $A \in M_{r^2}(2)$. Then there exist $t \leq r^2$ matrix pairs $M_i, N_i \in M_r(2)$ such that $A = \sum M_i \otimes N_i$.*

Proof. Let $A_{i,j} \in M_r(2)$ be the (i,j)th block of A. Then

$$A = \sum_{i,j=1}^{r} E_{i,j} \otimes A_{i,j} \tag{11}$$

is a decomposition of the required form. □

Note that the minimal t required can often be much lower than r^2, and such a minimal decomposition is typically referred to as the *tensor rank decomposition*.

The task now is to convert an expression of the form (11) into a similar expression comprising only invertible matrices.

Lemma IX.7. *Let $M \in M_k(2)$, $(k \geq 2)$ then M is a sum of at most two elements in $GL_k(2)$.*

Proof. First let's check that the claim holds for the identity matrix.

$$I_2 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \text{ and } I_3 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

are appropriate decompositions for $k = 2, 3$. Then clearly larger I_k can be decomposed as blocks of 2 or 3 and treated as block sums of the above.

Now for arbitrary $M \in M_k(2)$ of rank $l \leq k$, there exist invertible matrices $g, g' \in GL_k(2)$ corresponding to row and column operations respectively, such that

$$gMg' = \begin{pmatrix} I_l & \\ & 0_{k-l} \end{pmatrix},$$

and zeros elsewhere via Gaussian elimination. But we know that there exist $X, Y \in GL_l(2)$ such that $X + Y = I_l$ and hence

$$M = (g)^{-1} \begin{pmatrix} X & \\ & I_{k-l} \end{pmatrix} (g')^{-1} + (g)^{-1} \begin{pmatrix} Y & \\ & I_{k-l} \end{pmatrix} (g')^{-1}$$

is a decomposition as a sum of two matrices in $GL_k(2)$. NB: if M has rank 1 then simply take

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

and proceed similarly. □

As the tensor product is distributive over addition, each non-invertible matrix may be split in two using Lemma IX.7 to yield

Corollary IX.8. *Let $A \in M_{r^2}(2)$. Then $w(A) \leq 4r^2$.*

Although this establishes a bound $w(A) \in O(r^2)$, we'll see that the constant factor 4 can be greatly improved.

Proposition IX.9. *Let $A \in M_{r^2}(2)$ have a tensor decomposition of rank t, i.e., there exists $M_i, N_i \in M_r(2)$ such that $A = \sum_{i=1}^{t} M_i \otimes N_i$. Then*

$$w(A) \leq \min(4t, 2t + 8, t + r + 6, r^2 + r + 4).$$

The first step in proving Proposition IX.9 is to generalise Lemma IX.7 to vector spaces and spanning sets of invertible matrices. This also requires an understanding of the proportion of binary matrices that are invertible.

Lemma IX.10. *Let $r \geq 1$. Then*

$$|GL_r(2)|/|M_r(2)| > \frac{1}{4}.$$

Proof. First observe from standard formulae that

$$|GL_r(2)|/|M_r(2)| = \prod_{i=1}^{r}(1-2^{-i}),$$

and hence the Lemma clearly holds for $r = 1$. For $r \geq 2$, we'll prove the slightly stronger statement

$$|GL_r(2)|/|M_r(2)| > \frac{1}{4} + 2^{-(r+3/2)},$$

via induction. The statement is easily checked for $r = 2$ and assuming the result holds for some $l \geq 2$:

$$\begin{aligned}
|GL_{l+1}(2)|/|M_{l+1}(2)| &= |GL_l(2)|/|M_l(2)| \cdot (1-2^{-(l+1)}) \\
&> (\frac{1}{4} + 2^{-(l+3/2)}) \cdot (1-2^{-(l+1)}) \\
&= \frac{1}{4} + 2^{-(l+3/2)} - 2^{-(l+3)} - 2^{-(2l+5/2)} \\
&= [\frac{1}{4} + 2^{-(l+1+3/2)}] \\
&\quad + [2^{-(l+1+3/2)} - 2^{-(l+3)} - 2^{-(2l+5/2)}].
\end{aligned}$$

So the result follows for $l+1$, provided

$$2^{-(l+5/2)} - 2^{-(l+3)} - 2^{-(2l+5/2)} > 0.$$

But this holds if and only if

$$2^l - 2^{l-1/2} - 1 = 2^{l-1}(2-\sqrt{2}) - 1 > 0,$$

which is indeed true for $l \geq 2$. □

Lemma IX.11. *Let $V \leq M_r(2)$ be a vector subspace of dimension d. Then there exists $T \subset GL_r(2)$ such that $|T| \leq \min(2d, r^2, d+2)$ and $V \leq \langle T \rangle$.*

Proof. The first two entries in the bound $\min(2d, r^2, d+2)$ follow immediately from Lemma IX.7 and the fact that $\langle GL_r(2) \rangle = M_r(2)$ which has dimension r^2. It therefore remains to show the final $d+2$ bound. Well if V contains an invertible matrix g_1 then $V = \langle g_1, V' \rangle$ for a vector space V' of strictly smaller dimension. Hence we assume without loss of generality that $V \leq M_r(2) \backslash GL_r(2)$ consists solely of non-invertible matrices. Let's denote a basis of V by $A_1, \ldots, A_d$.

As the cosets $M_r(2)/V$ tile the space $M_r(2)$, by Lemma IX.10 there exists $M \in M_r(2)$ such that the proportion of invertible matrices in the coset $M+V$ is greater than $1/4$. I.e., if we denote these invertible elements by $G \subseteq M+V$, then $|G| > 2^{d-2}$. Now as each element of G has the form $M + \sum_i \alpha_i A_i$, it follows that even-weight linear combinations of elements in G are non-invertible. Hence

$$|\langle G \rangle| \geq 2|G| > 2^{d-1},$$

and thus $\langle G \rangle$ has dimension at at least d. So there exist linearly independent $g_1, \ldots, g_d \in G$ and the subspace of even weight linear combinations $V' \leq \langle g_1, \ldots, g_d \rangle$ is a $d-1$ dimensional subspace $V' < V$. Finally, taking any $A \in V \backslash V'$ and applying Lemma IX.7 to yield a sum of two invertibles $A = g_{d+1} + g_{d+2}$, it follows that

$$V = \langle V', A \rangle \leq \langle g_1, \ldots, g_{d+2} \rangle.$$

□

Next we prove some useful Lemmas that study the effect of adding diagonal matrices and permutation matrices.

Lemma IX.12. *For all $A \in M_r(2)$ there exists a diagonal $D \in M_r(2)$ such that $A + D \in GL_r(2)$.*

Proof. We proceed by induction on r. The case $r = 1$ is clear, and suppose the Lemma holds holds for l. Then for $A \in M_{l+1}(2)$ we may write

$$A = \left(\begin{array}{c|c} A' & v \\ \hline u^T & a \end{array}\right)$$

for some $A' \in M_l(2)$, $u, v \in \mathbb{F}_2^l$, and $a \in \mathbb{F}_2$. By the induction hypothesis there exists diagonal $D' \in M_l(2)$ such that $A' + D' \in GL_l(2)$. Then for diagonal

$$D = \left(\begin{array}{c|c} D' & 0 \\ \hline 0 & a + u^T(A'+D')^{-1}v + 1 \end{array}\right)$$

we have

$$\begin{aligned}
A + D &= \left(\begin{array}{c|c} A'+D' & v \\ \hline u^T & u^T(A'+D')^{-1}v + 1 \end{array}\right) \\
&= \left(\begin{array}{c|c} I & 0 \\ \hline u^T(A'+D')^{-1} & 1 \end{array}\right) \cdot \left(\begin{array}{c|c} A'+D' & 0 \\ \hline 0 & 1 \end{array}\right) \\
&\quad \cdot \left(\begin{array}{c|c} I & (A'+D')^{-1}v \\ \hline 0 & 1 \end{array}\right).
\end{aligned}$$

As each matrix in the product decomposition above is invertible, $A + D \in GL_{l+1}(2)$. □

Corollary IX.13. *Let $A \in M_r(2)$ and $P \in GL_r(2)$ be a permutation matrix. Then there exists $O \in M_r(2)$ with nonzero entries supported on the nonzero entries of P such that $A + O \in GL_r(2)$.*

Proof. Apply Lemma IX.12 to $P^{-1}A$ to find a diagonal D such that $P^{-1}A + D$ is invertible and then define $O = PD$. □

Lemma IX.14. *Let $D \in M_r(2)$ be a diagonal matrix. Then either $D \in GL_r(2)$, or $D + P \in GL_r(2)$ for all r-cycle permutations P.*

Proof. Clearly D is invertible if and only if $D = I$, so we restrict to the case where $D \neq I$. First note that as P is an r-cycle, the full set of indices $1, \ldots, r$ is contained in the single orbit of P. I.e., given any starting point i,

the list $i,\ P(i),\ P^2(i),\ldots,\ P^{r-1}(i)$, is a re-ordering of $1,\ldots,r$. Consequently, we may index columns of $D+P$ by the numbers $P^l(i)$. Now since $D \neq I$, there exists some i such that the i-th diagonal entry of D, is zero. Choosing this as the starting point of our column indexing, we see that the columns of $D+P$ are given by

$$(De_i^T + e_{P(i)}^T = e_{P(i)}^T, De_{P(i)}^T + e_{P^2(i)}^T, \ldots, De_{P^{r-1}(i)}^T + e_{P^r(i)}^T).$$

By induction, we show the span of the first m elements of this sequence is given by

$$\langle e_{P(i)}^T, e_{P^2(i)}^T, \ldots, e_{P^m(i)}^T \rangle.$$

The base case is trivial by assumption. For the inductive step, observe that the $l+1$ entry of the sequence is either $e_{P^{l+1}(i)}^T$ or $e_{P^l(i)}^T + e_{P^{l+1}(i)}^T$, neither of which are in $\langle e_{P(i)}^T, e_{P^2(i)}^T, \ldots, e_{P^l(i)}^T \rangle$ since P is an r-cycle, and either of which when added to the generating set yields a vector space given by $\langle e_{P(i)}^T, e_{P^2(i)}^T, \ldots, e_{P^{l+1}(i)}^T \rangle$. Thus the span of the columns is

$$\left\langle e_{P(i)}^T, e_{P^2(i)}^T, \ldots, e_{P^r(i)}^T \right\rangle = \left\langle e_1^T, e_2^T, \ldots, e_r^T \right\rangle = \mathbb{F}_2^r.$$

As the columns of $D+P$ span $\mathbb{F}_2^r$, $D+P$ is invertible as desired. □

Corollary IX.15. *Let $P \in GL_r(2)$ be a permutation, and $D \in M_r(2)$ be diagonal. Then either $PD \in GL_r(2)$ or $PD + PQ \in GL_r(2)$ for all r-cycle permutations $Q \in GL_r(2)$.*

Proof. Apply Lemma IX.14 to D, then multiply by P, which preserves invertibility. □

With these results in place, we are ready to move onto the proof of Proposition IX.9

Proof. (Proof of Proposition IX.9). By assumption, we can write

$$A = \sum_{i=1}^{t} M_i \otimes N_i$$

for some $M_i, N_i \in M_r(2)$. By Lemma IX.11, we can find a set of q elements $B \subseteq GL_r(2)$ that generate all M_i, for $q \leq \min(2t, r^2, t+2)$. Expressing each M_i as a linear combination of elements of B, we have

$$A = \sum_{i=1}^{t} \left(\sum_{j=1}^{q} \mu_{ji} B_j \right) \otimes N_i = \sum_{j=1}^{q} B_j \otimes \left(\sum_{i=1}^{t} \mu_{ji} N_i \right).$$

By Lemma IX.12, there exists some $C_j \in GL_r(2)$ invertible and some D_j diagonal so that

$$C_j + D_j = \sum_{i=1}^{t} \mu_{ji} N_i$$

for all $1 \leq j \leq q$. The vector space spanned by the set $\{D_j\}_{j=1}^q$ has dimension $p \leq \min(q, r)$, and after expanding each D_j in a basis E for $\langle D_j \rangle$, we can rewrite A as

$$\begin{aligned} A &= \sum_{j=1}^{q} B_j \otimes C_j + \sum_{j=1}^{q} B_j \otimes D_j \\ &= \sum_{j=1}^{q} B_j \otimes C_j + \sum_{j=1}^{q} B_j \otimes \left(\sum_{\ell=1}^{p} \delta_{\ell j} E_\ell \right) \\ &= \sum_{j=1}^{q} B_j \otimes C_j + \sum_{\ell=1}^{p} \left(\sum_{j=1}^{q} \delta_{\ell j} B_j \right) \otimes E_\ell. \end{aligned}$$

Again, by Lemma IX.11 we can compute a set of m elements F from $GL_r(2)$ that contains $\left\langle \sum_{j=1}^{q} \delta_{\ell j} B_j \ :\ 1 \leq \ell \leq p \right\rangle$. Expressing elements from the span as F-linear combinations, we have

$$\begin{aligned} A &= \sum_{j=1}^{q} B_j \otimes C_j + \sum_{\ell=1}^{p} \left(\sum_{a=1}^{m} \beta_{a\ell} F_a \right) \otimes E_\ell \\ &= \sum_{j=1}^{q} B_j \otimes C_j + \sum_{a=1}^{m} F_a \otimes \left(\sum_{\ell=1}^{p} \beta_{a\ell} E_\ell \right). \end{aligned}$$

Next, by Lemma IX.14, every $\sum_{\ell=1}^{p} \beta_{a\ell} E_\ell$ is such that adding $\rho_a P$ for some $\rho_a \in \mathbb{F}_2$ and P a fixed r-cycle makes the matrix invertible, and so we have

$$\begin{aligned} A &= \sum_{j=1}^{q} B_j \otimes C_j + \sum_{a=1}^{m} F_a \otimes \left(\rho_a P + \sum_{\ell=1}^{p} \beta_{a\ell} E_\ell \right) \\ &\quad + \sum_{a=1}^{m} F_a \otimes (\rho_a P) \\ &= \sum_{j=1}^{q} B_j \otimes C_j + \sum_{a=1}^{m} F_a \otimes \left(\rho_a P + \sum_{\ell=1}^{p} \beta_{a\ell} E_\ell \right) \\ &\quad + \left(\sum_{a=1}^{m} \rho_a F_a \right) \otimes P \end{aligned}$$

Finally, by Lemma IX.7 we know that $\sum_{a=1}^{m} \rho_a F_a$ is the sum of at most two invertibles. This permits us to write for $G_1, G_2 \in \{0\} \cup GL_r(2)$

$$\begin{aligned} A &= \sum_{j=1}^{q} B_j \otimes C_j + \sum_{a=1}^{m} F_a \otimes \left(\rho_a P + \sum_{\ell=1}^{p} \beta_{a\ell} E_\ell \right) \\ &\quad + G_1 \otimes P + G_2 \otimes P \end{aligned}$$

where by construction every lone or bracketed term is an element of $GL_r(2)$. Therefore, we conclude $w(A) \leq q + m + 2$, where we recall

$$\begin{aligned} q &\leq \min(2t, r^2, t+2) \\ m &\leq \min(t+2, r) + 2. \end{aligned}$$

There are now four relevant regimes to bound $w(A)$:

($\mathbf{t \leq 3}$): rather than follow the process above, each M_i, N_i can be decomposed as a sum of two invertible matrices using Lemma IX.7 to yield $w(A) \leq 4t$.

($\mathbf{3 < t \leq r - 2}$): here we take q to be bounded by $t+2$ and m bounded by $t+4$ to give $w(A) \leq 2t + 8$.

($\mathbf{r - 2 < t \leq r^2 - 2}$): here we take q to be bounded by $t+2$ and m bounded by $r+2$ to give $w(A) \leq t + r + 6$.

($\mathbf{t > r^2 - 2}$): finally here we take q bounded by r^2 and $m \leq r + 2$ to yield $w(A) \leq r^2 + r + 4$. □

Summarising the worst case of Proposition IX.9, Theorem IX.4 is now immediate, and we repeat it below for convenience

Theorem IX.16. *Let $A \in M_{r^2}(2)$. Then $w(A) \leq r^2 + r + 4$.*

For applications in decomposing arbitary Clifford operators it is useful to consider the more specific case of upper-triangular matrices:

Lemma IX.17. *Let $A \in M_{r^2}(2)$ be any invertible upper-triangular matrix. Then $w(A) \leq r(r+1)/2 + 6$.*

Proof. Following similarly to the proof of Proposition IX.9, we can always write any such A as the sum

$$A = \sum_{i=1}^{\frac{r(r-1)}{2}} S_i \otimes N_i + \sum_{i=1}^{r} E_{i,i} \otimes U_i$$

for S_i strictly upper triangular matrices, N_i arbitrary elements of $M_r(2)$, $E_{i,i}$ the usual weight one diagonal matrices, and U_i an invertible upper triangular matrix. Computing a generating set G of invertible matrices for the space spanned by N_i with at most $q \leq \frac{r(r-1)}{2} + 2$ elements and collecting terms, we have

$$\begin{aligned} A &= \sum_{j=1}^{q} \left(\sum_{i=1}^{\frac{r(r-1)}{2}} \nu_{ji} S_i \right) \otimes G_j + \sum_{i=1}^{r} E_{i,i} \otimes U_i \\ &= I \otimes \left(\sum_{j=1}^{q} G_j \right) + \sum_{j=1}^{q} \left(I + \sum_{i=1}^{\frac{r(r+1)}{2}} \nu_{ji} S_i \right) \otimes G_j \\ &\quad + \sum_{i=1}^{r} E_{i,i} \otimes U_i. \end{aligned}$$

Applying Lemma IX.7 to the first term, and noting that each $I + \sum_{i=1}^{\frac{r(r+1)}{2}} \nu_{ji} S_i \in GL_r(2)$ yields

$$w(A) \leq q + 2 + w(\sum_{i=1}^{r} E_{i,i} \otimes U_i).$$

But then $E_{i,i} + C$ is invertible for any r-cycle C and hence the final term is bounded by

$$\sum_{i=1}^{r} w(E_{i,i} + C, U_i) + w(C, \sum_{i=1}^{r} U_i) \leq r + 2.$$

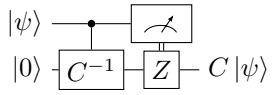
The result now follows. □

B. Arbitrary CNOT operators

Theorem IX.3 establishes efficient fault-tolerant implementations of cross-block CNOT circuits. It thus remains to show how our lifted automorphisms may implement CNOT circuits within a single code block. We remark that this does follow from [6, Thm. 4], however by relying not on generating transvections, but rather an auxiliary-block based trick, we minimise additional overhead.

Lemma IX.18. *Any logical CNOT circuit on a code block of $SHYPS(r)$ (i.e., any element of $GL_{r^2}(2)$) may be generated by $r^2 + r + 4$ depth-1 CNOT circuits from $\mathcal{A} \cup \mathcal{A}^T$, using a single auxiliary code block.*

Proof. Take an arbitrary $C \in GL_{r^2}(2)$. The CNOT circuit C can be executed using a scheme based on the well-known quantum teleportation circuit:



So at the cost of introducing an additional auxiliary code block in the $|0\rangle$ state (which may be prepared offline in constant depth) and teleporting our state $|\psi\rangle$, C is applied within the code block for the cost of a single element from $\langle \mathcal{A} \cup \mathcal{A}^T \rangle$. The lemma then follows directly from Theorem IX.3.

