Automorphism gadgets in homological product codes

Noah Berthusen[1], Michael J. Gullans[1], Yifan Hong*[1,2],
Maryam Mudassar[1], and Shi Jie Samuel Tan†[1]

[1]Joint Center for Quantum Information and Computer Science,
University of Maryland and NIST, College Park, MD 20742, USA
[2]Joint Quantum Institute, University of Maryland and NIST, College Park, MD 20742, USA

August 8, 2025

Abstract

The homological product is a general-purpose recipe that forges new quantum codes from arbitrary classical or quantum input codes, often providing enhanced error-correcting properties. When the input codes are classical linear codes, it is also known as the hypergraph product. We investigate structured homological product codes that admit logical operations arising from permutation symmetries in their input codes. We present a broad theoretical framework that characterizes the logical operations resulting from these underlying automorphisms. In general, these logical operations can be performed by a combination of physical qubit permutations and a subsystem circuit. In special cases related to symmetries of the input Tanner graphs, logical operations can be performed solely through qubit permutations. We further demonstrate that these "automorphism gadgets" can possess inherent fault-tolerant properties such as effective distance preservation, assuming physical permutations are free. Finally, we survey the literature of classical linear codes with rich automorphism structures and show how various classical code families fit into our framework. Complementary to other fault-tolerant gadgets for homological product codes, our results further advance the search for practical fault tolerance beyond topological codes in platforms capable of long-range connectivity.

*yhong137@umd.edu
†stan97@umd.edu
¶The authors are listed in alphabetical order.

Contents

1 Introduction

Quantum error correction, and more generally fault-tolerant quantum computation, promises reliable large-scale computations at the cost of additional resources [KLZ98, ABO08, Kit03]. Since the advent of topological codes [Kit06, BMD06], there has been a significant push to lower the resource overhead for error correction, culminating in the discovery of high-rate quantum low-density parity-check (qLDPC) codes [BE21b, BCG+24]. These codes sacrifice geometric locality in order to more efficiently encode their logical qubits compared to topological codes. It was then shown by Gottesman that constant-rate qLDPC codes can enable fault-tolerant quantum computation with asymptotically constant spatial overhead [Got14] but linear time overhead, compared to the unencoded circuit. The time overhead has since been improved down to polylogarithmic [NP24, TKY24].

Although the above results are encouraging from a theoretical standpoint, in practice there are often a lot of large factors hiding in the constants which may make near-term implementations difficult. On this front, there has been a recent surge of interest in constructing explicit low-overhead gadgets applicable to large families of qLDPC codes, going by the names of bridges [CHRY24], adapters [SJOY25] and extractors [HCWY25] as well as homomorphic measurements [XZZ+24]. These gadgets can measure arbitrary combinations of logical Pauli operators and so are well-suited for the Pauli-based computational model [Lit19]. In this work, we will explore logical gates implemented by unitary circuits such as qubit permutations and transversal gates. A particularly alluring type of gate is an automorphism gate, in which logical operations are performed by simply permuting the physical qubits.

Automorphisms are a reordering of the bits (or qubits) that maps a code to itself. The group arising from the composition of classical error correcting code automorphisms has been a topic of intense study, both as a subject of intrinsic mathematical interest and for applications such as improved decoding [Mac64]. It was later shown that automorphisms could enable low-overhead logical gates in quantum error correcting codes [CRSS97, GR13] as well as assist in their decoding [KSWB25, PCV25]. Unfortunately, logical gates that can be implemented in this way are limited, see Section 3; nonetheless, several recent works have shown that automorphism gates can provide computational speed-ups [XZZ+24, MGF+25] and increased logical gate fidelity [HDSHL24, BDSB24, RAC+24], motivating a deeper understanding of how to obtain quantum codes with useful automorphism groups.