□

Note that in the remainder of this section, we consider arbitrary CNOT operators, modulo a possible permutation of the logical qubits. These will be accounted for later when compiling an arbitrary Clifford operator in Section XII, and in fact SWAP circuits may be implemented more efficiently than generic CNOT operators (see Section XI and Table IV). In particular, logical permutations on b code blocks are implementable in $O(r^2)$ depth, a cost that crucially doesn't scale with the number of code blocks (see Theorem XI.4).

Lemma IX.19. *Up to logical permutation, any logical CNOT circuit across two code blocks of $SHYPS(r)$ (i.e., any element of $GL_{2r^2}(2)$) may be generated by $4r^2 + 4r + 16$ depth-1 CNOT circuits from $\mathcal{A} \cup \mathcal{A}^T$, and executed in depth $3r^2 + 3r + 12$, using at most 2 additional auxiliary code blocks.*

Proof. Let $X \in GL_{2r^2}(2)$ and assume that X has PLU decomposition

$$X = P \begin{pmatrix} C_L & 0 \\ B_L & C'_L \end{pmatrix} \begin{pmatrix} C_U & A_U \\ 0 & C'_U \end{pmatrix}.$$

As all diagonal blocks in L and U are invertible, there

exist $A'_L, B'_U \in M_{r^2}(2)$ such that

$$X = P \begin{pmatrix} I & 0 \\ B'_L & I \end{pmatrix} \begin{pmatrix} C_L C_U & 0 \\ 0 & C'_L C'_U \end{pmatrix} \begin{pmatrix} I & A'_U \\ 0 & I \end{pmatrix}.$$

Here P is a cross block permutation that as stated in the Lemma, we need not consider (in practice it will be accounted for as part of a holistic Clifford decomposition, or simply tracked in software). Of the remaining three factors, the outer two matrices are clearly in $\langle \mathcal{A}^T \rangle$ and $\langle \mathcal{A} \rangle$, respectively. Hence it remains to implement

$$\begin{pmatrix} C_L C_U & 0 \\ 0 & C'_L C'_U \end{pmatrix},$$

i.e., arbitrary invertible CNOT circuits $C = C_L C_U$ and $C' = C'_L C'_U$, within the individual code blocks. It follows from Lemma IX.18 that these can be implemented for the cost of a single element from $\langle \mathcal{A} \cup \mathcal{A}^T \rangle$, each using a single auxiliary code block.

We conclude that X is implemented by a circuit consisting of at most 4 logical cross-block CNOT circuits, and hence $4(r^2 + r + 4)$ depth-1 circuits from $\mathcal{A} \cup \mathcal{A}^T$. Furthermore, C and C' can be performed in parallel since they are applied to different code blocks. Hence, X can be implemented in a depth no greater than $3(r^2 + r + 4)$. □

We conclude this section by generalising the above to an arbitrary number of blocks. For ease of presentation in the proof, and to avoid overly complicated compiling formulae we restrict our attention to $b = 2^a$ blocks - the general case follows similarly. We state the following result in terms of rounds of cross-block CNOT circuits in $\langle \mathcal{A} \cup \mathcal{A}^T \rangle$

Lemma IX.20. *Let $b = 2^a$ and $X = GL_{br^2}(2)$ be an arbitary CNOT circuit across b code blocks. Up to possible logical permutation, X is implemented by $b(b+1)$ cross-block CNOT circuits in $\langle \mathcal{A} \cup \mathcal{A}^T \rangle$, performed over $2b-1$ rounds.*

Proof. We start again with a PLU decomposition for X,

$$X = P \begin{pmatrix} C_L & 0 \\ B_L & C'_L \end{pmatrix} \begin{pmatrix} C_U & A_U \\ 0 & C'_U \end{pmatrix},$$

where each block in L, U themselves consist of $b/2$ sub-blocks, and the permutation P may be ignored. Isolating U, we have that

$$U = \begin{pmatrix} C_U & A_U \\ 0 & C'_U \end{pmatrix} = \begin{pmatrix} C_U & 0 \\ 0 & C'_U \end{pmatrix} \begin{pmatrix} I & C_U^{-1} A_U \\ 0 & I \end{pmatrix}.$$

But $C^{-1} A_U$ consists of $b/2 \times b/2$ blocks $M_{i,j} \in M_{r^2}(2)$ and hence

$$\begin{pmatrix} I & C_U^{-1} A_U \\ 0 & I \end{pmatrix} = \prod_{j=1}^{b/2} \begin{pmatrix} I & \sum_i^{b/2} E_{i,\sigma^j(i)} \otimes M_{i,\sigma^j(i)} \\ 0 & I \end{pmatrix}$$

$$= \prod_{i,j=1}^{b/2} \begin{pmatrix} I & E_{i,\sigma^j(i)} \otimes M_{i,\sigma^j(i)} \\ 0 & I \end{pmatrix},$$

where $\sigma = (1, 2, \ldots, b/2)$ is the length $b/2$ cyclic shift of indices. For example if $b = 4$, $\sigma = (1, 2)$ and

$$\begin{pmatrix} I & C_U^{-1} A_U \\ 0 & I \end{pmatrix} = \left[\begin{pmatrix} I & E_{1,1} \otimes M_{1,1} \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} I & E_{2,2} \otimes M_{2,2} \\ 0 & I \end{pmatrix} \right] \cdot \left[\begin{pmatrix} I & E_{1,2} \otimes M_{1,2} \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} I & E_{2,1} \otimes M_{2,1} \\ 0 & I \end{pmatrix} \right].$$

Observe that each square bracketed term, corresponding to a fixed power σ^j, consists of $b/2$ block to block operators in $\langle \mathcal{A} \rangle$ that may be performed in parallel. Hence the off-diagonal matrix in the decomposition of U requires $b^2/4$ operators in $\langle \mathcal{A} \rangle$, that with parallelisation can be achieved in $b/2$ rounds.

To cost the diagonal term in U we proceed by induction: assume that each (upper triangular) block $C_U, C'_U \in GL_{(b/2)r^2}(2)$ may be implemented by $f(b/2)$ generators in $\mathcal{A}$, over $t(b/2)$ rounds (both are performed in parallel). Then

$$t(b) = t(b/2) + b/2 \text{ and } f(b) = 2f(b/2) + b^2/4.$$

Solving these recurrence relations using initial conditions $t(2) = 2$ and $f(2) = 3$ yields

$$t(b) = b \text{ and } f(b) = \frac{b}{2}(b+1).$$

Recall that these costings are for U only, however the costings for L are identical. Lastly, we note that the b in-block CNOT operators appearing in L can be combined with those in U and executed together in a single round. Thus given cross-block operators in $\langle \mathcal{A} \cup \mathcal{A}^T \rangle$, any b-block CNOT circuit $X \in GL_{br^2}(2)$ can be implemented in $2b-1$ rounds, for a total $b(b+1)$ cross-block operators. □

Corollary IX.21. *Let $b = 2^a$ and $X = GL_{br^2}(2)$ be an arbitrary logical CNOT circuit across b code blocks of $SHYPS(r)$. Then up to a possible logical permutation, X is implementable in depth at most $(2b-1)(r^2+r+4)$, using depth-1 CNOT circuits in $\mathcal{A} \cup \mathcal{A}^T$.*

Proof. This is immediate from Theorem IX.3 and Lemma IX.20. □

Lastly, let's consider the case of single CNOT operators

Lemma IX.22. *Any single logical CNOT operator between or within code blocks is implementable fault-tolerantly in depth at most 4.*

Proof. A single CNOT gate across code blocks that connects the (i, j)th and (l, k)th qubits corresponds to the matrix in $GL_{2r^2}(2)$ with off diagonal block $E_{i,k} \otimes E_{j,l}$. The result then follows by decomposing each term in the tensor product into a sum of two invertible matrices using Lemma IX.7.

For an in-block gate, we instead decompose the matrix $I + E_{a,b} \otimes E_{c,d} \in GL_{r^2}(2)$ where at least one of $a \neq b$, or $c \neq d$ holds, and then apply the scheme introduced in the proof of Lemma IX.18. In particular, observe that

$$\begin{aligned} I + E_{a,b} \otimes E_{c,d} &= I \otimes (I + E_{c,d}) + (I + E_{a,b}) \otimes E_{c,d} \\ &= (I + E_{a,b}) \otimes I + E_{a,b} \otimes (I + E_{c,d}) \end{aligned}$$

One of the two expressions above will always contain at most two non-invertible terms, and hence two applications of Lemma IX.7 yields $w(I + E_{a,b} \otimes E_{c,d}) \leq 4$. Thus the corresponding in-block CNOT operator has depth at most 4.

We remark that the most common instance will be that both $a \neq b$ *and* $c \neq d$, in which case there is only one non-invertible term, yielding depth 3. □

Remark. **(Space Costs)** Throughout this section we have been primarily concerned with minimising the depth of CNOT operator implementations. This directly translates to minimising the number of logical cycles required in SHYPS codes and (assuming a fixed time cost for syndrome extraction), the total time cost. Of lesser importance has been tracking the space overhead of these operator implementations but this can be easily derived: cross-block logical generators in $\mathcal{A} \cup \mathcal{A}^T$ require 0 additional qubits, whereas Lemma IX.18 demonstrates that applying an in-block CNOT circuit requires a single auxiliary code block. Hence, assuming that in-block operators are performed in parallel (as in the proof of Lemma IX.20), a logical CNOT operator on b code blocks incurs a space overhead of at most bn qubits. Additionally, in the large b limit, [6, Thm. 4] implies that we can get away with 0 additional code blocks without changing the leading order terms for time overheads.

X. DIAGONAL OPERATORS IN SHYPS CODES

The previous section characterizes the depth-1 CNOT circuits that arise as lifts of classical automorphisms, and furthermore, how such circuits generate the full group of CNOT operators across multiple blocks of the SHYPS codes. We now turn our attention to diagonal Clifford gates, i.e., those that correspond to diagonal unitary matrices with respect to the computational basis. In particular we describe a collection of depth-1 logical diagonal Clifford operators that leverage automorphisms of the classical simplex code $C(r)$, and demonstrate how these efficiently generate all diagonal Cliffords. Recall from Section VIII A that up to phase, such Clifford operators are generated by S and CZ gates and form an abelian subgroup of $\mathcal{C}_n$.

Before going through the details on diagonal operators, we note that all results below can be trivially converted to results on X-diagonal operators, i.e., Clifford operators that are diagonal unitary matrices when considered in the *Hadamard-rotated* computational basis. This subgroup of Clifford operators is obtained by conjugating diagonal operators by the all-qubit Hadamard operator. The fact that results on diagonal operators carry over to X-diagonal operators can be understood by considering that conjugating a diagonal operator by the all-qubit Hadamard operator transforms its symplectic representation by moving the off-diagonal block from the top-right quadrant to the bottom-left quadrant.

A. Lifting classical automorphisms

Recall that diagonal Clifford operators are represented modulo Paulis by symplectic matrices

$$\{\begin{pmatrix} I & B \\ 0 & I \end{pmatrix} \in Sp_{2k}(2) \mid B \in SYM_k(2)\}, \tag{12}$$

where $SYM_k(2) \subset M_k(2)$ denotes the subspace of symmetric matrices.

As the diagonal subgroup is abelian, efficiently generating all operators thus becomes a question of decomposing arbitrary symmetric matrices as short sums from a particular collection of generators. We construct these generating symmetric matrices from classical automorphisms such that the corresponding diagonal operators have favorable depth and fault-tolerance properties.

Before proceeding we first define a distinguished permutation on our qubit array; in the language of [11], this permutation is a *ZX-duality*. Intuitively, the map τ exchanges the vector spaces defining X- and Z-gauge operators, and will be used to account for the fact that conjugating X-Paulis by diagonal Cliffords may produce Z-Paulis.

Definition X.1. Given a^2 qubits (physical or logical) arranged in an $a \times a$ array, let $\tau_a \in S_{a^2}$ be the permutation that exchanges qubits across the diagonal, i.e., τ_a is an involution exchanging qubits $(i, j) \leftrightarrow (j, i)$. In particular, $\tau_a = \tau_a^{-1} = \tau_a^T$.

As we typically index qubits by the tensor product basis $e_i^T \otimes e_j^T$, we have equivalently in vector form that $\tau_a(e_i^T \otimes e_j^T) = e_j^T \otimes e_i^T$. This extends naturally to matrices acting on this basis:

$$\tau_a(A \otimes B)\tau_a = B \otimes A.$$

We're now ready to introduce our generating set of logical diagonal Clifford operators in the SHYPS codes: a collection of so-called *phase-type* fold-transversal operators [11].

Lemma X.2. *Let $r \geq 3$ and $C(r)$ be the $(n_r, r, 2^{r-1})$-simplex code with generator matrix G. Furthermore, assume that $\sigma \in \mathrm{Aut}(C(r))$ with corresponding linear transformation $g_\sigma \in GL_r(2)$ such that $g_\sigma G = G\sigma$.*

Then σ lifts to a physical diagonal Clifford operator

$$U(\sigma) := \begin{pmatrix} I & (\sigma \otimes \sigma^T) \cdot \tau_{n_r} \\ 0 & I \end{pmatrix}$$

that preserves $SHYPS(r)$. Moreover, this depth-1 circuit of diagonal gates has logical action given by

$$\overline{U}(g_\sigma) := \begin{pmatrix} I & (g_\sigma^{-T} \otimes g_\sigma^{-1}) \cdot \tau_r \\ 0 & I \end{pmatrix}.$$

Proof. Firstly we need to establish that $B = (\sigma \otimes \sigma^T) \cdot \tau_{n_r}$ is symmetric for $U(\sigma)$ to define a valid physical diagonal Clifford. As the transpose and inverse of a permutation matrix are equal, it suffices to check that B is self-inverse:

$$\begin{aligned} B^2 &= (\sigma \otimes \sigma^T)\tau_{n_r}(\sigma \otimes \sigma^T)\tau_{n_r} \\ &= (\sigma \otimes \sigma^T)(\sigma^T \otimes \sigma) \\ &= (\sigma\sigma^T \otimes \sigma^T\sigma) \\ &= I \end{aligned}$$

Next observe that as a product of permutation matrices, B has row/column weights equal to one and thus corresponds to a depth-1 physical circuit. As U commutes with Z-gauge operators, to confirm that U acts as a logical operator of $SHYPS(r)$ we need only check the action on X-gauge operators, described by G_X. Well

$$\begin{aligned} G_X \cdot B &= (H \otimes I_{n_r}) \cdot (\sigma \otimes \sigma^T) \cdot \tau_{n_r} \\ &= (H\sigma \otimes \sigma^T) \cdot \tau_{n_r} \\ &= (h_\sigma \otimes \sigma^T)(H \otimes I) \cdot \tau_{n_r} \\ &= (h_\sigma \otimes \sigma^T) \cdot \tau_r \cdot (I \otimes H), \end{aligned}$$

where we've used that $\mathrm{Aut}(C(r)) = \mathrm{Aut}(C(r)^\perp)$ and hence there exists h_σ such that $H\sigma = h_\sigma H$. But

$$(h_\sigma \otimes \sigma^T) \cdot \tau_r \cdot (I \otimes H) = (h_\sigma \otimes \sigma^T) \cdot \tau_r \cdot G_Z$$

and hence the action of $U(\sigma)$ is given by

$$[G_X, 0] \cdot U = [G_X, (h_\sigma \otimes \sigma) \cdot \tau_r \cdot G_Z].$$

We conclude that conjugation by the physical Clifford U preserves stabilizers and hence performs a valid logical Clifford operator on $SHYPS(r)$. Note that by the nature of the symplectic group, this logical Clifford is implemented up to a Pauli correction. As there is always a Pauli operator with the appropriate (anti)commutation relations with the stabilizers and logical operators of the code to fix any logical/stabilizer action sign issues, we can ignore this subtlety [12]. The logical action of $\overline{U}$ is then determined akin to the proof of Lemma VIII.12:

$$\begin{aligned} L_X \cdot B &= (P \otimes G) \cdot (\sigma \otimes \sigma^T) \cdot \tau_{n_r} \\ &= (P\sigma \otimes G\sigma^T) \cdot \tau_{n_r} \\ &= (g^{-T} \otimes g^{-1})(P \otimes G) \cdot \tau_{n_r} \\ &= (g^{-T} \otimes g^{-1}) \cdot \tau_r \cdot (G \otimes P) \\ &= (g^{-T} \otimes g^{-1}) \cdot \tau_r \cdot L_Z. \end{aligned}$$

□

As we are focused primarily on the logical action and not the particular automorphism being used, we shall typically relabel $g^{-T} \mapsto g$ for ease of presentation. Summarising the above we have

Corollary X.3. *Let $g \in GL_r(2)$. The logical Clifford operator $\overline{U} = \begin{pmatrix} I & (g \otimes g^T) \cdot \tau_r \\ 0 & I \end{pmatrix}$ on code $SHYPS(r)$ is implementable by a depth-1 physical diagonal Clifford circuit.*

Example X.4. Let $r = 3$ and G be the simplex code generator matrix from (7). Observe that

$$\begin{aligned} \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} G &= \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \\ &= G \cdot (1,4)(3,5). \end{aligned}$$

Hence

$$U = \begin{pmatrix} I & [(1,4)(3,5) \otimes (1,4)(3,5)] \cdot \tau_7 \\ 0 & I \end{pmatrix}$$

implements

$$\overline{U} = \begin{pmatrix} I & \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tau_3 \\ 0 & I \end{pmatrix}.$$

Remark. Recall that the presence of S and CZ gates in diagonal Clifford operators (12) is determined by the diagonal and off-diagonal entries of B, respectively. Then specifically for our generators, diagonal entries are determined by the fixed points of the permutation matrix $\sigma \otimes \sigma^T \cdot \tau_{n_r}$. Here we see that

$$\begin{aligned} \sigma \otimes \sigma^T \cdot \tau_{n_r}(e_i^T \otimes e_j^T) &= \sigma \otimes \sigma^T(e_j^T \otimes e_i^T) \\ &= e_{\sigma(j)}^T \otimes e_{\sigma^T(i)}^T \end{aligned}$$

equals $e_i^T \otimes e_j^T$ if and only if $\sigma(j) = i$. Hence $\sigma \otimes \sigma^T \cdot \tau_{n_r}$ has exactly $n_r = \sqrt{n}$ fixed points $\{e_{\sigma(j)}^T \otimes e_j^T\}_{j=1}^{n_r}$, and the corresponding physical diagonal Clifford generator contains n_r S gates, and $(n_r^2 - n_r)/2$ CZ gates.

Corollary X.3 establishes a collection of logical diagonal Cliffords that are implemented in SHYPS codes by depth-1 physical circuits and we denote these by

$$\mathcal{B} := \{ \begin{pmatrix} I & (g \otimes g^T) \cdot \tau_r \\ 0 & I \end{pmatrix} : g \in GL_r(2) \}.$$

Unlike the CNOT circuits of Lemma IX.1 these operators are not implemented strictly transversally. Furthermore, as they contain entangling CZ gates within a code block, one may be concerned that X-faults occurring mid-circuit spread to a high-weight Z-error pattern on different qubits. However from a fault-tolerance perspective, this error spread is only of concern if the additional Paulis increase the support of a nontrivial logical operator. Otherwise the final error pattern, and the

collection of circuit faults that caused it, are equally correctable by the code's error correction protocol.

This notion of fault-tolerance is quantified by the *circuit distance* d_{circ} which is the minimum number of faults required in a noisy quantum circuit to produce a logical error, without triggering any syndromes. It's clear that d_{circ} is bounded above by the code distance, and if $d_{circ} = d$ we call the circuit *distance preserving*.

As the following Lemma demonstrates, our diagonal operators $\mathcal{B}$ meet this distance preserving criterion.

Lemma X.5. *Let $\overline{U} \in \mathcal{B}$. Then there exists a physical implementation with circuit distance $d_{circ} = d$.*

Proof. By Lemma X.2, there exists $\sigma \in S_{n_r}$ such that $\overline{U}$ is implemented by $U(\sigma \otimes \sigma^T)$ where the off-diagonal entries of $\sigma \otimes \sigma^T$ determine the presence of physical CZ gates. As discussed above, CZ gates spread Y/Z faults but by the structure of the logical Paulis in $SHYPS(r)$ (see Theorem VIII.10), a single X-fault spreads to a weight two Z-Pauli in the support of a logical operator if and only if the CZ connects qubits in the same row or column. Hence to show that $d_{circ} = d$ it suffices to show that this is not the case.

Well firstly consider two distinct qubits $e_i \otimes e_j$ and $e_i \otimes e_k$, $j \neq k$ in some fixed row i. Then

$$\begin{aligned}(\sigma \otimes \sigma^T)\tau_{n_{r(i,j),(i,k)}} &= (e_i \otimes e_j) \cdot (\sigma \otimes \sigma^T)\tau_{n_r} \cdot (e_i \otimes e_k)^T \\ &= (e_i \sigma e_k^T) \otimes (e_j \sigma^T e_i^T) \\ &= (e_i \sigma e_k^T) \otimes (e_i \sigma e_j^T)^T.\end{aligned}$$

But $(e_i \sigma e_k^T) = 1$ if and only if $\sigma(k) = i$ and this cannot also hold for $k \neq j$. Hence $(\sigma \otimes \sigma^T)\tau_{n_{r(i,j),(i,k)}} = 0$ for $j \neq k$, confirming that $U(\sigma \otimes \sigma^T)$ contains no CZ gates joining qubits within a row. The result for columns follows identically. □

In summary, the diagonal operators $\overline{U}(g \otimes g^T)$ are depth-1, distance-preserving (i.e., fault-tolerant) logical operators of $SHYPS(r)$. It's natural therefore to proceed as in Section IX A and consider the subgroup of logical diagonal operators they generate.

B. Generating in-block operators

In this section we show that the fault-tolerant logical operators $\mathcal{B}$ generate all diagonal Clifford operators of the SHYPS codes. Furthermore, the compilation of operators using this generating set scales asymptotically optimally. For exact costings of the overhead required to implement a range of diagonal operators we refer the reader to Table III, but first we focus on proving the following result.

Theorem X.6. *Every in-block logical diagonal Clifford operator (modulo Paulis) in $SHYPS(r)$ is implementable by a sequence of at most $r^2 + 5r + 2$ generators from $\mathcal{B}$ for $r \geq 4$ and at most $r^2 + 8r + 2$ generators for $r = 3$.*

As aforementioned, the diagonal Clifford operators (modulo Paulis) of the $SHYPS(r)$ code are in one-to-one correspondence with symmetric matrices $SYM_{r^2}(2)$. Hence Theorem X.6 is an immediate corollary of the following result.

Theorem X.7. *Let $S \in SYM_{r^2}(2)$. Then there exist $A_1, \ldots, A_p \in GL_r(2)$ for some $p \leq r^2 + 5r + 2$ ($r^2 + 8r + 2$ when $r = 3$) such that*

$$S = \sum_{i=1}^{p} (A_i \otimes A_i^T)\tau.$$

Similar to the work in Section IX, the strategy for proving Theorem X.7 is to first show an analogous (and typically stronger) result over (possibly) non-invertible matrices. We then adapt such a decomposition to invertible matrices, while minimising any additional overhead incurred.

Lemma X.8. *Let $S \in SYM_{r^2}(2)$. Then there exist $M_1, \ldots, M_a \in M_r(2)$ for some $a \leq r^2 + 1$, such that*

$$S = \sum_{i=1}^{a} (M_i \otimes M_i^T)\tau.$$

The proof of Lemma X.8 uses the following notions of matrix reshaping

Definition X.9. (Flatten) Let $M = (m_{i,j}) \in M_{r^2}(2)$. Then the *flatten function* $fl : M_r(2) \longrightarrow \mathbb{F}_2^{1 \times r^2}$ is defined

$$fl : \begin{pmatrix} m_{1,1} & \ldots & m_{1,r} \\ \vdots & & \vdots \\ m_{r,1} & \ldots & m_{r,r} \end{pmatrix} \mapsto \begin{pmatrix} m_{1,1} & \ldots & m_{1,r} & m_{2,1} & \ldots & m_{r,r} \end{pmatrix}.$$

By considering the action of $fl(M)$ on basis vectors $e_i^T \otimes e_j^T$ and $e_j^T \otimes e_i^T$, we see immediately that $fl(M^T) = fl(M) \cdot \tau$.

(Reshape) Given $M \in M_{r^2}(2)$, there exist block matrices $M_{i,j}$ such that

$$M = \sum_{1 \leq i,j \leq r} E_{i,j} \otimes M_{i,j}.$$

Then the re-shape function $re : M_{r^2}(2) \longrightarrow M_{r^2}(2)$ is defined as follows

$$re : M \mapsto \sum_{1 \leq i,j \leq r} fl(E_{i,j})^T \cdot fl(M_{i,j})$$

I.e., the successive rows of $re(M)$ are formed by flattening successive blocks of M

For example

$$re : \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \mapsto \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}.$$

The effect of the reshape map is more evident on how it acts on basis vectors

$$(e_{i_1} \otimes e_{i_2}) re(M) (e_{j_1}^T \otimes e_{j_2}^T) = (e_{i_1} \otimes e_{j_1}) M (e_{i_2}^T \otimes e_{j_2}^T).$$

I.e., it effectively swaps the indices i_2 and j_1. This is thus linear and also self-inverse.

Proof. (Proof of Lemma X.8) The proof strategy is to use a binary Cholesky decomposition [13] that decomposes symmetric matrices as sums of outer products of vectors with themselves. To account for τ we don't apply this directly to the desired S but rather a transformed matrix $S' = re(S \cdot \tau) \cdot \tau$ - undoing this reshaping later correspondingly transforms outer products into summands of the desired tensor product form.

First observe that

$$\begin{aligned}(e_{i_1} \otimes e_{i_2}) S' (e_{j_1}^T \otimes e_{j_2}^T) &= (e_{i_1} \otimes e_{i_2}) re(S \cdot \tau) \cdot \tau (e_{j_1}^T \otimes e_{j_2}^T) \\ &= (e_{i_1} \otimes e_{j_2}) S (e_{j_1}^T \otimes e_{i_2}^T) \\ &= (e_{j_1} \otimes e_{i_2}) S (e_{i_1}^T \otimes e_{j_2}^T) \\ &= (e_{j_1} \otimes e_{j_2}) S' (e_{i_1}^T \otimes e_{i_2}^T),\end{aligned}$$

where we have unpacked the definitions of τ, re and used that S is symmetric. Hence, by definition S' is also symmetric.

Now we apply the work of [13] to decompose S' via a binary analogue of the well-known Cholesky decomposition. Let $a = \operatorname{rank}(S') + (\prod (S_{i,i} + 1) \mod 2)$ (i.e., the rank of S', with an additional plus one if all diagonal entries are zero, so in particular $a \leq r^2 + 1$). Then there exists $L \in \mathbb{F}_2^{r^2 \times a}$ such that $S' = LL^T$ [13, Thm. 1].

Next observe that the matrix product LL^T may be rewritten as a sum of outerproducts of the columns l_k of L with themselves

$$S' = LL^T = (l_1 \dots l_a)(l_1 \dots l_a)^T = \sum_{k=1}^{a} l_k l_k^T.$$

Considering a single summand of the form ll^T, there exists $M = (m_{i,j}) \in M_r(2)$ such that $l^T = fl(M)$ and hence

$$ll^T = \begin{pmatrix} m_{1,1} \\ \vdots \\ m_{r,r} \end{pmatrix} (m_{1,1}, \dots, m_{r,r}) = \sum_{i,j} m_{i,j} fl(E_{i,j})^T fl(M).$$

It then follows that

$$\begin{aligned} re^{-1}(ll^T \cdot \tau) &= \sum_{i,j} m_{i,j} \cdot re^{-1}\left(fl(E_{i,j})^T fl(M) \cdot \tau \right) \\ &= \sum_{i,j} m_{i,j} \cdot re^{-1}\left(fl(E_{i,j})^T fl(M^T) \right) \\ &= \sum_{i,j} m_{i,j} E_{i,j} \otimes M^T \\ &= M \otimes M^T. \end{aligned}$$

Finally, letting $fl(M_k) = l_k^T$ for each column l_k, we have by linearity that

$$\begin{aligned} S &= re^{-1}(S' \cdot \tau) \cdot \tau \\ &= \sum_{k=1}^{a} (M_k \otimes M_k^T) \tau. \end{aligned}$$

□

Example X.10. Consider

$$S = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \mapsto S' = re(S \cdot \tau) \cdot \tau = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}.$$

Applying the constructive algorithm in [13, Sec. 3] we produce a minimal binary Cholesky decomposition

$$S' = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}.$$

Isolating the summand corresponding to the second column, we have

$$\begin{aligned} re^{-1}(l_2 l_2^T \tau) &= re^{-1}\left(\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \tau \right) \\ &= \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^T. \end{aligned}$$

Handling the first column l_1 similarly then yields

$$S = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}^T \cdot \tau + \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^T \cdot \tau.$$

With Lemma X.8 established, we now investigate methods for converting a sum of non-invertible $M_i \otimes M_i^T$ into a similar decomposition comprising only invertible matrices.

To this end, it is useful to set up some notation and collect a few preliminary lemmas.

Definition X.11. Let $A, B \in M_r(2)$. We define

$$S(A) := (A \otimes A^T)\tau$$

$$\begin{aligned} D(A, B) &:= S(A + B) + S(A) + S(B) \\ &= (A \otimes B^T + B \otimes A^T)\tau \end{aligned}$$

In this notation, the goal is to find a minimal number of $A_i \in GL_r(2)$ such that $S = \sum_i S(A_i)$. Similarly to

Section IX, it useful to introduce a weight function to track this

Definition X.12. Let $S \in SYM_{r^2}(2)$. The *weight* of S, denoted $w(S)$, is the minimal number of $A_i \in GL_r(2)$ such that $S = \sum_i^{w(S)} S(A_i)$.

We remark that terms of the form $D(A, B)$ are the natural "cross-terms" incurred when splitting non-invertible matrices into sums of invertibles - dealing with these is the main remaining challenge.

The following facts are immediate from the definitions

Lemma X.13. *Let $A, B, C_i \in M_r(2)$. Then the following hold*

(a) $D(A, B) = D(B, A)$

(b) $D(A, B) = D(A, A + B)$

(c) $D(A, \sum_i C_i) = \sum_i D(A, C_i)$.

Lemma X.14. *Let $A, B \in M_r(2)$ and assume there exists $C \in GL_r(2)$ such that*

$$A + C, B + C, A + B + C \in GL_r(2).$$

Then

$$D(A, B) = S(A+C)+S(B+C)+S(A+B+C)+S(C).$$

In particular $w(D(A, B)) \leq 4$.

Lemma X.15. *Let $A, B \in M_r(2)$ and $P, Q \in GL_r(2)$. Then*

$$(P \otimes Q)S(A)(Q^T \otimes P^T) = S(PAQ^T)$$

$$(P \otimes Q)D(A, B)(Q^T \otimes P^T) = D(PAQ^T, PBQ^T)$$

Hence for $M \in M_{r^2}(2)$, we have that

$$w(M) = w((P \otimes Q)M(Q^T \otimes P^T)).$$

In particular, $w(D(A, B)) = w(D(PAQ^T, PBQ^T))$.

Proof. (Proof of Theorem X.7) By Lemma X.8 there exists a decomposition

$$S = \sum_{i=1}^{a} (M_i \otimes M_i^T)\tau, \tag{13}$$

for some $a \leq r^2 + 1$. We first consider the addition of diagonal matrices D_i to each component M_i - as seen in Section IX, this can be advantageous as the space spanned by the D_i has dimension at most r. So, applying Lemma IX.12, there exist invertible matrices A_i and diagonal D_i such that $M_i = A_i + D_i$. It follows that

$$S = \sum_{i=1}^{a} S(M_i) \tag{14}$$

$$= \sum_{i=1}^{a} S(A_i) + S(D_i) + D(A_i, D_i). \tag{15}$$

Note that each diagonal D_i may itself be written in the standard diagonal basis $E_{j,j} \in M_r(2)$ to yield

$$S(D_i) = \sum_{j=1}^{r} d_{i,j} \left(S(E_{j,j}) + D(E_{j,j}, \sum_{k>j} d_{i,k} E_{k,k}) \right), \tag{16}$$

for some $d_{i,j} \in \mathbb{F}_2$. Then substituting each decomposition (16) into (15), expanding each $D(A_i, D_i)$ in the digaonal basis, and collecting terms by repeated use of Lemma X.13 (a) and (c) yields

$$S = \sum_{i=1}^{a} S(A_i) + \sum_{j=1}^{r} e_j S(E_{j,j}) + \sum_{j=1}^{r} f_j D(E_{j,j}, B_j), \tag{17}$$

for some $e_i, f_i \in \mathbb{F}_2$ and $B_i \in M_r(2)$. Note in particular that the latter two sums contain at most r terms each, and that the B_j are not necessarily invertible. Of course each $E_{j,j} \in M_r(2) \backslash GL_r(2)$ also, but applying Lemma IX.14, there exists a single r-cycle permutation matrix σ such that each $E_{i,i} + \sigma \in GL_r(2)$. By substituting

$$S(E_{i,i}) = S(E_{i,i} + \sigma) + S(\sigma) + D(E_{i,i}, \sigma)$$

into (17) and again collecting like terms by applying Lemma X.13 yields

$$S = \sum_{i=1}^{a} S(A_i) + \sum_{j=1}^{r} e_j S(E_{j,j} + \sigma) \tag{18}$$

$$+ S(\sigma) \cdot (\sum_j e_j) + \sum_{j=1}^{r} f'_j D(E_{j,j}, B'_j). \tag{19}$$

To summarise, up to the addition of $\sum_i^r f'_i D(E_{i,i}, B'_i)$, we have that $w(S) \leq r^2+r+2$. It thus remains to handle this final term. Moreover, as there exist permutation matrices $P, Q \in GL_r(2)$ such that $E_{j,j} = PE_{1,1}Q^T$, we may restrict our attention to a single summand $D(E_{1,1}, B_1)$ by Lemma X.15.

Now again using Lemma X.15, observe that if $P, Q \in GL_r(2)$ are such that $PE_{1,1}Q^T = E_{1,1}$, then $w(D(E_{1,1}, B_1)) = w(D(E_{1,1}, PB_1Q^T))$. But this holds for a large range of P and Q:

$$\begin{pmatrix} 1 & p^T \\ \mathbf{0} & P' \end{pmatrix} \cdot E_{1,1} \cdot \begin{pmatrix} 1 & \mathbf{0}^T \\ q & Q' \end{pmatrix} = E_{1,1},$$

for all vectors $p, q \in \mathbb{F}_2^{r-1}$ and $P', Q' \in GL_{r-1}(2)$. Left and right multiplication by such P and Q^T allows us to perform a modified Gaussian elimination on B_1, reducing

it to the form

$$B_1' = \begin{pmatrix} X & \\ & D \end{pmatrix} = \begin{pmatrix} a & b & 0 & 0 & \\ c & d & 0 & 0 & \\ 0 & 0 & e & 0 & \\ 0 & 0 & 0 & f & \\ & & & & D \end{pmatrix},$$

where $D \in \mathrm{Diag}_{r-4}(2)$ is diagonal.

Now if $r = 4, 5$ we check computationally that for all B_1' there exists $C \in GL_r(2)$ such that $C, C + B_1', C + E_{1,1}, C + B_1' + E_{1,1}$ are invertible. Thus by Lemma X.14, $w(D(E_{1,1}, B_1')) \leq 4$, and by the arguments above, all $w(D(E_{j,j}, B_j)) \leq 4$.

If $r \geq 6$ then first note by the $r = 4$ computations, there exists an invertible $C \in GL_4(2)$ such that $X + C, E_{1,1} + C$ and $X + E_{1,1} + C$ are all invertible. To handle the lower block of dimension $r - 4 \geq 2$, if D is not full rank, then we choose C' to be any $(r-4)$-cycle permutation matrix and $D + C' \in GL_{r-4}(2)$ by Lemma IX.14. If $D = I_{r-4}$, then it suffices to choose any invertible C' that fixes only the zero vector (and such a matrix always exists for $r - 4 \geq 2$).

Hence letting $C'' = \begin{pmatrix} C & \\ & C' \end{pmatrix}$, we have that C''

$$E_{1,1} + C'' = \begin{pmatrix} E_{1,1} + C & \\ & C' \end{pmatrix},$$

$$B_1' + C'' = \begin{pmatrix} X + C & \\ & D + C' \end{pmatrix},$$

$$B_1' + E_{1,1} + C'' = \begin{pmatrix} X + E_{1,1} + C & \\ & D + C' \end{pmatrix}$$

are all invertible. Thus as above, we have by Lemma X.14, $w(D(E_{1,1}, B_1')) \leq 4$, and similarly, all $w(D(E_{j,j}, B_j)) \leq 4$. In conclusion, when $r \geq 4$ we collect terms in (18), to see that there exists at most $r^2 + 5r + 2$ matrices $A_i \in GL_r(2)$ such that $S = \sum_i S(A_i)$.

It remains to consider the case $r = 3$. Here we note that provided

$$B_1' \neq \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{20}$$

we may proceed as above, as there exists a choice of $C \in GL_3(2)$ that satisfies the conditions of Lemma X.14. However if B_1' is one of the two exclusions in (20) then no such C exists, and we in fact check computationally that $w(D(E_{1,1}, B_1')) = 7$. The r possible terms of this form thus incurs an additional cost of up to $7r$, giving an overall bound of $r^2 + 8r + 2$. □

In summary, we have demonstrated that all logical diagonal Clifford operators of the $SHYPS(r)$ codes may be implemented fault-tolerantly, using generators of (physical) depth 1. Moreover, the total depth required for an arbitrary diagonal operator is bounded above by $r^2 + 5r + 2$ (for $r \geq 4$). Comparing the number of generators

$$|\mathcal{B}| = |GL_r(2)| \sim 2^{r^2}$$

to the total number of diagonal operators

$$|SYM_{r^2}(2)| = 2^{r^2(r^2+1)/2},$$

we see that our achieved compilation in $O(r^2)$ steps is optimal up to a constant factor. Moreover as $k = r^2$ is the number of logical qubits, this mirrors the case of k un-encoded qubits: given a symmetric matrix $S \in SYM_k(2)$, the depth of the (un-encoded) diagonal Clifford operator is determined by solving an associated graph colouring problem (ignoring diagonal entries, treat S as an adjacency matrix). Colouring the complete graph on k vertices requires $k+1$ colours, and correspondingly, there exist diagonal operators that require depth $k + 1$. Hence encoding using the SHYPS codes incurs a negligible depth penalty.

C. Compiling specific operators

Theorem X.6 demonstrates efficient compiling of arbitrary (in-block) diagonal Clifford operators in the SHYPS codes. However, costings of single one- and two-qubit logical gates are also of interest, and we examine these below. For a full breakdown of logical diagonal Clifford costs, we refer the reader to Table III

Let's first consider S circuits.