Given a stabilizer code, it is not often obvious how to compute the available symmetries and the resulting logical gates. For classical linear codes, it is known that computing the automorphism group is intimately related to the PERMUTATION CODE EQUIVALENCE problem that can be reduced to the GRAPH ISOMORPHISM problem [PR97]. Sayginel et al. [SKW+25] made progress on this problem for stabilizer codes by providing a method for finding fault-tolerant logical Clifford gates arising from code automorphisms. The method presented there is general and applies to all stabilizer codes. In this work, we focus on Calderbank-Shor-Steane (CSS) stabilizer codes constructed from the product of classical or quantum codes, of which the hypergraph product [TZ14], homological product [FH14, BH14, ZP19, Cam19], lifted product [PK21a, PK21b], and balanced product [BE21a] are examples. Intuitively, since these codes are "products" of underlying codes, one might expect them to inherit the automorphism symmetries of their input classical codes. For the cases of hypergraph and homological products, we show that this intuition is nearly correct, and we provide a systematic method of obtaining a quantum error-correcting code with a desired automorphism group. In particular, we show that the automorphism groups of the underlying classical codes can be utilized by the resulting quantum code to yield a set of logical gates which can be implementing on the physical level through a combination of qubit permutations and a (not necessarily shallow) circuit on a subsystem of the data qubits. Because of the possible existence of these circuits, we refer to these logical gates as *automorphism gadgets*, as opposed to *automorphism gates*, which consist solely of qubit permutations. To this end, we show that a stricter notion of code automorphism, known as Tanner graph automorphism, does give rise to true automorphism gates in the corresponding product codes. It might be of interest to some readers that Lockhart and González-Guillén have studied the Stabilizer State Isomorphism problem which can be loosely defined as the problem that decides whether there exists some qubit permutation that sends a stabilizer state to another stabilizer state. They provided evidence that the Stabilizer State Isomorphism is an intermediate problem for QCMA and showed that the polynomial hierarchy collapses if the problem is QCMA-complete [LG17]. While the problem is not exactly the same as the problem of identifying true automorphism gates for product codes, the complexity of the Stabilizer State Isomorphism problem is likely to suggest hardness for the problem of our interest.

1.1 Related works

We acknowledge some previous and related works regarding quantum code automorphisms and their utility in fault-tolerant computation. Grassl and Roetteler [GR13] introduced the concept of utilizing code automorphisms to perform fault-tolerant logical gates in quantum stabilizer codes. They also show that combining automorphisms with transversal CNOT can significantly enlarge the set of fault-tolerant gates for certain CSS codes. Chao and Reichardt [CR18a] later showed how the full logical Clifford group could be realized in the $[\![15, 7, 3]\!]$ quantum Hamming code through a combination of permutations (automorphisms), transversal H, and round-robin CZ circuits. Gong and Renes [GR24] demonstrate a similar result using quantum Reed-Muller codes and propose an implementation in 2D neutral atom arrays.

Hong et al. [HMKL24] presented a construction of inherited automorphism gates for hypergraph product codes in the context of concatenation. In this work, we provide a broad theoretical framework that encapsulates and builds upon this earlier work. In particular, we generalize to higher-fold homological products and prove statements about inherent fault tolerance of the resulting gadgets.

More recently, Malcolm et al. [MGF+25] show how automorphisms of classical codes lift to automorphisms in a subsystem variant of the hypergraph product code. They leverage these inherited automorphisms to achieve a $O(1)$ Clifford compiling ratio using the classical Simplex code family. However, it should be noted that although the weight of the gauge check operators of the subsystem hypergraph product code can be constant, the weight of the stabilizer generators is proportional to the code distance, and so the LDPC property of the stabilizers must be sacrificed if one desires more than a constant code distance. Nonetheless, they show that small-length instances achieve competitive performance under circuit-level noise compared to surface codes of similar distance.

Guyot and Jaques [GJ25] provide generic bounds on the parameters of any quantum CSS code based on the size and structure of its automorphism group. Since the product constructions in our work produce CSS codes, all of our results are broadly encompassed by their bounds. When restricted to the class of product codes, we are able to derive tighter bounds by examining automorphism limitations of the input codes themselves.

1.2 Summary of results

Our contributions can be sorted into three categories: general recipes, analyses of fault tolerance, and a survey of input codes.