Lemma X.16. *Let $S_{(i,j)}$ be a single qubit logical S operator corresponding to the (i, j)-position in the $r \times r$ logical qubit array of $SHYPS(r)$. Then the following holds*

1. *There exists an implementation of $S_{(i,j)}$ using 6 (9 when $r = 3$) generators from $\mathcal{B}$.*
2. *There exists a depth-1 operator in $\mathcal{B}$ that performs $S_{(i,j)}$ (plus additional diagonal gates, in general)*

Proof. Recall that the operator $S_{(i,j)}$ corresponds to the symmetric matrix B with single nonzero entry in the (i, j)th diagonal position. In fact, $B = (E_{i,j} \otimes E_{i,j}^T)\tau_r =: S(E_{i,j})$. By Lemma IX.7 there exists $X \in GL_r(2)$ such that $E_{i,j} + X \in GL_r(2)$ and this yields

$$S(E_{i,j}) = S(E_{i,j} + X) + S(X) + D(E_{i,j}, X).$$

Observing that there exists permutations matrices P, Q such that $PE_{i,j}Q^T = E_{1,1}$, we follow the proof of Theorem X.7) to show that $w(D(E_{1,1}, X)) \leq 4$ and hence $w(B) \leq 6$, provided $r \geq 4$. If $r = 3$ then the associated X above may produce one of the exceptions listed in (20), yielding $w(B) \leq 7 + 2 = 9$ instead.

For the second statement, it suffices to choose any $g \in GL_r(2)$ such that $(g \otimes g^T) \cdot \tau_r$ has a nonzero entry in the (i,j)th diagonal position. Well

$$\begin{aligned}(e_i \otimes e_j) \cdot ((g \otimes g^T) \cdot \tau_r) \cdot (e_i \otimes e_j)^T &= e_i g e_j^T \otimes e_j g^T e_i^T \\ &= e_i g e_j^T \otimes (e_i g e_j^T)^T \\ &= (g)_{i,j}.\end{aligned}$$

So it suffices to choose any invertible $g \in GL_r(2)$ with nonzero (i,j)th entry. A simple choice is the permutation matrix corresponding to the 2 cycle (i,j) - note that such a choice minimises the weight of $(g \otimes g^T) \cdot \tau_r$ and thus has a 'minimal' logical action (containing $S_{(i,j)}$). □

Corollary X.17. *Let S^V be a logical S circuit, where the (a,b)th entry of $V \in M_r(2)$ indicates an S gate performed on the (a,b)th qubit. If V is invertible then there exists a depth-1 generator that implements a logical diagonal Clifford operator containing S^V.*

We are similarly able to produce low-depth implementations for isolated CZ gates in the SHYPS codes.

Lemma X.18. *Let $(a,b) \neq (c,d)$ be qubit positions in the logical qubit array of $SHYPS(r)$. Then there exists an implementation of $CZ_{(a,b),(c,d)}$ in physical depth 4, using 4 generators from $\mathcal{B}$.*

Proof. Let $S = D(E_{a,d}, E_{c,b})$. Clearly S is symmetric and contains a single pair of off-diagonal entries. Furthermore, $(a,b) \neq (c,d)$ implies that

$$\begin{aligned}(e_a \otimes e_b) S (e_c \otimes e_d)^T &= e_a E_{a,d} e_d^T \otimes e_b E_{b,c} e_c^T \\ &= 1.\end{aligned}$$

Hence the nonzero entries in S correspond to the logical operator $CZ_{(a,b),(c,d)}$. Next observe that there exist row and column permutations P and Q^T such that the nonzero entries of $PE_{a,d}Q^T, PE_{c,b}Q^T$ lie below the diagonal. So in particular, adding the identity I_r to these lower diagonal matrices is guaranteed to be invertible. The result then follows by Lemma X.14 and Lemma X.15. □

Further results for low-depth implementations of specific diagonal operators are given in Section XI. There we shall see that for a fixed set of logical qubits, the ability to perform *any* logical diagonal on said qubits, relates to low-depth implementations of arbitrary Hadamard circuits.

D. Multi-block diagonal Cliffords

To generate logical diagonal Clifford circuits across multiple blocks of our chosen SHYPS code, we rely on the results of Section IX A by Hadamard transforming cross-block CNOT operators. Furthermore, due to the tensor product form of the gauge generators in the SHYPS codes, an all-qubit logical Hadamard is particularly easy to implement.

Lemma X.19. *In $SHYPS(r)$, the physical Hadamard-SWAP operator $H_1 \cdots H_{n^2} \tau_{n_r}$ implements the logical operator $H_1 \cdots H_{r^2} \tau_r$.*

Proof. This is clear from the definitions of G_X, L_X etc, and the fact that $(P \otimes G)\tau_{n_r} = \tau_r(G \otimes P)$. □

In essence, the all-qubit logical Hadamard is implemented by an all-qubit physical Hadamard. As the following Lemma for cross-block CZ operators demonstrates, the additional SWAP circuits τ_n and τ_r can be easily accounted for by adjusting the automorphisms in our depth-1 CNOT generators. Thus cross-block CZ circuits may be implemented analogously to cross-block CNOT operators, by a low-depth sequence of transversal CZ circuits.

Lemma X.20. *Let $A \in M_{r^2}(2)$ and*

$$U_{1,2}(A) = \prod_{i,j=1}^{r} CZ_{i,r^2+j}^{a_{i,j}}.$$

be the corresponding cross-block circuit of logical CZ gates. Then $U_{1,2}(A)$ is implemented by a sequence of at most $r^2 + r + 4$ transversal physical CZ circuits.

Proof. First observe that U has symplectic matrix representation

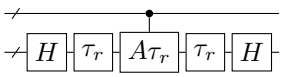
$$\left(\begin{array}{cc|cc} I & 0 & 0 & A \\ 0 & I & A^T & 0 \\ \hline 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{array}\right) = \left(\begin{array}{cc|cc} I & 0 & 0 & 0 \\ 0 & 0 & 0 & I \\ \hline 0 & 0 & I & 0 \\ 0 & I & 0 & 0 \end{array}\right) \left(\begin{array}{cc|cc} I & A & 0 & 0 \\ 0 & I & 0 & 0 \\ \hline 0 & 0 & I & 0 \\ 0 & 0 & A^T & I \end{array}\right) \cdot \left(\begin{array}{cc|cc} I & 0 & 0 & 0 \\ 0 & 0 & 0 & I \\ \hline 0 & 0 & I & 0 \\ 0 & I & 0 & 0 \end{array}\right).$$

Inserting two instances of $\tau_r^2 = 1$ then yields

$$\begin{aligned}U_{1,2}(A) = &\left(\begin{array}{cc|cc} I & 0 & 0 & 0 \\ 0 & 0 & 0 & I \\ \hline 0 & 0 & I & 0 \\ 0 & I & 0 & 0 \end{array}\right) \left(\begin{array}{cc|cc} I & 0 & 0 & 0 \\ 0 & \tau_r & 0 & 0 \\ \hline 0 & 0 & I & 0 \\ 0 & 0 & 0 & \tau_r \end{array}\right) \\ &\cdot \left(\begin{array}{cc|cc} I & A \cdot \tau_r & 0 & 0 \\ 0 & I & 0 & 0 \\ \hline 0 & 0 & I & 0 \\ 0 & 0 & (A \cdot \tau_r)^T & I \end{array}\right) \left(\begin{array}{cc|cc} I & 0 & 0 & 0 \\ 0 & \tau_r & 0 & 0 \\ \hline 0 & 0 & I & 0 \\ 0 & 0 & 0 & \tau_r \end{array}\right) \\ &\cdot \left(\begin{array}{cc|cc} I & 0 & 0 & 0 \\ 0 & 0 & 0 & I \\ \hline 0 & 0 & I & 0 \\ 0 & I & 0 & 0 \end{array}\right). \qquad (21)\end{aligned}$$

So this is exactly a cross-block CNOT circuit given by off-diagonal matrix $A \cdot \tau_r$, conjugated by the logical Hadamard-SWAP operator $H^{\otimes r^2} \tau_r$ on the second code

block. In circuit form:



Combining Theorem IX.4 and Lemma X.19, there exists $a \leq r^2 + r + 4$ automorphisms $\pi_i = \sigma_{i_1} \otimes \sigma_{i_2} \in \mathrm{Aut}(SHYPS(r))$ such that $U_{1,2}(A)$ is implemented by the sequence of conjugated transversal CNOT operators

$$(I_{n_r^2} \otimes H^{\otimes n_r^2} \tau_{n_r}) \cdot \prod_{i=1}^{a} \prod_{j=1}^{n_r^2} CNOT_{j,\pi_i^{-1}(j)+n_r^2} \quad (22)$$

$$\cdot (I_{n_r^2} \otimes \tau_{n_r} H^{\otimes n_r^2}), \quad (23)$$

Now, conjugating by τ_{n_r} exchanges the factors in automorphisms π_i, while the Hadamard action transforms every physical CNOT into a physical CZ gate. Hence letting $\rho_i = \sigma_{i_2} \otimes \sigma_{i_1} \in \mathrm{Aut}(SHYPS(r))$ we have that (23) equals

$$\prod_{i=1}^{a} \left(\prod_{j=1}^{n_r^2} CZ_{j,\rho_i^{-1}(j)+n_r^2} \right).$$

In particular, each bracketed term above is a logical operator of $SHYPS(r)$ that is evidently depth-1 and transversal (and therefore fault-tolerant). □

The following corollary is immediate by following the proof above, but inserting the improved depth bound Lemma IX.22 for isolated CNOT gates.

Corollary X.21. *Any single logical CZ operator between two SHYPS code blocks is implementable fault-tolerantly in depth at most 4.*

Generalising the above operator to $U_{i,j}(A_{i,j})$, it follows that an arbitrary b-block diagonal Clifford consisting of only cross-block CZ gates may be written

$$\prod_{1 \leq i < j \leq b} U_{i,j}(A_{i,j}) = \left(\begin{array}{ccc|cccc} I & & & 0 & A_{1,1} & \cdots & A_{1,b} \\ & I & & A_{1,1}^T & 0 & \cdots & A_{2,b} \\ & & \ddots & \vdots & \vdots & \ddots & \vdots \\ & & I & A_{1,b}^T & A_{2,b}^T & \cdots & 0 \\ \hline & & & I & & & \\ & & & & I & & \\ & & & & & \ddots & \\ & & & & & & I \end{array} \right).$$

We now cost this operator in a similar manner to Section IX. As in Lemma IX.20 we restrict our attention to $b = 2^a$ blocks for ease of presentation.

Lemma X.22. *Let $b = 2^l$. The diagonal operator $U = \prod_{1 \leq i < j \leq b} U_{i,j}(A_{i,j})$ on b blocks requires at most $b/2 \cdot (b-1)$ block-to-block operators $U_{i,j}(A)$, implemented over at most $b-1$ rounds.*

Proof. The number of operators required is clear and so we need only prove the time constraint. We proceed by induction: in the base case $b = 2$, $U = U_{1,2}(A)$ and hence requires exactly 1 round. Next assume that the result holds for $b/2$ blocks and observe that

$$U = \left[\prod_{1 \leq i < j \leq b/2} U_{i,j}(A_{i,j}) \cdot \prod_{b/2+1 \leq i < j \leq b} U_{i,j}(A_{i,j}) \right] \cdot \prod_{1 \leq i,j \leq b/2} U_{i,j+b/2}(A_{i,j}) =: [U_1 \cdot U_2] \cdot U_3.$$

As U_1 and U_2 act on disjoint sets of $b/2$ blocks, by assumption they can be simultaneously implemented in $b/2 - 1$ rounds. The remaining $(b/2)^2$ operators that make up U_3 are then implementable in a further $b/2$ rounds (scheduled via a cyclic shift as in the proof of Lemma IX.20), giving a total of $b - 1$. □

The following Corollary is immediate by combining Lemma X.22 and Lemma X.20

Corollary X.23. *The diagonal operator $U = \prod_{1 \leq i < j \leq b} U_{i,j}(A_{i,j})$ on b blocks is implemented in depth at most $(b-1)(r^2 + r + 4)$.*

To complete this section on diagonal Clifford operators, we combine our work on in-block and cross-block operators in Theorem X.24. Note that the results of this section are also presented in Table III, while Section XI contains additional work on diagonal operators that are useful for compiling Hadamard circuits.

Theorem X.24. *Any logical diagonal Clifford operator (modulo Paulis) on b-blocks of $SHYPS(r)$, is implemented in depth at most*

$$br^2 + r(b+4) + (4b-2), \ r \geq 4,$$

$$16b + 25 = br^2 + r(b+7) + (4b-2), \ r = 3.$$

Furthermore, such implementations require no auxiliary qubits, and thus have zero space cost.

Proof. As the subgroup of diagonal Clifford operators is abelian, we may implement an arbitrary operator as a sequence of parallel in-block gates, followed by any necessary cross-block gates. We therefore accrue an initial circuit of depth $r^2 + 5r + 2$ (or $r^2 + 8r + 2$ when $r = 3$) for in-block gates by Theorem X.6, regardless of block count. This is then followed by additional depth of at most $(b-1)(r^2 + r + 4)$ by Corollary X.23.

It's clear that implementing diagonal operators in SHYPS codes incurs zero space overhead: in-block diagonal operators are implemented directly using logical generators from $\mathcal{B}$, whereas cross-block logical CZ gates mirrors the case of cross-block CNOT operators seen in Section IX. □

XI. HADAMARD-SWAP OPERATORS IN SHYPS CODES

To complete the analysis of logical Clifford implementations in the SHYPS codes, we need lastly to characterize the Hadamard gates. However as Lemma X.19 suggests, it is natural to consider Hadamard circuits in conjunction with SWAP circuits as together these operators generate a subgroup of $Sp_{2k}(2)$ called the *signed-symmetric group* or *hyperoctahedral group*.

We find that logical permutations within a single SHYPS code block present significant depth savings compared to generic CNOT circuits ($O(r)$ versus $O(r^2)$). Moreover, in Section XI B we show that the depth of multi-block permutations does not grow with the number of blocks b. Lastly, we examine arbitrary Hadamard circuits in Section XI C. Here we combine our work on in-block permutations with certain depth-1 diagonal circuits, to implement Hadamard circuits in depth $O(r)$.

A. In-block permutations

First recall Lemma IX.1 that for any choice of $g_1, g_2 \in GL_r(2)$, the logical CNOT circuit $\begin{pmatrix} I & g_1 \otimes g_2 \\ 0 & I \end{pmatrix} \in GL_{2r^2}(2)$ may be implemented by a depth-1 physical CNOT circuit. Moreover, these depth-1 circuits generate the full algebra of upper-right cross block CNOT operations

$$\{\begin{pmatrix} I & A \\ 0 & I \end{pmatrix} : A \in M_{r^2}(2)\}.$$

To achieve the full algebra, we generically require $O(r^2)$ depth-1 generators but let's restrict our attention to certain permutations in $GL_{r^2}(2)$: For $\sigma_1, \ldots, \sigma_r \in S_r$ we define

$$A(\sigma_i) := \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{pmatrix} = \sum_{i=1}^{r} E_{i,i} \otimes \sigma_i,$$

and then for an additional $\upsilon \in S_r$, consider the product

$$A(\sigma_i) \cdot (\upsilon \otimes I) \in GL_{r^2}(2). \tag{24}$$

Now clearly these elements are permutations of the r^2 logical qubits, but moreover they form a subgroup isomorphic to $S_r^r \rtimes S_r \leq S_{r^2}$ - this can be observed by checking that

$$\begin{aligned}(\upsilon^{-1} \otimes I)(\sum_i E_{i,i} \otimes \sigma_i)(\upsilon \otimes I) &= \sum_i E_{\upsilon^{-1}(i),\upsilon^{-1}(i)} \otimes \sigma_i \\ &= \sum_i E_{i,i} \otimes \sigma_{\upsilon(i)}.\end{aligned}$$

To understand exactly how this subgroup acts, first consider the action of $A(\sigma_i)$:

$$\begin{aligned}(\sum_{j,k} \alpha_{j,k} e_j \otimes e_k) A(\sigma_i) &= (\sum_{j,k} \alpha_{j,k} e_j \otimes e_k) \sum_{i=1}^{r} E_{i,i} \otimes \sigma_i \\ &= \sum_{i=1}^{r} \sum_k \alpha_{i,k} e_i \otimes e_k \sigma_i \\ &= \sum_{i=1}^{r} \sum_k \alpha_{i,k} e_i \otimes e_{\sigma_i^{-1}(k)}.\end{aligned}$$

In particular for each chosen σ_i in A, there is a corresponding permutation action on the basis vectors $e_i \otimes e_1, \ldots, e_i \otimes e_r$. In qubit terms, $A(\sigma_i)$ performs a permutation σ_i along each row of the $r \times r$ grid. Similarly, it's easy to check that the action of $(\upsilon \otimes I)$ on basis vectors induces a permutation of the rows of qubits.

Expanding the product (24) we see that

$$A(\sigma_i) \cdot (\upsilon \otimes I) = \sum_{i=1}^{r} E_{i,\upsilon^{-1}(i)} \otimes \sigma_i. \tag{25}$$

Since $E_{i,\upsilon^{-1}(i)} \cdot \upsilon^{-1} = E_{i,i}$, it follows from Lemma IX.14 that $E_{i,\upsilon^{-1}(i)} \cdot \upsilon^{-1} + \rho \in GL_r(2)$ for any r-cycle ρ and hence $E_{i,\upsilon^{-1}(i)} + \rho\upsilon \in GL_r(2)$. Collecting terms in Eq. (25) then yields

$$\begin{aligned}A(\sigma_i) \cdot (\upsilon \otimes I) = &\sum_i (E_{i,\upsilon^{-1}(i)} + \rho\upsilon) \otimes \sigma_i \\ &+ \rho\upsilon \otimes (\sum_i \sigma_i).\end{aligned}$$

Finally, applying Lemma IX.7 to the (possibly) non-invertible permutation sum in the second term provides a decomposition of $A(\sigma_i) \cdot (\upsilon \otimes I)$ into at most $r+2$ matrices $g_1 \otimes g_2 \in Gl_r(2)^2$. It follows that the cross-block CNOT circuit $\begin{pmatrix} I & A(\sigma_i) \cdot (\upsilon \otimes I) \\ 0 & I \end{pmatrix}$ and the in-block CNOT circuit $\begin{pmatrix} A(\sigma_i) \cdot (\upsilon \otimes I) & 0 \\ 0 & I \end{pmatrix}$ are implemented by $r+2$ depth-1 generators, respectively. Here for the in-block operator we have used the cross-block trick from Lemma IX.18.

Now in the analysis above, we could have instead chosen matrices that act on columns rather than rows, i.e., of the form:

$$B(\sigma_i) \cdot (I \otimes \upsilon) := (\sum_{i=1}^{r} \sigma_i \otimes E_{i,i})(I \otimes \upsilon).$$

By the same arguments, these generate a subgroup isomorphic to $S_r^r \rtimes S_r$, implementable using at most $r+2$ depth-1 physical generators. Observe however that these two subgroups, called say K_{row} and K_{col}, are distinct. They are also maximal subgroups of S_{r^2} by the famous Onan-Scott Theorem [14], and hence $S_{r^2} = \langle K_{\text{row}}, K_{\text{col}} \rangle$.

I.e., these two particular collection of logical SWAP circuits, generate all permutations on a code block of r^2 logical qubits. In fact, one can show that $S_{r^2} = K_{\text{row}} \cdot K_{\text{col}} \cdot K_{\text{row}}$ [15], yielding the following result:

Proposition XI.1. *Any logical permutation on a single $SHYPS(r)$ code block is implemented fault-tolerantly, using CNOT generators $\mathcal{A} \cup \mathcal{A}^T$, in depth at most $3(r+2)$.*

In-block permutations can be used to derive a useful result on cross-block CNOT circuits: *depth-1* cross-block CNOT circuits can be implemented in $O(r)$ depth, rather than $O(r^2)$ as required for general cross-block CNOT circuits.