Our primary result is explicit recipes of logical gadgets for homological product codes based on the underlying automorphism structures of their input codes; the details of the construction can be found in Section 4. Typically, a code is defined with respect to its codewords, in which case a code automorphism is defined as a permutation symmetry of the codewords. However, the homological product is defined with respect to the parity-check matrices of the classical seed codes rather than their codewords and so depends strongly on the choice of parity-check matrices. Additionally, both the parity-check matrix and its transpose are put on an equal footing in the homological product; as a consequence, we will need to keep track of how the columns *and* rows transform in the parity-check matrix from a code automorphism. Geometrically, the Tanner graph of the homological quantum code will be (up to node relabeling) the Euclidean graph product of the input Tanner graphs. As such, one can always arrange the qubits and parity checks of the homological product code in a rectangular layout so that all qubit-check interactions are strictly "horizontal" or "vertical". Permutations of the (qu)bits in the input codes now become permutations of rows and columns in the homological product code; see Figure 4 for an illustration.

Theorem 1.1 (Inherited automorphism gadgets (informal))**.** *For a homological product code with two input codes, which can either be classical linear codes or quantum CSS codes, the number of distinct automorphism gadgets is the product of the number of distinct automorphisms (or gadgets) of the input codes. In particular, the group generated by these automorphism gadgets is a direct product of the automorphism groups of the input codes.*

By "distinct", we mean that each automorphism gadget corresponds to a distinct transformation (gate) on our logical qubits. This property does not generically hold for all codes, but we show in Theorem 3.5 and Corollary 3.6 that this property does indeed hold for classical linear codes with dual distance at least 3.[1] Our recipes are very general and include the case where an input quantum code can itself be a homological product code with automorphism gadgets. These gadgets can then be lifted once again by further homological products. From this recursion, the above theorem also includes the case where we take k-fold homological products of k input codes.

Within the category of fault tolerance, we derive several results concerning both the circuit depth and the spread of correlated errors for our automorphism gadgets.

Corollary 1.2 (Circuit depth preservation (informal))**.** *Suppose we have a code possessing an automorphism gadget with circuit depth T. Then its lifted gadget after a homological product will also have circuit depth T.*

The above corollary provides us with an incredible amount of flexibility when it comes to designing homological product codes with desired automorphism gadgets; namely, it implies that we do not need to worry about any conflicts between different input codes.

A generic code automorphism corresponding to a permutation transformation on the columns of a parity-check matrix may not necessarily be a permutation transformation on the rows. We identify a subgroup of code automorphisms, which we call Tanner graph automorphisms, where the above property does indeed hold. The name stems from the fact that when a code automorphism

[1] An analogous result for quantum CSS codes has been recently shown in [GJ25].

acts as a permutation on both sides of the parity-check matrix, it can be interpreted as an automorphism of the associated Tanner graph. An automorphism gadget corresponding to an input Tanner graph automorphism acts strictly as a permutation and so becomes a code automorphism of the homological product code. In fact, we show an even stronger result, which is that these Tanner graph automorphisms lift to Tanner graph automorphisms themselves.

Theorem 1.3 (Tanner graph automorphism inheritance (informal))**.** *Suppose we have a code possessing a Tanner graph automorphism of its parity-check matrix. Then its lifted gadget after a homological product is also a Tanner graph automorphism consisting solely of qubit permutations on the physical level.*

The above theorem also implies that when we recursively perform additional homological products, these Tanner graph automorphisms carry over to every iteration of the product, irrespective of the other input codes. Finally, for the case where one of the input codes is classical, we show that our automorphism gadgets can be made distance-preserving upon ignoring (gauging) some of the logical qubits. Specifically, we show that circuit-level correlated errors do not decrease the effective fault distance, under the assumption that physical permutations do not spread errors. This assumption is typically valid when permutations are performed by physical movement rather than a series of entangling gates, like in trapped ion or neutral atom architectures. We were only able to show a partial result (Theorem 5.18) for the case where *both* input codes are quantum.

Theorem 1.4 (Effective distance preservation (informally Theorems 5.5 and 5.12))**.** *For hypergraph product and (quantum $\times$ classical) homological product codes, if the input automorphism gadgets preserve the effective fault distance, then so will their lifted versions after the homological product.*

The above theorem says that even when our automorphism gadgets involve circuits that can spread errors, these correlated errors do not affect our homological product code in the most adversarial way. Along the way, we prove some technical statements about logical sector weights in repeated homological products with classical input codes (Proposition 5.11) and quantum input codes (Proposition 5.17), which may be of independent interest.

We also explore the rich literature of classical linear codes and survey various code families that best suit our framework. These codes can be fed into the homological product to either produce a new CSS code or upgrade the logical gadgets of an existing CSS code. Table 1 presents a summary of the classical codes we study in this work, including information about their parameters, automorphism groups, and method of implementation. To the best of our knowledge, the only previously studied examples of homological product codes with automorphism gates were hypergraph products of quasicyclic codes [XZZ+24]. For near-term implementation, the most promising code families we have identified are group-algebra codes and cycle codes. Both of these code constructions are capable of producing LDPC codes with constant rate and growing distance. For classical codes based on group algebras, we show that they always possess a Tanner graph automorphism group isomorphic to the underlying group in the algebra, even when the group is non-abelian. For the cycle codes, any automorphism of the underlying graph is a Tanner graph automorphism. Importantly, the Tanner graph automorphism group for the cycle codes can sometimes be much larger than the length of the code, unlike the lower bound that we have for the group-algebra codes. An interesting observation with these two code constructions is that they often produce rank-deficient parity-check matrices. Nonetheless, the redundant parity checks are crucial to the Tanner graph automorphism structure; although removing these redundant parity checks will not alter the code automorphism group, it will change both the structure of the homological product as well as the implementation of its corresponding automorphism gadgets.

Classical code $\mathcal{C}$	Code parameters $[n,k,d]$	$\subseteq \mathrm{Aut}(\mathcal{C})$	LDPC?	Aut gadget
Cycle $(\mathsf{G}=(V,E))$	$\big[\,\lvert V\rvert, \lvert E\rvert - \lvert V\rvert + 1, \mathrm{girth}(\mathsf{G})\,\big]$	$\mathrm{Aut}(\mathsf{G})$	✓	perm
Group-algebra $(\mathcal{G})$	$[\,\lvert\mathcal{G}\rvert, ?, ?\,]$	$\mathcal{G}$	✓	perm
$\mathcal{G}$-lift (ℓ, m)	$\big[\,m\lvert\mathcal{G}\rvert, \geq (m-\ell)\lvert\mathcal{G}\rvert, ?\,\big]$	$\mathcal{G}$	✓	perm
Hamming	$[\,2^r-1, 2^r-r-1, 3\,]$	$\mathrm{GL}_r(\mathbb{F}_2)$	✗	perm + circuit
Simplex	$[\,2^r-1, r, 2^{r-1}\,]$	$\mathrm{GL}_r(\mathbb{F}_2)$	✓	perm + circuit
Reed-Muller $\mathrm{RM}(r,m)$	$\big[\,2^m, \sum_{i=0}^{r}\binom{m}{i}, 2^{m-r}\,\big]$	$\mathrm{GA}_m(\mathbb{F}_2)$	✗	perm + circuit
Punctured $\mathrm{RM}^*(r,m)$	$\big[\,2^m-1, \sum_{i=0}^{r}\binom{m}{i}, 2^{m-r}-1\,\big]$	$\mathrm{GL}_m(\mathbb{F}_2)$	✗	perm + circuit

Table 1: A survey of classical linear codes and their relevant properties for the construction of automorphism gadgets.

Lastly, we present three explicit, finite-length quantum codes built from the hypergraph and homological product that possess interesting automorphism structures. In the first example, we show that the existence of automorphism gadgets in a $[\![52,10,3]\!]$ hypergraph product code can reduce the overhead of external resources when generating logical Clifford gates. In the second and third examples, we use automorphism gadgets in a $[\![48,6,3]\!]$ hypergraph product code and a $[\![288, 9, (d_X = 16, d_Z = 3)]\!]$ code to increase the addressability of transversal CZ and CCZ gates based on cohomology invariants [BDET24].