Corollary XI.2. *Any logical depth-1 cross-block CNOT circuit across two $SHYPS(r)$ code blocks can be implemented in depth no greater than $8r + 18$, using CNOT generators $\mathcal{A} \cup \mathcal{A}^T$.*

Proof. Take an arbitrary logical depth on cross-block CNOT circuit

$$X = \begin{pmatrix} I & A \\ 0 & I \end{pmatrix} \in GL_{2r^2}(2).$$

Since X is depth-1, the matrix $A \in M_{r^2}(2)$ has at most a single nonzero entry in each row and column. Therefore, there exists a permutation $P \in S_{r^2}$ such that $AP \in \text{Diag}_{r^2}$. We first rewrite X as

$$X = \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \begin{pmatrix} I & AP \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & P^{-1} \end{pmatrix}. \tag{26}$$

As any matrix in $\text{Diag}_{r^2}(2)$ has tensor rank at most r, it follows from Proposition IX.9 that $w(AP) \leq 2r + 6$. Therefore, the cross-block CNOT operator appearing in Eq. (26) can be implemented using at most $2r+6$ depth-1 generators from $\mathcal{A}$.

To conclude the proof, we need to add the cost of the in-block permutations P and P^{-1}. By Proposition XI.1, it follows that P and P^{-1} can each be implemented in depth at most $3r+6$, which results in a maximal physical depth for X of $8r + 18$. □

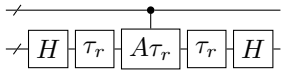
A similar result exists for depth-1 cross-block CZ circuits.

Corollary XI.3. *Any logical depth-1 cross-block CZ circuit across two $SHYPS(r)$ code blocks can be implemented in depth no greater than $8r + 18$.*

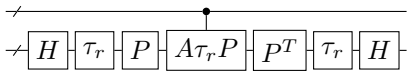
Proof. Take any logical depth-1 cross-block CZ circuit U. First observe that U has symplectic matrix representation

$$\left(\begin{array}{cc|cc} I & 0 & 0 & A \\ 0 & I & A^T & 0 \\ \hline 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{array}\right), \tag{27}$$

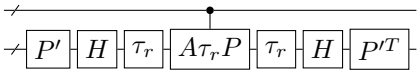
where the matrix A has at most one nonzero entry in each row or column. Similar to the strategy used in the proof of Lemma X.20, we first rewrite U as a cross-block CNOT operator, conjugated by $H^{\otimes r^2}\tau_r$ on the second code block:



Note that since A has rows and columns of weight at most one, so has $A\tau_r$. We decompose the depth-1 cross-block CNOT operator in the circuit above as we did in the proof of Lemma IX.1:



where $P \in S_{r^2}$ is such that $A\tau_r P \in \text{Diag}_{r^2}(2)$. Conjugating P by $H^{\otimes r^2}\tau_r$ yields a different permutation P', and hence we can rewrite the previous circuit as



Now, since $A\tau_r P \in \text{Diag}_{r^2}(2)$, its tensor rank is no greater than r. Therefore, Proposition IX.9, implies that the cross-block CNOT operator $\begin{pmatrix} I & A\tau_r P \\ 0 & I \end{pmatrix} \in GL_{2r^2}$ can be implemented using at most $2r + 6$ depth-1 generators from $\mathcal{A}$. Following the same reasoning as in the proof of Lemma X.20, we then find that U can be implemented using two logical in-block permutations and up to $2r + 6$ transversal physical CZ circuits.

Finally, implementing the logical in-block permutations P' and P'^T requires at most depth $2 \cdot (3r + 6)$ by Proposition XI.1. We thus find that U can be implemented in depth at most $8r + 18$. □

B. Multi-block permutations

Next, we turn to logical multi-block permutations. One might initially expect that the depth of their physical implementation would scale with the total number of logical qubits (i.e., br^2 for b code blocks of $SHYPS(r)$) like it does for, e.g., multi-block CZ circuits. However, we will show below that general permutations can be done much more efficiently. In particular, they are implementable in a depth that does not scale with the number of code blocks, but simply with the number of logical qubits per block.

The strategy to efficiently implement a general permutation is to decompose it as the product of two involutions (i.e., two depth-1 SWAP circuits). The logical SWAP gates in these involutions may then be scheduled efficiently using an edge-coloring algorithm, and the resulting number of required rounds does not scale with

the number of code blocks but rather with the number of qubits per block. Since individual SWAP gates can be performed in depth $O(1)$, it then follows that multi-block permutations can be performed in detph $O(r^2)$.

Theorem XI.4. *A logical permutation operator on b code blocks of $SHYPS(r)$ can be implemented in depth $36r^2 + 3r + 6$.*

Proof. Take any permutation operator P acting on br^2 qubits. We first write P as the product of two involutions: $P = A \cdot B$. Each of these involutions can in turn be written as a product of an involution consisting of in-block SWAP operators, and one consisting of cross-block SWAP operators, denoted by A_{IB} and A_{CB}, respectively. Note that in-block permutations normalize cross-block involutions. Hence, we find $P = A_{\mathrm{CB}}A_{\mathrm{IB}}B_{\mathrm{IB}}B_{\mathrm{CB}}$.

It follows form XI.1 that $A_{\mathrm{IB}}B_{\mathrm{IB}}$ can be implemented in depth $3r+6$ It remains to determine the depth required to implement the cross-block components A_{CB} and B_{CB}.

Consider a multigraph $\mathcal{G}_A$ with b vertices, corresponding to the b code blocks, and an edge between two vertices for each two-cycle in A_{CB} swapping qubits between the corresponding code blocks (i.e., if m qubits are swapped between a pair of code blocks, the corresponding vertices are connected by m edges). A theorem by Shannon [16] states that a proper edge coloring of a multigraph $\mathcal{G}$ with maximum vertex degree $\Delta(\mathcal{G})$ requires at most $3/2\Delta(\mathcal{G})$ colors. Since A is an involution, it immediately follows that $\Delta(\mathcal{G}_A) \leq r^2$. Therefore, A_{CB} can be implemented in at most $3/2r^2$ rounds, each containing at most a single SWAP operator per code block. By Lemma IX.22 a single logical CNOT operator between code blocks is implementable in depth at most 4, and hence a single SWAP operator between code blocks is implementable in depth at most 12. It follows that A_{CB} is implementable in depth at most $18r^2$, and the same applies to B_{CB}.

We conclude that P is implementable in depth at most $36r^2 + 3r + 6$. □

C. Hadamard circuits

We now turn to implementing arbitrary Hadamard operators in the SHYPS codes. Recall that SHYPS codes have a fold-transversal Hadamard operator $H^{\otimes n}\tau_{n_r}$, which implements the logical Hadamard-SWAP operator $H^{\otimes r^2}\tau_r$ (see Lemma X.19). We may then apply the result on in-block permutations from Proposition XI.1 to τ_r, to cost the isolated all-qubit logical Hadamard.

Corollary XI.5. *The all-qubit logical Hadamard operator H^{r^2} on $SHYPS(r)$ is implemented in depth at most $3r + 10$.*

Proof. By Lemma X.19, $H^{\otimes r^2}$ is implemented by depth-1 all-qubit physical Hadamard $H^{\otimes n}$ gates, composed with τ_{n_r} and τ_r. As τ_{n_r} is a physical SWAP circuit, it is implementable in a depth-3 CNOT circuit. Whereas $\tau_r \in S_{r^2}$ requires depth at most $3r+6$ by Proposition XI.1. □

As for the fault-tolerance of this implementation, we recall from the discussion in Lemma X.5 that τ_n only exchanges qubits lying in different rows and columns of the physical qubit array. Thus a single gate failure during implementation will not lead to a weight two Pauli in the support of a single logical operator, and the circuit is therefore distance preserving. This guarantees the fault-tolerance of $H^{\otimes r^2}$. We note that for qubit architectures with high-connectivity, such a swap may be implemented in practice by simple qubit relabelling. This however leads to only modest constant savings in depth.

Combined with CNOT operators and diagonal gates, the all-qubit Hadamard is sufficient to generate all Clifford operators. However for practical applications it is desirable to more accurately cost H^V for $V \in M_r(2)$, where as before, $V_{i,j} = 1$ indicates a Hadamard gate on qubit (i, j).

To implement such arbitrary Hadamard circuits, the following circuit identities are useful

Lemma XI.6. *Let $V \in M_r(2)$. Then the arbitrary Hadamard circuit H^V is implemented by repeated application of $H^{\otimes r^2}$ and S^V.*

Proof. It's easy to check that $(S \cdot H)^3 = (1 + \mathrm{i})/\sqrt{2} \cdot I_2$, i.e., the identity up to global phase. Hence because H has order 2,

$$(S^V \cdot H^{\otimes r^2})^2 S^V = H^V.$$

□

Lemma XI.7. *Let $V \in M_r(2)$ with $V_{i,j} = 1$ for an even number of entries. Then the arbitrary Hadamard circuit H^V is implemented by repeated application of $H^{\otimes r^2}$, any depth-1 circuit of CZ gates with support strictly on qubits $\{(i, j) \mid V_{i,j} = 1\}$, and a qubit permutation.*

Proof. It is readily verified that the following circuit identity holds:

$$(CZ_{1,2} \cdot (H \otimes H))^3 = SWAP_{1,2}. \tag{28}$$

Since H has order two, one then finds

$$\Big(\big(\prod_{\{a,b\}\in\mathcal{P}} CZ_{a,b}\big)\cdot H^{\otimes r^2}\Big)^2 \cdot \big(\prod_{\{a,b\}\in\mathcal{P}} CZ_{a,b}\cdot SWAP_{a,b}\big) = H^V,$$

where $\mathcal{P}$ is any partition of the set $\{(i, j) \mid V_{i,j} = 1\}$ into subsets of size 2. The lemma now follows from the fact that $\prod_{\{a,b\}\in\mathcal{P}} CZ_{a,b}$ is a depth-1 circuit. □

These two lemmas can be combined into the following statement:

Corollary XI.8. *Let $V \in M_r(2)$. Then H^V is implemented by repeated application of $H^{\otimes r^2}$, any depth-1 circuit of diagonal gates with support strictly on qubits $\{(i, j) \mid V_{i,j} = 1\}$, and a qubit permutation.*

We now introduce a particular set of depth-1 (logical) diagonal Clifford circuits on r^2 qubits:

$$\Xi := \{\begin{pmatrix} I & D\tau_r \\ 0 & I \end{pmatrix} \in Sp_{2r^2} \mid D \in \mathrm{Diag}_{r^2}(2),\, D{\cdot}\tau_r \in SYM_{r^2}(2)\}$$

For convenience, we denote the set of diagonal matrices appearing in the definition above by

$$\xi := \{D \in \mathrm{Diag}_{r^2}(2) \mid D \cdot \tau_r \in SYM_{r^2}(2)\}.$$

Lemma XI.9. *For integer $0 \leq s \leq r^2$, the set Ξ contains an operator with support on exactly s qubits.*

Proof. Recall that the right action of τ_r on any matrix in $M_{r^2}(2)$ is to swap the columns (i,j) and (j,i) for $1 \leq i \neq j \leq r$, while leaving columns (i,i) unchanged. Hence, the following set forms a basis of ξ:

$$\{E_{(i,i),(i,i)} \mid 1 \leq i \leq r\} \cup \{E_{(i,j),(i,j)} + E_{(j,i),(j,i)} \mid 1 \leq i \neq j \leq r\}.$$

These basis elements of ξ correspond to diagonal operators $S_{(i,i)}$ and $CZ_{(i,j),(j,i)}$ in Ξ, respectively. The two subsets contain r and $(r^2-r)/2$ elements, respectively. The lemma now follows by observing that any integer $0 \leq s \leq r^2$ can be written as a sum $a + 2b$ for some $0 \leq a \leq r$ and $0 \leq b \leq (r^2-r)/2$. □

Using circuits from Ξ in conjunction with Corollary XI.8 allows us to execute a Hadamard circuit on any number of qubits inside a $SHYPS(r)$ code block, but we are not free to choose *which* qubits. Conjugating these circuits by a permutation operator, however, allows for the implementation of any arbitrary Hadamard circuit H^V for $V \in M_r(2)$. Given that both the all-qubit logical Hadamard operator and all logical permutations within a $SHYPS(r)$ code block can be implemented in depth $O(r)$, a depth-$O(r)$ implementation of all logical depth-1 diagonal operators in the aforementioned set would guarantee that all logical Hadamard circuits are implementable in depth $O(r)$ as well.

We first prove a small lemma on a generating set for ξ.

Lemma XI.10. *The set of matrices ξ is generated (under addition) by matrices of the form*

$$D \otimes D \quad \text{where } D \in \mathrm{Diag}_r(2).$$

Proof. One can readily verify that $(D \otimes D)\tau_r$ is symmetric for any diagonal matrix $D \in M_r(2)$. Since the set ξ is closed under addition, it suffices to show that we can generate all elements of some basis. Recall that the following set forms a basis of ξ:

$$\{E_{(i,i),(i,i)} \mid 1 \leq i \leq r\} \cup \{E_{(i,j),(i,j)} + E_{(j,i),(j,i)} \mid 1 \leq i \neq j \leq r\}.$$

The lemma follows by observing that $E_{(i,i),(i,i)} = E_{i,i} \otimes E_{i,i}$, and $E_{(i,j),(i,j)} + E_{(j,i),(j,i)} = (E_{i,i} + E_{j,j}) \otimes (E_{i,i} + E_{j,j}) + E_{i,i} \otimes E_{i,i} + E_{j,j} \otimes E_{j,j}$. □

Theorem XI.11. *Let $D \in \xi$. Then there exist $A_1, \ldots, A_p \in GL_r(2)$ for some $p \leq 5r+1$, such that*

$$D = \sum_{i=1}^{p} (A_i \otimes A_i^T).$$

Proof. Using Lemma XI.10, we first decompose $D \in \xi$ as

$$D = \sum_{i=1}^{a} D_i \otimes D_i \tag{29}$$

for $D_i \in \mathrm{Diag}_r(2)$ and $a \leq 2^r - 1$. One can then proceed in a manner similar to the proof of Theorem X.7, starting from equation 15 (note that we would only retain the second term in that expression). In particular, we decompose each diagonal matrix D_i in the basis $\{E_{j,j} \mid 1 \leq j \leq r\}$:

$$D = \sum_{i=1}^{a} \sum_{j=1}^{r} d_{i,j} E_{j,j} \otimes \sum_{k=1}^{r} d_{i,k} E_{k,k},$$

and then collect the terms as follows

$$D = \sum_{j=1}^{r} e_j E_{j,j} \otimes E_{j,j} + \sum_{j=1}^{r} f_j (E_{j,j} \otimes B_j + B_j \otimes E_{j,j}),$$

for some $e_j, f_j \in \mathbb{F}_2$ and $B_j \in \mathrm{Diag}_r(2)$. Provided $r > 3$, one can then invoke Lemma IX.14, Lemma X.13 and Lemma X.15 as in the proof of Theorem X.7 to obtain that $w(D\tau) \leq 5r+1$. When $r = 3$ we check computationally that $w(D\tau) \leq 12$ for all D of the form (29), completing the proof. □

The bound on the weight of the diagonal matrices in ξ specified in Theorem XI.11 implies the following statement when combined with Corollary X.3:

Corollary XI.12. *The logical depth-1 diagonal circuits in Ξ on code $SHYPS(r)$ may be implemented by a physical circuit of depth at most $5r+1$.*

Finally, we can combine the result above with those on in-block logical permutations and the circuit identity in Corollary XI.8 to obtain a $O(r)$ depth scaling for arbitrary Hadamard circuits.

Theorem XI.13. *Let $V \in M_r(2)$. Then the arbitrary logical Hadamard circuit H^V on code $SHYPS(r)$ may be implemented by a physical circuit of depth $21r+15$.*

Proof. Using Corollary XI.8, one can write

$$H^V = S_V \cdot H^{\otimes r^2} \cdot S_V \cdot H^{\otimes r^2} \cdot S_V \cdot \Lambda_V \,,$$

were S_V is a logical depth-1 diagonal circuit with support strictly on qubits $\{(i,j) \mid V_{i,j} = 1\}$, and Λ_V is a depth-1 logical SWAP circuit.

We first focus on the special case $V \in SYM_r(2)$. In this instance, one may choose $S_V \in \Xi$. The depth of S_V and Λ_V follows from Corollary XI.12, and Proposition XI.1, respectively. Note that $H^{\otimes r^2} \cdot S_V \cdot H^{\otimes r^2}$ is an X-diagonal operator which can be executed in the same depth as S_V. Hence, we find in a total depth of $3(5r+1) + 3r + 6 = 18r + 9$.

To obtain the cost for $V \in M_r(2) \backslash SYM_r(2)$, all we need to do is find a $V' \in SYM_r(2)$ with the same number of nonzero entries. H^V and $H^{V'}$ then have the same weight, and can be mapped onto one another by conjugation with an appropriate logical qubit permutation, this is indeed always possible by Lemma XI.9. In particular, for any such V, there exists a $V' \in SYM_r(2)$ and a permutation $P \in S_{r^2}$ such that $fl(V) = fl(V') \cdot P$. One then has $H^V = PH^{V'}P^{-1}$. We may combine the $\Lambda_{V'}$ and P^{-1} into a single permutation, which results in total depth $21r + 15$. □

We end this section by noting that a single logical Hadamard operator can be executed at a constant cost. To see this, recall that (up to a global phase) $S_i \cdot (H_i \cdot S_i \cdot H_i) \cdot S_i = H_i$. Note that we can replace the initial and final S_i operators by some logical depth-1 diagonal operator D_i that contains S_i. We may then use Lemma X.16 to determine the total cost, noting that $H_i \cdot S_i \cdot H_i$, as an X-diagonal operator, can be implemented with the same depth as S_i.

Corollary XI.14. *A single logical qubit Hadamard is implementable in depth* 8 *(*11 *when* $r = 3$*).*

XII. SHYPS COMPILING SUMMARY

In this section we present a novel Clifford decomposition, which we then use to synthesize arbitrary Clifford operators in terms of the logical generators presented in this paper. Contrary to other known Clifford decompositions, the one introduced below does not contain any Hadamard gates. Instead, it only contains X- and Z-diagonal operators, a CNOT circuit, and a depth-1 diagonal circuit. For codes where arbitrary Hadamard circuits are more expensive than depth-1 diagonal gates, as is the case for SHYPS codes, this choice of decomposition can be advantageous to minimize the total depth when synthesizing a Clifford circuit.

Theorem XII.1. *Any Clifford operator* $C \in \mathcal{C}_n$*, can be written as the product*

$$C = DZ \cdot CX \cdot DX \cdot DZ(1)\,, \tag{30}$$

where

- DZ *is a* Z*-diagonal operator,*
- CX *is a CNOT operator,*
- DX *is an* X*-diagonal operator,*
- $DZ(1)$ *is a depth-1* Z*-diagonal operator.*

Remark. One can obtain several related decompositions by altering the order of the constituents. In particular, since CNOT normalizes both the group of Z-diagonal operators and that of X-diagonal operators, one can move CX above through either DZ or DX. One can also invert the entire order by considering the inverse of the decomposition (30) for Clifford operator C^{-1}. Finally, a decomposition of the form

$$C = DX \cdot CX \cdot DZ \cdot DX(1) \tag{31}$$

can be obtained by conjugating the decomposition (30) of Clifford operator $H^{\otimes n} C H^{\otimes n}$ by the all-qubit Hadamard operator $H^{\otimes n}$. In total 12 different decompositions of this kind can be obtained by combining these three tricks.

In order to prove Theorem XII.1, we first prove two supporting lemmas.

Lemma XII.2. *Let* $\chi = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \in Sp_{2n}(2)$*. Then there exists* $\chi' = \begin{pmatrix} I & K \\ 0 & I \end{pmatrix} \in Sp_{2n}(2)$ *(with* $K = K^T$*), such that* $\chi' \cdot \chi$ *has invertible top-left quadrant. Furthermore,* K *can always be chosen such that each of its columns contains at most a single nonzero entry.*

Proof. Denoting the rank of $A \in M_n(2)$ by k, we may right-multiply χ with a symplectic matrix of the form $\begin{pmatrix} U & 0 \\ 0 & U^{-T} \end{pmatrix}$ to perform Gaussian elimination on the first n columns, yielding

$$\chi \cdot \begin{pmatrix} U & 0 \\ 0 & U^{-T} \end{pmatrix} = \begin{pmatrix} A_1 & 0 & B' \\ A_2 & 0 & \\ C_1 & C_3 & D' \\ C_2 & C_4 & \end{pmatrix},$$

where $A_1 \in M_k(2)$ and $\begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$ has rank k. Note that since $\begin{pmatrix} A \\ C \end{pmatrix}$ has rank n, $\begin{pmatrix} C_3 \\ C_4 \end{pmatrix}$ necessarily has rank $n-k$. We can thus perform further Gaussian elimination on the latter $n-k$ columns. In particular, there exists a $V \in GL_{n-k}$ and a permutation $P \in S_n$ such that

$$\begin{pmatrix} C_3 \\ C_4 \end{pmatrix} \cdot V =: P \cdot \begin{pmatrix} C_3' \\ I_{n-k} \end{pmatrix}.$$

For $U' = \begin{pmatrix} I_k & 0 \\ 0 & V \end{pmatrix} U$, we then have

$$\begin{pmatrix} P^{-1} & 0 \\ 0 & P^{-1} \end{pmatrix} \cdot \chi \cdot \begin{pmatrix} U' & 0 \\ 0 & U'^{-T} \end{pmatrix} = \begin{pmatrix} A_1' & 0 & B'' \\ A_2' & 0 & \\ C_1' & C_3' & D'' \\ C_2' & I_{n-k} & \end{pmatrix}.$$

The symplectic condition for the matrix above implies that $C_3'^T A_1' = A_2'$, which in turn implies that A_1' has rank k because rank $\begin{pmatrix} A_1' \\ A_2' \end{pmatrix} = k$. It follows that the matrix $\begin{pmatrix} A_1' & 0 \\ A_2' + C_2' & I_{n-k} \end{pmatrix}$ is full rank, and therefore the matrix

$$\begin{pmatrix} I & \mathrm{Diag}(0, I_{n-k}) \\ 0 & I \end{pmatrix} \begin{pmatrix} P^{-1} & 0 \\ 0 & P^{-1} \end{pmatrix} \cdot \chi \cdot \begin{pmatrix} U' & 0 \\ 0 & U'^{-T} \end{pmatrix}$$

has an invertible top-left quadrant. Since multiplying this matrix from the left with $\mathrm{Diag}(P, P)$ and from the right with $\mathrm{Diag}(U'^{-1}, U'^T)$ does not change that, we conclude that

$$\begin{pmatrix} I & K \\ 0 & I \end{pmatrix} \cdot \chi$$

with $K = P \cdot \mathrm{Diag}(0, I_{n-k}) \cdot P^{-1}$ has an invertible top-left quadrant. K is symmetric and has columns with at most a single nonzero entry, this concludes the proof. □

Note that the symmetric matrix K found in the proof above is always diagonal and therefore the symplectic matrix χ' in the lemma corresponds to a circuit of S gates. However, for any involution $J \in S_n$ obeying $KJ = JK$, the matrix $K' = KJ$ is a valid solution too. With this choice for the off-diagonal block, χ' is the symplectic representation of a depth-1 circuit containing CZ gates.

Lemma XII.3. *Let* $\chi = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \in Sp_{2n}(2)$ *with invertible top-left quadrant* $A \in GL_n(2)$. *Then there exist symplectic matrices*

- $\alpha = \begin{pmatrix} I & 0 \\ E & I \end{pmatrix}$ *with* $E \in M_n(2)$ *and* $E^T = E$,
- $\beta = \begin{pmatrix} F & 0 \\ 0 & F^{-T} \end{pmatrix}$ *with* $F \in GL_n(2)$, *and*
- $\gamma = \begin{pmatrix} I & G \\ 0 & I \end{pmatrix}$ *with* $G \in M_n(2)$ *and* $G^T = G$,

such that $\chi = \alpha \cdot \beta \cdot \gamma$.

Proof. Define matrices α, β and γ as in the lemma above. We compute their product:

$$\begin{aligned} \alpha \cdot \beta \cdot \gamma &= \begin{pmatrix} I & 0 \\ E & I \end{pmatrix} \cdot \begin{pmatrix} F & 0 \\ 0 & F^{-T} \end{pmatrix} \cdot \begin{pmatrix} I & G \\ 0 & I \end{pmatrix} \\ &= \begin{pmatrix} F & FG \\ EF & EFG + F^{-T} \end{pmatrix} . \end{aligned}$$

By choosing $E = CA^{-1}$, $F = A$, and $G = A^{-1}B$, the product above yields

$$\alpha \cdot \beta \cdot \gamma = \begin{pmatrix} A & B \\ C & CA^{-1}B + A^{-T} \end{pmatrix}$$

To prove the lemma, we must show that $CA^{-1}B + A^{-T} = D$, and that our choices of E, F, and G result in valid symplectic matrices α, β, and γ. Recall that the matrix χ is symplectic, i.e., it satisfies the symplectic condition $\chi^T \Omega \chi = \Omega$. This yields the following conditions on the submatrices A, B, C, and D:

$$A^T D + C^T B = I \,, \tag{32}$$

$$A^T C + C^T A = 0 \,, \tag{33}$$

$$B^T D + D^T B = 0 \,. \tag{34}$$

Since A is invertible, condition (32) can be restated as

$$D = A^{-T} + A^{-T} C^T B \,. \tag{35}$$

Similarly, condition (33) can be reformulated as

$$CA^{-1} = A^{-T} C^T \,. \tag{36}$$

Combining these two equations, we find that $D = CA^{-1}B + A^{-T}$, and hence $\alpha\beta\gamma = \chi$.

The symplectic condition for α requires $E = E^T$. For our choice $E = CA^{-1}$, this follows from Eq. (36). The symplectic condition for β requires $F \in GL_n(2)$, which is guaranteed for our choice $F = A$. Finally, the symplectic condition for γ requires $G = G^T$, which for our choice of G becomes $A^{-1}B + B^T A^{-T} = 0$. Since A is invertible, this condition is equivalent to $AB^T + BA^T = 0$. We find the following equalities:

$$\begin{aligned} & AB^T + BA^T \\ &\quad = AB^T + BA^T + AB^T \left(A^{-T}C^T + CA^{-1}\right) BA^T \\ &\quad = AB^T \left(I + A^{-T}C^T BA^T\right) + \left(I + AB^T CA^{-1}\right) BA^T \\ &\quad = AB^T DA^T + AD^T BA^T \\ &\quad = A \left(B^T D + D^T B\right) A^T \\ &\quad = 0 \,, \end{aligned}$$

where the first equality follows from Eq. 36, the third equality follows from Eq. 35, and the last equality follows form Eq. 34. □

Having established Lemma XII.2 and Lemma XII.3, we now proceed to prove Theorem XII.1.

Proof. (Proof of Theorem XII.1) For an arbitrary Clifford operator C with symplectic representation as χ, it follows from Lemma XII.2 and Lemma XII.3 that we can decompose χ as

$$\chi = \begin{pmatrix} I & K \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} I & 0 \\ E & I \end{pmatrix} \cdot \begin{pmatrix} F & 0 \\ 0 & F^{-T} \end{pmatrix} \cdot \begin{pmatrix} I & G \\ 0 & I \end{pmatrix} ,$$

with $K = K^T$, each column of K having a weight of at most one, $E = E^T$, $F \in GL_n(2)$ and $G = G^T$.

These matrices are the symplectic representations of a depth-1 Z-diagonal operator, an X-diagonal operator, a CNOT circuit, and a Z-diagonal operator, respectively.

Recall that the symplectic representation of Clifford operators, $\chi : \mathcal{C}_n \to Sp_{2n}(2)$, was defined to right-act on row-vectors, and therefore the order of multiplication must be inverted, i.e., for $C_1, C_2 \in \mathcal{C}_n$ we have $\chi(C_1C_2) = \chi(C_2)\chi(C_1)$. We have thus obtained the desired decomposition of the Clifford operator C. □

Theorem XII.1 can be used in conjunction with the results for the depth of various types of logical operators found in Sections IX, X and XI to determine an upper bound on the depth required for an arbitrary logical Clifford operator. Note that $DZ(1)$ can always be chosen to be an in-block depth-1 diagonal circuit. Furthermore, it can be chosen to contain no more than r S gates per $SHYPS(r)$ code block, which allows us to use Corollary XI.12 along with two in-block permutations to implement it a depth no larger than $11r + 13$. Also note that since the depth of multi-block CNOT gates was determined up to a logical permutation, we should add the cost of a b-block permutation to it. This, however, does not contribute to the leading order of the depth of general CNOT circuits because multi-block permutations have an implementation of depth $O(r^2)$ while b block CNOT circuits require $O(br^2)$.

We find the following total depth for a Clifford synthesized with this decomposition, using the constructions of logical generators presented in previous sections:

$$\begin{array}{rr} DZ: & br^2 + (b+4)r + 4b - 2 \\ +CX: & (2b+35)r^2 + (2b+2)r + 8b + 2 \\ +DX: & br^2 + (b+4)r + 4b - 2 \\ +DZ(1): & 11r + 13 \\ \hline \text{Total Depth:} & (4b+35)r^2 + (4b+21)r + 16b + 11 \end{array}$$

Note that the above costing of diagonal gates assumes that $r \geq 4$. As shown in Theorem X.7, in-block diagonal gates may incur an additional depth of up to $3r$ when $r = 3$, and incorporating this in the above calculation produces an overall depth upper bound of $64b + 407$ in this case.

We end this section with a short remark on the space overhead required to perform an arbitrary Clifford operator in SHYPS codes. As detailed in sections IX, X and XI, cross-block logical operations in the SHYPS code do not require any space overhead. On the other hand, in-block operations require at most a single auxiliary code block. Hence, performing an arbitrary logical Clifford operator on b code blocks of the SHYPS code (each containing r^2 logical qubits) requires at most b auxiliary code blocks, resulting in a space overhead of br^2.

The following result summarises the discussion above, and provides a comprehensive costing of Clifford operator implementations in the SHYPS codes.

Theorem XII.4. *Let $r \geq 4$. Any Clifford operator on b blocks of $SHYPS(r)$ is implemented fault-tolerantly in depth at most*

$$(4b+35)r^2 + (4b+21)r + 16b + 11.$$

Moreover this implementation incurs a space overhead of at most b auxiliary code blocks, totalling at most bn physical qubits. When $r = 3$, the depth is at most $64b + 407$.

XIII. FAULT-TOLERANT DEMONSTRATION

In this section, we discuss our simulations of quantum memories and Clifford circuits using SHYPS codes. We begin in Section XIII A with a detailed description of the setup used for memory and logic simulations. Next, in Section XIII B we discuss fault-tolerant syndrome extraction (SE), as well as SE scheduling options for the SHYPS codes. In Section XIII D, we give details of the decoder used in the simulations.

A. Numerical simulations

We use two types of numerical simulations in this paper:

- **Memory simulation** to establish performance of a single code block of the $[49, 9, 4]$ and $[225, 16, 8]$ SHYPS codes as compared to surface codes of analogous scale.
- **Logic simulation** of a random Clifford operation decomposed into efficient logical generators applied across two blocks of the $[49, 9, 4]$ SHYPS code.

In what follows, we describe the details of each of these simulations. Although we describe concepts in the context of SHYPS codes, this discussion would also be applicable to simulations of other CSS codes.

1. Memory simulations

Quantum memory simulations are the standard approach to study the circuit-level performance of surface codes, and have recently been extended to analyze QLDPC codes under circuit-level noise [17–21]. Circuits that implement quantum memory experiments follow a specific set of steps:

1. **Transversal Initialization:** Initialize the data qubits in the Z (X) basis and perform a single SE round.
2. **Syndrome Extraction:** Perform a predefined number of SE rounds. In most cases, d SE rounds are used for a code with distance d.
3. **Transversal measurement of data qubits:** Measure the data qubits in the Z (X) basis.

We use the open-source Clifford simulation package `Stim` [22] to build descriptions of these circuits, each annotated with *detectors* and *logical observables*. A detector is the parity of measurement outcome bits in a quantum error correction circuit that is deterministic in the absence of errors, while a logical observable is the linear combination of measurement bits whose outcome corresponds to the measurement of a logical Pauli operator [23]. To describe detectors we label the bits produced during SE with the stabilizer index i, SE round index t, and basis $B \in \{X, Z\}$. Throughout this section, unless explicitly stated, we assume use of the Z basis for initialization and transversal measurement. A similar approach is valid for X basis simulations.

The circuits we use for memory simulation begin with transversal initialization: initializing all data qubits to $|0\rangle$ and performing a single SE round. Note that this does not produce the actual logical all-zero state $|\bar{0}\rangle$, but rather an equivalent version where not all stabilizers have eigenvalue $+1$. Specifically, all Z stabilizers will be in the $+1$-eigenspace, but X stabilizers will be projected to take on random values. For this reason, at the end of this first SE round we only define detectors based on the Z stabilizer measurement results:

$$D_i^{t=0}(Z) = s_i^{t=0}(Z) \ .$$

Next, SE rounds are repeated a predefined number of times. Since the first round of SE forces the X stabilizers to have either a $+1$ or -1 eigenvalue, their expected value during these subsequent SE round is no longer random. We define detectors following these rounds of SE by comparing stabilizer measurement results in between SE rounds:

$$D_i^t(Z) = s_i^t(Z) \oplus s_i^{t-1}(Z), \ D_i^t(X) = s_i^t(X) \oplus s_i^{t-1}(X),$$

where $\oplus$ represents addition modulo 2.

The last step is transversal measurement in the Z basis. In this case, we define detectors by comparing the stabilizer values from the final noisy SE round to the final stabilizer values computed from products of the data qubit measurements. We do not define detectors for X stabilizers as they are unknown following measurement in the Z basis. Logical observables are defined to be the logical Z operators of the SHYPS code. We can write them as

$$L(Z) = \bigoplus_{m_i \in L_Z} m_i \ , \tag{37}$$

where m_i takes values 0 or 1 and represents data qubit measurements produced during transversal readout in the Z basis, L_Z represents the chosen basis for the logical Z operators of the SHYPS code and $i \in \{0, \ldots, n-1\}$.

Once descriptions of memory circuits are constructed, complete with detectors and observables, the decoding problem for memory simulations can be cast within the framework of Detector Error Models (DEM) [23, 24]. DEMs convey information about SE circuits in the form of a detector check matrix $\mathbf{H}_{\text{DCM}}$, a logical observable matrix L, and a vector of priors $\mathbf{p}$. The rows and columns of $\mathbf{H}_{\text{DCM}}$ represent detectors and independent error mechanisms in the circuit, respectively. The entry in position (i, j) of $\mathbf{H}_{\text{DCM}}$ will be 1 iff the i-th detector is flipped (recall that detectors are deterministically 0 in the absence of noise) whenever the j-th error occurs and zero otherwise. Similarly, the rows and columns of L represent the k logical observables we are attempting to preserve with our protocol, and the independent error mechanisms in the circuit, respectively. The entry in position (i, j) of L will be 1 iff the i-th logical observable is flipped by error mechanism j and zero otherwise. The vector of priors $\mathbf{p}$ contains the prior error probability for each of the individual error mechanisms in the circuit.

We use `Stim` to compute $\mathbf{H}_{\text{DCM}}$, L, and $\mathbf{p}$ for memory circuits under standard circuit-level depolarizing noise [25]. This noise model assumes that each element in a quantum circuit is independently either ideal or faulty with probability $1-p$ and p, respectively, where p is the model parameter called the physical error rate. In the context of our circuits for memory simulations, we have the following faulty operations:

- **State preparation:** With probability p, the orthogonal state (e.g., $|0\rangle$ instead of $|1\rangle$) is prepared.
- **Measurement:** With probability p, the classical measurement result is flipped (from 0 to 1 or vice versa).
- **Single qubit gates:** With probability p, apply X, Y, or Z (the specific Pauli operator is picked uniformly at random). An idle qubit in any time step experiences a noisy I gate.
- **Two qubit gates:** With probability p, apply one of the 15 nontrivial 2-qubit Pauli operations on the control and the target qubits $\{IX, IY, IZ, XI, \ldots, ZZ\}$ (the specific Pauli operator is picked uniformly at random).

Aside from using `Stim` to compute the triplet $\mathbf{H}_{\text{DCM}}$, L, and $\mathbf{p}$, we also use it to simulate our circuits efficiently and produce detector and observable samples over different physical noise realizations. This allows us to formulate the decoding problem for quantum memories: provided with $\mathbf{H}_{\text{DCM}}$, $\mathbf{p}$, and the detector samples, the decoder provides an estimate of the real circuit error, which we denote by $\mathbf{c}$. We assess the accuracy of the error correction protocol by comparing the observable samples provided by `Stim` to the logical effect of $\mathbf{c}$, computed as $L \cdot \mathbf{c}$. Repeating this procedure over different physical noise realizations allows us to estimate the logical error rate of quantum memories.

We refer those looking for extensive discussions on detector error models and the circuit-level decoding problem to [23, 24] and [21], respectively.

Detector considerations

In recent works, memory simulations have used either strictly Z-type or strictly X-type detectors [17–21]. More explicitly, $D(X)$ ($D(Z)$) have not been used to decode Z (X) basis experiments [26]. This is primarily due to a substantial increase in the size of the detector check matrix H when both detector types are used, and the fact that most decoders cannot exploit the traces left by Y-errors on both the X and Z stabilizers. To keep comparisons fair, we simulate our memory circuits using only Z-type detectors.

2. Simulations of logical operation

As with memory, we construct descriptions of logical circuits in `Stim`. The following steps outline a logical circuit:

1. **Transversal Initialization:** Initialize the data qubits in the Z (X) basis and perform a round of syndrome extraction.
2. **Logical Operations Interleaved with Syndrome Extraction.** The Clifford unitary of interest is synthesized as a depth-D sequence of Clifford generators. Each of these Clifford generators is applied to the circuit followed by a round of stabilizer extraction. For the logic simulation shown in the main text, the circuit simulated comprises of $2 \times 63 = 126$ stabilizer generators interleaved with syndrome extraction.
3. **Transversal measurement of data qubits:** Measure the data qubits in the Z (X) basis.

In order to annotate our `Stim` circuits with deterministic observables using equation (37), we also apply the inverse of the synthesized operator. This is why the total depth of the circuit we used for our Clifford simulation has twice the depth as the sampled Clifford circuit itself. Noise is added to our descriptions of logical circuits using the standard circuit-level noise model.