1.3 Discussion and outlook

Our general-purpose recipes for automorphism gadgets are directly compatible with other fault-tolerant gadgets for homological product codes, such as distance-preserving single-ancilla syndrome extraction [MC25, TS24], single-shot error correction [Cam19], fold-transversal Clifford gates from ZX-dualities [QWV23, BB24], extractor systems for logical Pauli measurements [HCWY25], and transversal Clifford and non-Clifford gates from cohomological invariants [BDET24, HKZ24][2]. In addition, it has been shown that the row-column structure of the parity checks of homological product codes is particularly amenable to implementation in neutral atom arrays [XBAP+24, CFD+24]; since our gadgets follow the same row-column structure, their implementation should be similarly suitable in this platform.

We note that these gadgets also have an interpretation in condensed matter physics. A quantum stabilizer code implicitly defines a commuting and frustration-free Hamiltonian comprised of a uniform sum over all parity checks and whose ground state subspace is the codespace. If the code is LDPC and the distance is macroscopic (i.e. the code is degenerate) then we say the model is topological [BHM10]. Excitations and energies are related to error syndromes and syndrome weights respectively. There has been a large body of recent works which analyzes qLDPC codes and their physical implications through this lens, ranging from code constructions [RK23, RK24] to stability against perturbations [LGAB24, YL24, RKL+24] and thermal fluctuations [HGL25, GKZ24, PRBK24]. Furthermore, many fault-tolerant gadgets can be understood from the framework of higher-form symmetries [ZSP+24]. From this physical perspective, a code automorphism is a spectrum-preserving permutation symmetry of the code Hamiltonian, and a

[2] We note that the more general construction in the last section of [HKZ24] requires a post-modification by Freedman and Hastings [FH21], and it is unclear to us whether any automorphism gadgets can survive this transformation.

Tanner graph automorphism is an exact permutation symmetry of the Hamiltonian; e.g. translational invariance. Notably, this permutation symmetry can act nontrivially on the ground state subspace. We also point out that this notion of permutation symmetry acts on spatial positions and is different than the usual symmetries which act on internal degrees of freedom. Since we show that Tanner graph automorphisms lift to Tanner graph automorphisms in homological product codes, we provide a method to construct topological stabilizer models exhibiting nontrivial permutation symmetry structures beyond translational invariance (but may have long range).

There are a few outstanding open questions and future directions that we summarize below.

Open Problem 1 (Effective distance preservation for quantum input codes)**.** *Do the automorphism gadgets of (quantum × quantum) homological product codes inherit the same effective distance preservation from their input gadgets?*

We could only achieve a partial result (Theorem 5.18) on the effective fault distance of our automorphism gadgets in the (quantum × quantum) homological product case. This problem is related to the lower bound on the minimum distances of generic (quantum × quantum) homological product codes, which has not yet been shown to match the upper bound [ZP20]. It would be interesting to see if our effective distance lower bound could be tightened to match these upper bounds. This would allow us to feed in multiple quantum CSS codes into the homological product without having to worry about their underlying gadgets decreasing the effective distance of the code.

Open Problem 2 (Full logical Clifford group)**.** *Does there exist any high-rate, LDPC homological product code family that can realize the full logical Clifford group using a combination of transversal, fold-transversal and constant-depth automorphism gadgets?*

By "high rate", we do not mean constant rate, but rather simply $k = \omega(1)$ in order to discount surface and color codes, which are also LDPC but have $k = O(1)$. Note that transversal, fold-transversal and constant-depth automorphism gadgets (assuming permutations are free) all only cost $O(1)$ spacetime overhead in their implementation and so are extremely desirable from a computational standpoint. A somewhat discouraging observation is that when we take a k-fold homological product of k input codes, the total number of automorphism gadgets is a k-fold product that can scale at most exponentially with k;[3] however, the number of logical qubits k also increases exponentially with k, and so the size of the Clifford group modulo the Pauli group $\mathcal{C}_k/\mathcal{P}_k \simeq \mathrm{Sp}_{2k}(\mathbb{F}_2)$ scales *doubly* exponentially with k. Fortunately, some previous works have demonstrated that (non-LDPC) quantum codes with seemingly small automorphism groups can be enlarged to fill the entire logical Clifford group when supplemented with only a few additional transversal and fold-transversal gates [GR13, GR24, MGF+25][4]. It has yet to be seen if any qLDPC stabilizer code family satisfies the same property. We hope that the framework presented in this paper leads to conducive progress along this research front.