It is worth mentioning that it is also possible to perform analysis based on logical measurement results that are random. This was recently done in [27], where the authors override the `Stim` requirement for deterministic observables by defining *gauge detectors* and then interpreting the random measurement results according to their proposed methodology. Running simulations with non-deterministic observables is important for a full characterization. In fact, it might even be more practical in some ways, as it would eliminate the need for inverting the logical action of the Clifford operators we simulate. As is done in most quantum error correction analyses and simulations, we have focused on the case of deterministic observables for simplicity. We leave the extension to simulations with non-deterministic observables for future work.

Detector discovery for logical circuits

As with memories, the first step in a logical circuit is **transversal initialization**, so we define detectors in this first stage to be the Z stabilizer measurement outcomes:

$$D_i^{t=0}(Z) = s_i^{t=0}(Z) \ .$$

The next stage in the circuit involves performing multiple rounds of a logical operation followed by SE. Defining detectors in this stage is more nuanced than for quantum memories [28] because logical operations preserve the codespace but act non-trivially on the stabilizer generators we infer during SE [29]. For instance, the logical fold-transversal Hadamard gate $H^{\otimes n}\tau$ gate (see Section XI) maps X (Z) stabilizers in the $(i-1)$-th round to Z (X) stabilizers in the i-th round. To define valid detectors, we must track the map that each particular logical operator applies on the stabilizer generators. We do so based on a general linear algebra approach that computes the logical action of the operation on the stabilizers and returns the relationship between the stabilizers before and after the logical operation. Alternatively, it is also possible to define detectors in the context of non-trivial logic by using specific update rules for each type of logical operator that may appear in the circuit. In [27], the authors explain how to define detectors following the application of transversal logical H, CNOT, and S gates.

B. Syndrome extraction circuits

In our simulations we do not measure the stabilizers of an SHYPS code directly to perform SE. Instead, we measure the gauge generators of the code and then aggregate those measurement outcomes accordingly to infer the stabilizer measurement results. We use the SE circuits proposed in [30] to implement gauge generator readout for SHYPS codes in our simulations. This type of SE circuit belongs to the family of *bare-auxiliary* gadgets [31], where a single auxiliary qubit is used to readout a stabilizer or gauge generator. In what follows we explain the details of our SE strategy.

1. Circuit fault analysis

The bare-auxiliary method for extracting syndromes provides little protection against error-spread from auxiliary qubits to data qubits. However, this is not an issue for SHYPS codes. Consider the top circuit in Figure 4, which measures the Z-gauge generator $g_{Z,l} = Z_i Z_j Z_k$, where the subscript Z denotes the fact that the gauge generator is a composed purely of single-qubit Z operators and the subscript l is used to represent an arbitrary index. In this figure, numbered boxes are included not

to represent operations, but to differentiate moments in the circuit.

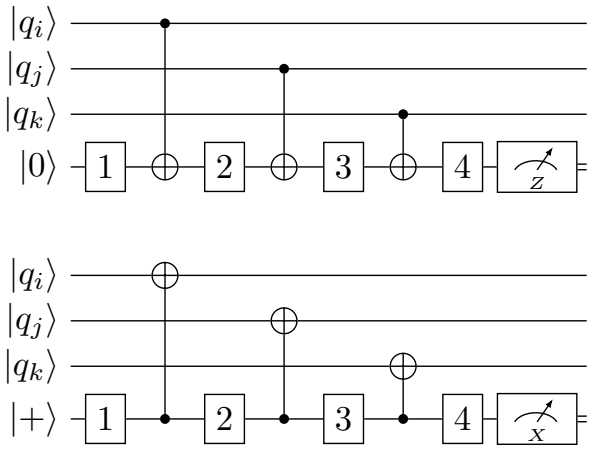


Figure 4. Circuits to measure the eigenvalue of Z-type (top) and X-type (bottom) gauge generators in an SHYPS code. The numbered boxes represent circuit moments.

If a single Z error occurs on the auxiliary qubit, depending on the circuit moment at which it happens (positions $1-4$), the error will propagate to one of the following data qubit errors: $\{Z_iZ_jZ_k, Z_jZ_k, Z_k\}$. Error $Z_iZ_jZ_k$ is the gauge generator $g_{Z,l}$ itself, so it can be ignored. By multiplying the second error Z_jZ_k by $g_{Z,l}$, we end up with Z_i, which is a weight-1 error. The third error is already a weight-1 error. Thus, a single Z error on the auxiliary qubit can lead to at most one Z error on the data qubits, modulo gauge generators. An X error on the auxiliary qubit cannot spread to data qubits, so its only effect is flipping the measurement outcome, which will produce a measurement error.

The circuit is fault-tolerant against Z-errors, as the single Z-faults on the auxiliary qubits are equivalent to at worst single Z-errors on the data qubits. It is fault-tolerant against X errors because they cannot spread to data qubits. Since a single-qubit error on the auxiliary qubits leads to a single-qubit error in the code block, the circuit is fault-tolerant and hence it is protected from high-weight hook errors [32]. A similar analysis can be applied to show that the second circuit shown in Figure 4, which measures the X-gauge generator $g_{X,l} = X_iX_jX_k$, is also fault-tolerant.

Note that X errors on the auxiliary qubits will alter the measurement outcome, indicating the presence of X-errors on the data when there are none. This problem is not unique to this method of syndrome extraction, and is what leads to the frequently seen notion of repeated stabilizer measurements with a number of times matching the code distance.

2. Scheduling SE for SHYPS codes

There are two approaches to scheduling gauge generator measurement circuits for SHYPS codes in the minimum-possible depth. The first relies on exploiting the particular structure of SHYPS codes while the second makes use of the coloration circuit approach from [33]. Both produce gauge generator measurement circuits of the same depth. For the simulations in this paper, we use the coloration circuit to schedule gauge generator readout.

Structure-based scheduling

Recall the structure of the X-gauge generators for the SHYPS code from Section VIII E:

$$G_X = H \otimes I_{n_r},$$

where H is the over-complete $n_r \times n_r$ parity check matrix of the classical (n_r, r, d_r)-simplex code. Moreover, H is the cyclic parity check matrix corresponding to a choice of weight-3 parity-check polynomial $h(x) = 1 + x^{d_1} + x^{d_1+d_2}$. So there exist n_r^2 gauge generators $r_i \otimes e_j$, corresponding to a choice of rows r_i and e_j from matrices H and I_n, respectively. Furthermore, the cyclic structure of H clearly implies that $g = X^{r_i \otimes e_j}$ is supported on qubits $q_{i,j}$, $q_{i+d_1,j}$, and $q_{i+d_1+d_2,j}$, where we denote qubits in a codeblock by q and qubit labels are combined modulo n_r.

Hence, we may schedule measurement of all n_r^2 X-gauge generators in 5 time steps:

1. Preparation of the auxiliary states $a_{i,j} = |+\rangle_{i,j}$
2. Apply $\prod CNOT(q_{i,j}, a_{i,j})$
3. Apply $\prod CNOT(q_{i+d_1,j}, a_{i,j})$
4. Apply $\prod CNOT(q_{i+d_1+d_2,j}, a_{i,j})$
5. Measure auxiliary qubits $a_{i,j}$.

This requires a total $3n_r^2$ physical CNOT gates. Note that other configurations of the CNOT circuit are possible, for example Steps 2, 3, and 4 could be taken in any order. A similar schedule applies for extracting the Z-gauge syndromes. A naive composition of the X and Z-gauge measurement circuits would yield a depth-10 circuit. However, a depth-8 circuit is possible if: X gauge generator measurement qubits are initialized during the last moment of Z gauge generator extraction, and Z gauge generator measurement qubits are measured in the first moment of X gauge generator extraction.

Coloration circuit approach

We can alternatively schedule gauge generator readout circuits for any SHYPS code using the edge-coloring al-

gorithm depicted in Algorithm 1, which is a slightly modified version of the algorithm proposed in [33]. Specifically, this algorithm works with a modified pair of Tanner graphs T'_X and T'_Z, where T'_X and T'_Z are Tanner graphs whose stabilizer check nodes have been substituted by gauge generator nodes while all other aspects of the algorithm remain exactly the same.

Algorithm 1: Edge Coloring Circuit

Data: The Tanner graphs T'_X and T'_Z, as well as their minimum edge colorings $\mathcal{C}_X$ and $\mathcal{C}_Z$
Result: Gauge measurement circuit for X and Z gauge generators of an SHYPS code
`# Z Gauge Generators`
Initialize all auxiliary qubits that will measure Z gauge generators in the $|0\rangle$ state;
for $c \in \mathcal{C}_Z$ **do**
  In the same circuit moment, apply $CNOT_{i\to j}$ gates from the i-th data qubit (control) to the j-th auxiliary qubit (target) supported on an edge $\{i, j\}$ with color c;
`# X Gauge Generators`
Initialize all auxiliary qubits that will measure X gauge generators in the $|+\rangle$ state;
for $c \in \mathcal{C}_X$ **do**
  In the same circuit moment, apply $CNOT_{i\leftarrow j}$ gates from the j-th auxiliary qubit (control) to the i-th data qubit (target) supported on an edge $\{i, j\}$ with color c;

Depth optimality of SE circuits

A critical aspect to consider in the context of SE scheduling is evaluating how *good* the obtained readout circuits are, since there is no guarantee that methods such as the coloration circuit achieve the minimum possible depth [33]. Note how, in principle, it is possible to schedule CNOTs for X- and Z- generators in the same circuit moment such that these interleaved circuits achieve lower depth than the coloration approach, which schedules CNOTs for X- and Z- stabilizers separately. For instance, the SE circuits produced by the edge-coloring approach for surface codes have depth 10, while the interleaved scheduling approach yields circuits of depth 6.

In the context of SHYPS codes, both the structure-based and edge-coloring approaches produce the minimum depth circuits that implement gauge generator readout. This is because SHYPS codes do not allow for interleaving CNOTs involved in X and Z gauge generators, as there are no idling qubits at any given circuit moment of the gauge generator readout circuit. In other words, since every data qubit is involved in each circuit moment, there is no way of pulling CNOT gates through the circuit so that they occur earlier and the depth is reduced.

3. Stabilizer aggregation for SHYPS codes

Once the gauge generators of an SHYPS code have been measured, we obtain the stabilizer measurement outcomes via stabilizer aggregation. We can determine how to perform the aggregation by exploiting the structure of SHYPS codes. Recall from Section VIII E that the X-gauge generators and X-stabilizers are given by

$$G_X = H \otimes I_{n_r},\ S_X = H \otimes G,$$

respectively. We can write $S_X = (I_{n_r} \otimes G)G_X$. This means that the nonzero entries in the rows of the classical generator matrix G indicate which gauge generator measurement results must be combined to compute each stabilizer measurement outcome: *The indices of the nonzero entries in row i of G are the gauge generators whose combination yields stabilizer generator i of the SHYPS code.* The same argument can be followed to compute the Z-stabilizer aggregation.

C. Monte Carlo simulation data

We produce our simulation results by Monte Carlo sampling memory and logic circuits with `Stim` and decoding over different physical error rates. We schedule our Monte Carlo simulations with enough runs to sample at least 100 logical errors for every physical error rate. In this section, we explain how we compute the uncertainties of our simulation results and how we normalize the logical error rate for Figures 1 and 2 in the main text.

1. Uncertainties of simulation results

We use the 95% confidence interval, shown as shaded regions in Figs. 1 and 2 of the main text as well as in Figs. 5 and 6 in Section XIII E of the supplementary material, to portray the uncertainties associated to our simulation results. We compute the 95% confidence interval as follows.

A Monte Carlo simulation that executes n_s runs of independent Bernoulli trials with an observed failure rate of $\tilde{p}$ has a variance

$$\sigma^2 = \frac{\tilde{p}(1-\tilde{p})}{n_s}. \tag{38}$$

Its standard deviation can thus be directly calculated as

$$\sigma = \sqrt{\frac{\tilde{p}(1-\tilde{p})}{n_s}}. \tag{39}$$

Thus, we can establish the uncertainty bounds of the true failure probability p using the observed failure rate

$\tilde{p}$ within a specified confidence interval:

$$p = \tilde{p} \pm z \cdot \sigma, \tag{40}$$

where, for example, $z = 1.96$ corresponds to a 95% confidence level.

Furthermore, when the logical error rates are normalized and scaled based on the number of logical observables v, the number of SE rounds s, and the number of copies of the code m – details of which will be provided in the next subsection – we apply the *propagation of uncertainty* to update the uncertainty bounds accordingly.

2. Normalizing and scaling logical error rates

At the end of our Monte Carlo simulations, we obtain the number of logical error events for a given number of Monte Carlo simulation runs. Let $p_{v,s}$ be the *observed logical error rate* at the end of a simulation that has v logical observables and s syndrome extraction rounds. Note that $p_{v,s}$ is also widely referred to as the *shot error rate*. In Fig. 1 and 2 of the main text we report a different quantity, the *logical error rate per syndrome extraction round*, which we denote as $p_{v,1}$. Next, we describe how $p_{v,1}$ is calculated from $p_{v,s}$, v, and s.

We begin by determining the *observed logical error rate per logical observable*, represented by $p_{1,s}$. Since any error or flip affecting the logical observables is treated as a single logical error event, we can directly define the relationship between $p_{v,s}$, $p_{1,s}$, and v as follows:

$$p_{v,s} = 1 - (1 - p_{1,s})^v. \tag{41}$$

By rearranging (41), we derive

$$p_{1,s} = 1 - (1 - p_{v,s})^{1/v}. \tag{42}$$

Next, we calculate the *logical error rate per logical observable per syndrome extraction round*, denoted by $p_{1,1}$. Consider that each of the logical observables undergo s successive Bernoulli trials, where an outcome of 0 represents success and 1 signifies failure, with each trial having a failure probability of $p_{1,1}$. Consequently, after performing s consecutive trials, the final result is effectively the XOR of the s trials, leading to an observed failure probability of $p_{1,s}$. This is equivalent to stating that each logical observable passes through a binary symmetric channel (BSC) repeated s times in sequence. Therefore, we can express the relationship between $p_{1,s}$, $p_{1,1}$, and s as follows:

$$p_{1,s} = \frac{1 - (1 - 2p_{1,1})^s}{2} \tag{43}$$

and by rearranging (43), we obtain

$$p_{1,1} = \frac{1 - (1 - 2p_{1,s})^{1/s}}{2} \tag{44}$$

Finally, we calculate the *logical error rate per syndrome extraction round*, denoted by $p_{v,1}$. Similar to (41), we can explicitly express the relationship between $p_{v,1}$, $p_{1,1}$, and v as follows:

$$p_{v,1} = 1 - (1 - p_{1,1})^v. \tag{45}$$

By substituting (42) into (44), and substituting the result into (45), we obtain a final expression for the logical error rate per syndrome extraction round, $p_{v,1}$, based on the observed logical error rate $p_{v,s}$, the number of logical observables v, and the number of syndrome extraction rounds s as follows:

$$p_{v,1} = 1 - \left(\frac{1 + \left[2(1 - p_{v,s})^{1/v} - 1\right]^{1/s}}{2} \right)^v. \tag{46}$$

The above calculation is equivalent to the method implemented in `sinter`, a package that integrates directly with `Stim`, to calculate the so-called *piece error rate* from the *shot error rate*.

To produce the scaled surface code data in Fig. 1 and the two-block memory data in Fig. 2 of the main text we calculate the logical error rate per syndrome extraction round $p_{v,1}$ for multiple patches of the same code. We denote this quantity by $p_{v,1}^m$, where m is the number of code patches. We compute the *scaled* logical error rate per syndrome extraction round based on the following equation:

$$p_{v,1}^m = 1 - (1 - p_{v,1})^m. \tag{47}$$

D. Decoding details

The results for SHYPS codes reported in Fig. 1 and 2 of the main text were generated using a proprietary implementation of a sliding window decoder with belief propagation and order 0 ordered statistics decoding (BPOSD0) as the constituent decoder, where we employ a modified version of the min-sum (MS) update rule to improve the decoding convergence. Surface code data was generated using the `pymatching` package [23]. Here we give details about our decoding approach. We provide the specific parameter configurations for MS BPOSD0 (scaling factor and iterations) used for each simulation in tables V and VI.

1. BPOSD

BPOSD is a two-stage decoder that combines BP with OSD. Belief propagation (BP) achieves excellent decoding performance for classical LDPC codes [34, 35]. However, standalone BP decoding may not perform as well for QLDPC codes due to a variety of reasons, including degenerate error patterns, short-cycles in the decoding graph, and split beliefs [36–38]. The quantum-specific

issues standalone BP suffers can be alleviated by providing the output of a BP decoder to an ordered statistics decoder (OSD). OS decoders rely on sorting the reliability of BP outputs and systematizing the decoding graph to improve performance. The resulting combination, referred to as BPOSD, is a high-performing general decoder that has become the state of the art for decoding QLDPC codes [36, 39, 40]. Unfortunately, the improvements in decoding performance provided by BPOSD come with a price. OS decoding involves an unavoidable matrix inversion step, which drastically increases the complexity when compared to standalone BP decoding. Significant effort is being made to curtail the complexity of BPOSD without sacrificing error correction performance [18, 19, 21, 41]. For our simulations we employ so-called fast OSD0, which provides most of the performance improvement while avoiding the complexity increase of higher order OSD [39].

BPOSD parameters

BPOSD offers a variety of tunable parameters that can be adjusted to improve decoding performance:

- **Maximum number of BP iterations, $\mathbf{BP_{it}^{max}}$**: BP is an iterative message-passing algorithm that will not generally terminate on its own [34]. As such, the maximum number of BP iterations represents the number of times that messages are allowed to be exchanged over the decoding graph, and, generally, more iterations result in better performance. If the BP decoder fails to find a converging solution after the maximum number of BP iterations is reached, then its outputs will be provided to the OS decoder for a subsequent attempt at decoding.

- **OSD order**: The order of the OS decoder (also known as search depth), $w \in \mathbb{Z}_0^+$, determines the number of candidates to evaluate as possible solutions to the decoding problem. A larger w incurs higher complexity while providing better error correction performance.

- **MS scaling factor**: For conciseness, we assume reader familiarity with BP decoders and refer readers to [36] for a full discourse on MS BP. For an MS BP decoder to achieve performance comparable to that of the Sum-Product (SP) BP algorithm, the scaling factor included in the check node message update rules must be adjusted based on the current BP iteration index. In [36], the MS scaling factor is defined as $\alpha = 1 - 2^{-i}$, where $i \in \mathbb{Z}^+$ is the BP iteration index. However, we found that due to the large size of the DCMs in our simulations, as well as the large number of BP iterations required in both memory and Clifford circuit simulations, the aforementioned MS scaling factor converged to 1 too quickly. This resulted in performance comparable to unscaled MS BP, which fails to approximate SP BP appropriately. Hence, we slowed down the convergence by introducing an additional damping factor β, which we refer to as the *dynamic scaling factor*, into the MS scaling factor equation [42]:

$$\alpha = 1 - 2^{-\lceil \beta i \rceil}. \tag{48}$$

2. Sliding window decoder

The dimensions of a detector check matrix (DCM), which we denote by $\mathbf{H}_{\text{DCM}}$, grow with the number of SE rounds. This implies that decoding successively deeper circuits with many SE rounds becomes increasingly difficult. For reference, the DCM associated to our logical Clifford simulation has $10,668$ rows and $387,590$ columns. However, empirical evidence shows that DCMs generally possess the characteristics of classical spatially-coupled LDPC codes [20, 43, 44]. In the context of classical coding theory, these codes can be decoded efficiently using a sliding window decoder (SWD) [45]. Instead of decoding across the entire DCM, which may be prohibitively large or add unnecessary decoding latency, SWD performs sequential decoding on smaller *window* decoding regions or subsets of the DCM. Decoding within the decoding regions is conducted with a standard decoding algorithm, such as standalone BP or BPOSD. Once the window is decoded, the decoder shifts forward by a specified *commit* step to the next decoding region of the PCM. To ensure continuity, consecutive windows typically overlap. The overlapping decoding region allows the decoder to account for correlations between adjacent subsets of the DCM. By focusing on smaller, localized regions, SWD reduces the computational load compared to decoding across the entire DCM at once. Memory usage is also significantly reduced since only the subsets of the DCM within the window decoding region need to be stored and processed. Despite its reduced complexity, SWD often achieves near-optimal performance for spatially-coupled LDPC codes as long as the information obtained from localized decoding propagates well to adjacent decoding regions.

The full DCM for multiple SE rounds has the following structure:

$$\mathbf{H}_{\text{DCM}} = \begin{bmatrix} H_{1,1} & & & & & \\ H_{2,1} & H_{2,2} & & & & \\ & H_{3,2} & H_{3,3} & & & \\ & & H_{4,3} & H_{4,4} & & \\ & & & \cdots & \cdots & \\ & & & & H_{t,t-1} & H_{t,t} \end{bmatrix}, \tag{49}$$

where t is the total number of sets of the detectors used in the simulation. In our simulation setup, $t = d + 2$, where d is the number of SE rounds, and the additive factor of 2 comes from the detectors defined after transversal initialization and transversal measurement. This formula

holds for both memory and Clifford circuit simulations.

Given the measured detector values $\mathbf{s}$, the decoding problem becomes finding the most likely error pattern $\widehat{\boldsymbol{e}}$ that satisfies

$$\mathbf{s}^T = \mathbf{H}_{\mathrm{DCM}} \cdot \widehat{\mathbf{e}}^T, \tag{50}$$

where

$$\widehat{\mathbf{e}} = [\widehat{e}_1 \;\; \widehat{e}_2 \;\; \widehat{e}_3 \;\; \cdots \;\; \widehat{e}_t], \tag{51}$$
$$\mathbf{s} = [s_1 \;\; s_2 \;\; s_3 \;\; \cdots \;\; s_t]. \tag{52}$$

Based on the above expansion, we can write the decoding problem as the following set of equations:

$$\begin{aligned} s_1^T &= H_{1,1} \cdot \widehat{e}_1^T \\ s_2^T &= H_{2,1} \cdot \widehat{e}_1^T + H_{2,2} \cdot \widehat{e}_2^T \\ &\vdots \\ s_{t-1}^T &= H_{t-1,t-2} \cdot \widehat{e}_{t-2}^T + H_{t-1,t-1} \cdot \widehat{e}_{t-1}^T \\ s_t^T &= H_{t,t-1} \cdot \widehat{e}_{t-1}^T + H_{t,t} \cdot \widehat{e}_t^T \end{aligned} \tag{53}$$

Since both formulations of the problem are equivalent, we can re-express the original decoding problem into a set of smaller decoding problems.

An SWD(w, c) has two defining parameters: the window size w and the commit size c. The window size w determines the number of sets of detectors that are used as inputs for each decoding round, while the commit size c determines the number of detectors that we fix for committing the error correction. To envisage how an SWD(w, c) works, take SWD$(3, 1)$ as an example and let i denote the index of the decoding round.

In the first decoding round ($i = 1$) of an SWD with $w = 3$, we will take the detector values $[s_1 \;\; s_2 \;\; s_3]$ and apply our chosen decoding algorithm to produce estimated errors $[\widehat{e}_1 \;\; \widehat{e}_2 \;\; \widehat{e}_3]$ using the following segment of the DCM:

$$\mathbf{H}_{\mathrm{DCM}}^{(i=1)} = \begin{bmatrix} H_{1,1} & & \\ H_{2,1} & H_{2,2} & \\ & H_{3,2} & H_{3,3} \end{bmatrix}. \tag{54}$$

For $c = 1$, we commit the correction of the detector values s_1, which means that we commit correction on the estimated error $\widehat{e}_1$. After we commit to the estimated error, we move on to the next decoding round ($i = 2$), where we now take the detector values $[s_2 \;\; s_3 \;\; s_4]$ at the start of the decoding round. However, s_2 involves the estimated error $\widehat{e}_1$ that we committed in the previous decoding round ($i = 1$), and we do not want to double-correct this error. To circumvent this, we update s_2 in this decoding round ($i = 2$) as follows:

$$s_2^{T\prime} = s_2^T + H_{2,1} \cdot \widehat{e}_1^T. \tag{55}$$

Now we perform decoding using the detector values $[s_2' \;\; s_3 \;\; s_4]$ and apply our chosen decoding algorithm to produce estimated errors $[\widehat{e}_2 \;\; \widehat{e}_3 \;\; \widehat{e}_4]$ on the following segment of the DCM:

$$\mathbf{H}_{\mathrm{DCM}}^{(i=2)} = \begin{bmatrix} H_{2,2} & & \\ H_{3,2} & H_{3,3} & \\ & H_{4,3} & H_{4,4} \end{bmatrix}. \tag{56}$$

We use the output of the second decoding round to commit $\widehat{e}_2$: the correction of the detector values s_2. Remember that we have committed to the correction of $\widehat{e}_1$ in the first decoding round, so we do not need to commit on it again. This overlapping decoding procedure is performed sequentially until we reach the window decoding region that involves the detector values $[s_{t-2} \;\; s_{t-1} \;\; s_t]$. Recall that we need to update s_{t-2} to s'_{t-2} accordingly before starting this final round. In the last decoding round ($i = t - 2$), we commit to the estimated errors in the entirety of $[e_{t-2} \;\; e_{t-1} \;\; e_t]$ after applying our chosen decoding algorithm on the following segment of the DCM:

$$\mathbf{H}_{\mathrm{DCM}}^{(i=t-2)} = \begin{bmatrix} H_{t-2,t-2} & & \\ H_{t-1,t-2} & H_{t-1,t-1} & \\ & H_{t,t-1} & H_{t,t} \end{bmatrix}. \tag{57}$$

Finally, we obtain the estimated error $\widehat{\mathbf{e}} = [\widehat{e}_1 \;\; \widehat{e}_2 \;\; \widehat{e}_3 \;\; \cdots \;\; \widehat{e}_t]$ after performing $(t - 2)$ sequential decoding rounds using an SWD$(3, 1)$.

E. Additional simulation results

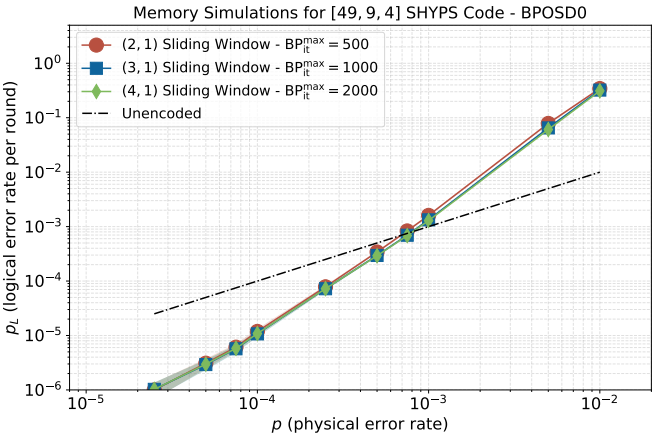


Figure 5. Logical error rate performance of $[49, 9, 4]$ SHYPS code using sliding window decoder.

In this section, we study the effects of varying the SWD configuration on the performance of our decoding approach. We find that decreasing the window size does not noticeably impact the performance of the $[49, 9, 4]$ SHYPS code, even while also decreasing $\mathrm{BP}_{\mathrm{it}}^{\mathrm{max}}$. A constant logical error rate suppression with decreasing window size is indicative of single-shot behaviour, which

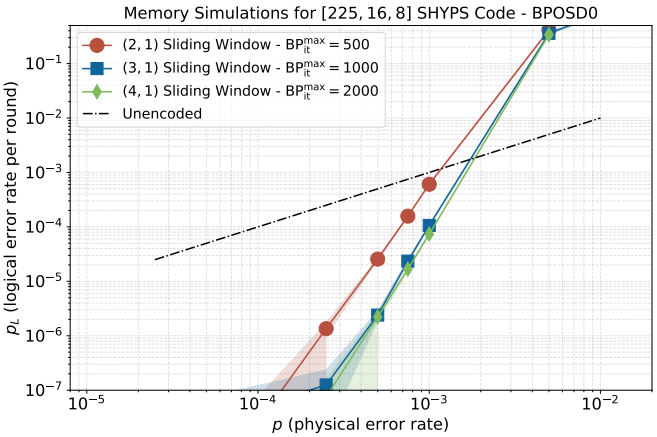


Figure 6. Logical error rate performance of $[225, 16, 8]$ SHYPS code using sliding window decoder.

means it is sufficient to run only a single round of SE between each logical operation. For the $[225, 16, 8]$ SHYPS code, we observe similar single-shot behaviour, with identical logical error rates for $w = 3$ and $w = 4$. The performance drops for $w = 2$ ($\mathrm{BP}_{\mathrm{it}}^{\max} = 500$), but this gap can likely be closed with further optimization of the BP parameters.

[1] N. Rengaswamy, R. Calderbank, H. D. Pfister, and S. Kadhe, in *Proceedings of IEEE International Symposium on Information Theory (ISIT)* (IEEE, 2018) pp. 791–795.

[2] S. Aaronson and D. Gottesman, Phys. Rev. A **70**, 052328 (2004).

[3] A. R. Calderbank and P. W. Shor, Phys. Rev. A **54**, 1098 (1996).

[4] D. Poulin, Phys. Rev. Lett. **95**, 230504 (2005).

[5] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*, 2nd ed. (North-holland Publishing Company, 1978).

[6] M. Grassl and M. Roetteler, in *Proceedings of IEEE International Symposium on Information Theory (ISIT)* (IEEE, 2013) pp. 534–538.

[7] The upper bound of 500 clearly encapsulates all codes that will ever be used in practice. That the result in fact holds for all r is an open conjecture [8].

[8] S. G. I. F. Blake and R. J. Phelps, Proc. Holloway Conf. on Finite Fields (1996).

[9] M. Li and T. J. Yoder, in *Proceedings of IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, 2020) pp. 109–119.

[10] A. O. Quintavalle, P. Webster, and M. Vasmer, Quantum **7**, 1153 (2023).

[11] N. P. Breuckmann and S. Burton, Quantum **8**, 1372 (2024).

[12] H. Sayginel, S. Koutsioumpas, M. Webster, A. Rajput, and D. E. Browne, Fault-Tolerant Logical Clifford Gates from Code Automorphisms (2024), arXiv:2409.18175 [quant-ph].

[13] A. Lempel, SIAM Journal on Computing **4**, 175 (1975).

[14] M. Aschbacher and L. Scott, Journal of Algebra **92**, 44 (1985).

[15] A full discussion of this question is available on Math Stackexchange: Steve D (https://math.stackexchange.com/users/265452/steve-d), Diameter of S_{n^2} with respect to two copies of $S_n \wr S_n$, URL (version: 2024-06-02): https://math.stackexchange.com/q/4926419.

[16] C. E. Shannon, Journal of Mathematics and Physics **28**, 148 (1949).

[17] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, Nature **627**, 778–782 (2024).

[18] A. deMarti iOlius and J. E. Martinez, The closed-branch decoder for quantum LDPC codes (2024), arXiv:2402.01532 [quant-ph].

[19] T. Hillmann, L. Berent, A. O. Quintavalle, J. Eisert, R. Wille, and J. Roffe, Localized statistics decoding: A parallel decoding algorithm for quantum low-density parity-check codes (2024), arXiv:2406.18655 [quant-ph].

[20] A. Gong, S. Cammerer, and J. M. Renes, Toward low-latency iterative decoding of QLDPC codes under circuit-level noise (2024), arXiv:2403.18901 [quant-ph].

[21] S. Wolanski and B. Barber, Ambiguity clustering: An accurate and efficient decoder for qLDPC codes (2024), arXiv:2406.14527 [quant-ph].

[22] C. Gidney, Quantum **5**, 497 (2021).

[23] O. Higgott and C. Gidney, Sparse blossom: correcting a million errors per core second with minimum-weight matching (2023), arXiv:2303.15933 [quant-ph].

[24] P.-J. H. S. Derks, A. Townsend-Teague, A. G. Burchards, and J. Eisert, Designing fault-tolerant circuits using detector error models (2024), arXiv:2407.13826 [quant-ph].

[25] A. G. Fowler, A. M. Stephens, and P. Groszkowski, Phys. Rev. A **80**, 052312 (2009).

[26] Note that all stabilizers are still measured, as omitting the X (Z) stabilizer measurements in a Z (X) memory experiment eliminates the guarantee that the protocol would work to preserve an arbitrary quantum state.

[27] H. Zhou, C. Zhao, M. Cain, D. Bluvstein, C. Duckering, H.-Y. Hu, S.-T. Wang, A. Kubica, and M. D. Lukin, Algorithmic fault tolerance for fast quantum computing (2024), arXiv:2406.17653 [quant-ph].

[28] Note that memories are just simulations of trivial logic.

[29] We do not measure the stabilizer generators of the SHYPS code directly. We measure their gauge generators and use those outcomes to compute the stabilizer measurement results.

[30] S. Bravyi, G. Duclos-Cianci, D. Poulin, and M. Suchara,

Subsystem surface codes with three-qubit check operators (2013), arXiv:1207.1443 [quant-ph].
[31] S. Huang and K. R. Brown, Physical Review Letters **127**, 10.1103/physrevlett.127.090505 (2021).
[32] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Journal of Mathematical Physics **43**, 4452 (2002), https://pubs.aip.org/aip/jmp/article-pdf/43/9/4452/8171926/4452_1_online.pdf.
[33] M. A. Tremblay, N. Delfosse, and M. E. Beverland, Phys. Rev. Lett. **129**, 050504 (2022).
[34] F. Kschischang, B. Frey, and H.-A. Loeliger, IEEE Transactions on Information Theory **47**, 498 (2001).
[35] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988).
[36] J. Roffe, D. R. White, S. Burton, and E. Campbell, Phys. Rev. Res. **2**, 043423 (2020).
[37] P. Fuentes, J. Etxezarreta Martinez, P. M. Crespo, and J. Garcia-Frias, IEEE Access **9**, 89093 (2021).
[38] N. Raveendran and B. Vasić, Quantum **5**, 562 (2021).
[39] P. Panteleev and G. Kalachev, Quantum **5**, 585 (2021).
[40] A. deMarti iOlius, P. Fuentes, R. Orús, P. M. Crespo, and J. Etxezarreta Martinez, Quantum **8**, 1498 (2024).
[41] A. deMarti iOlius, I. E. Martinez, J. Roffe, and J. E. Martinez, An almost-linear time decoding algorithm for quantum LDPC codes under circuit-level noise (2024), arXiv:2409.01440 [quant-ph].
[42] A. A. Emran and M. Elsabrouty, in *Proceedings of IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)* (IEEE, 2014) pp. 892–896.
[43] L. Berent, T. Hillmann, J. Eisert, R. Wille, and J. Roffe, PRX Quantum **5**, 10.1103/prxquantum.5.020349 (2024).
[44] S. Huang and S. Puri, Improved noisy syndrome decoding of quantum LDPC codes with sliding window (2023), arXiv:2311.03307 [quant-ph].
[45] A. R. Iyengar, M. Papaleo, P. H. Siegel, J. K. Wolf, A. Vanelli-Coralli, and G. E. Corazza, IEEE Transactions on Information Theory **58**, 2303 (2011).

Logical Gate	Depth bound
In-block CNOT operator $g_1 \otimes g_2$ for $g_1, g_2 \in GL_r(2)$	0 (qubit relabelling)
Cross-block CNOT operator $\begin{pmatrix} I & g_1 \otimes g_2 \\ 0 & I \end{pmatrix} \in GL_{2r^2}(2)$ for $g_1, g_2 \in GL_r(2)$	1
Arbitrary cross-block CNOT operator $\begin{pmatrix} I & M \\ 0 & I \end{pmatrix} \in GL_{2r^2}(2)$ for $M \in M_{r^2}(2)$	$r^2 + r + 4$
Depth-1 cross-block CNOT operator on two code blocks	$8r + 18$
Arbitrary in-block CNOT operator $A \in GL_{r^2}(2)$	$r^2 + r + 4$
$CNOT_{i,j}$ for qubits i, j in distinct or non-distinct code blocks	4
Arbitrary 2-block CNOT operator (modulo logical permutation)	$3(r^2 + r + 4)$
Arbitrary b-block upper-triangular CNOT operator	$(b-1)(r^2 + r + 4) + r(r+1)/2 + 6$
Arbitrary b-block CNOT operator (modulo logical permutation)	$(2b-1)(r^2 + r + 4)$

Table II. Depth bounds for logical CNOT operators in $SHYPS(r)$. Recall that a CNOT operator on b blocks of r^2 logical qubits is determined by matrices in $GL_{br^2}(2)$ - we give these matrices above where appropriate.

Logical Gate	Depth bound
In-block diagonal generator $B = (g \otimes g^T)\tau_r$ for $g \in GL_r(2)$	1
Single logical S gate $r \geq 4$ $(r = 3)$	6, (9)
$CZ_{i,j}$ for qubits i, j in distinct or non-distinct code blocks	4
Arbitrary in-block diagonal operator $r \geq 4$ $(r = 3)$	$r^2 + 5r + 2$, $(r^2 + 8r + 2)$
Arbitrary cross-block CZ circuit on b blocks	$(b-1)(r^2 + r + 4)$
Depth-1 cross-block CZ circuit on two code blocks	$8r + 18$
Arbitrary b-block diagonal circuit $r \geq 4$ $(r = 3)$	$br^2 + r(b+4) + (4b-2)$, $(16b + 25)$

Table III. Depth bounds for logical diagonal operators in $SHYPS(r)$. Recall that a diagonal operator on b blocks of r^2 logical qubits is determined (modulo Pauli) by a symmetric matrix $B \in SYM_{br^2}(2)$ - we give these matrices where appropriate. Note that τ_r is the permutation matrix exchanging qubit labels $(i, j) \longleftrightarrow (j, i)$.

Logical Gate	Depth bound
All qubit Hadamard (modulo logical SWAP)	4
All qubit Hadamard	$3r + 10$
Single qubit Hadamard gate $r \geq 4$ $(r = 3)$	8, (11)
Arbitrary Hadamard circuit	$21r + 15$
Arbitrary in-block permutation	$3r + 6$
Arbitrary permutation	$36r^2 + 3r + 6$

Table IV. Depth of logical Hadamard-SWAP operators in $SHYPS(r)$

Parameters	SHYPS $[49, 9, 4]$	SHYPS $[49, 9, 4]$	SHYPS $[49, 9, 4]$
Sliding Window	(2,1)	(3,1)	(4,1)
BP Iterations	500	1000	2000
MS Dynamic Scaling Factor	$\frac{1}{250}$	$\frac{1}{250}$	$\frac{1}{250}$
Parameters	SHYPS $[225, 16, 8]$	SHYPS $[225, 16, 8]$	SHYPS $[225, 16, 8]$
Sliding Window	(2,1)	(3,1)	(4,1)
BP Iterations	500	1000	2000
MS Dynamic Scaling Factor	$\frac{1}{250}$	$\frac{1}{250}$	$\frac{1}{250}$

Table V. Decoding parameters for memory simulations

Parameters	SHYPS $[49, 9, 4]$
Sliding Window	(3,1)
BP Iterations	1500
MS Dynamic Scaling Factor	$\frac{1}{250}$

Table VI. Decoding parameters for Clifford Simulation