Open Problem 3 (Full universality without distillation)**.** *Does there exist any high-rate, LDPC homological product code family that can natively realize universal logic using a combination of transversal, fold-transversal, constant-depth automorphism gadgets, and measurements without magic state distillation?*

[3] Noting that the largest possible automorphism group size on k logical qubits is $|\mathrm{GL}_k(\mathbb{F}_2)| \propto 2^{k^2}$.

[4] One can directly transfer the results of [MGF+25] to the usual hypergraph product, but the resulting automorphism gadgets will not have constant depth as far as we are aware.

No-go theorems on transversal gates tell us that universality cannot be achieved solely with the first three operations [EK09, WVSB22]. We added the word "natively" to discount the protocols that selectively teleport logical qubits from a homological product code to a surface code and perform all logic using surface-code gadgets [XBAP+24]. It has been recently shown that a k-fold hypergraph product code (of k classical codes) can admit transversal gates in the kth level of the Clifford hierarchy [BDET24, HKZ24]; it has been further proven that this lower bound is tight [FZLL25]. Given that universal computation without distillation is possible with 3D surface codes [VB19], perhaps similar techniques could be useful for high-rate homological product codes.

Lastly, we comment on possible generalizations of our framework to other product constructions such as lifted products [PK21b] and balanced products [BE21a]. Both of these constructions can be viewed as taking the usual hypergraph product of two input codes sharing a common symmetry and then factoring out a "diagonal" subgroup of the resulting product symmetry group, with the balanced product being more general. For example, taking a balanced product of two classical group-algebra codes (Section 6.2) with the same base symmetry (Tanner graph automorphism) group results in a two-block group-algebra quantum code [LP24] if we factor out the full diagonal subgroup. If the base symmetry group is abelian, e.g. in the bivariate bicycle codes [BCG+24], then it is still a subgroup of the automorphism group of the resulting code. If the base group is non-abelian, then we generically only retain its center after the symmetry reduction [LP24], which can be significantly smaller than the original base group itself. However, if our non-abelian base group contains a nontrivial normal subgroup, then we can choose to factor out by this smaller subgroup rather than the full group itself. The remaining structure associated with the quotient group then resembles the usual hypergraph product, and our input Tanner graph automorphisms carry over with respect to this quotient group. It would be interesting to explore whether any structure is preserved for the more generic automorphism gadgets that are not Tanner graph automorphisms.

1.4 Outline

The rest of the work is structured as follows. In Section 2, we introduce automorphisms of classical and quantum CSS codes as well as the main quantum code construction studied in this work: the homological product. We then review and discuss in Section 3 several limitations to the automorphism group and resulting codes in both the classical and quantum setting, with special emphasis to codes on graphs. In Section 4, we showcase how automorphisms of classical and quantum error-correcting codes lift to logical gadgets in their homological products. In Section 5, we prove effective distance preservation for two out of three of our gadgets and describe how they can be incorporated into a fault-tolerant architecture. In Section 6, we survey a handful of classical error-correcting codes that possess interesting automorphism groups and examine how they fit into our framework. In Section 7, we present three explicit quantum codes where we show that automorphism gadgets can introduce new logical gates or enhance existing logical gates.

2 Preliminaries

2.1 Notation

Let $[n]$ denote the set $\{1, 2, \ldots, n\}$. Let $\mathbf{x} \in \mathbb{F}_2^n$ be an n-dimensional column vector whose Hamming weight $|\mathbf{x}|$ is given by the number of non-zero entries in the vector. In the case where the entries of an n-dimensional column vector can be arranged in some grid of size $a \times b$, we denote $\mathbb{M}[\mathbf{x}] \in \mathbb{F}_2^{a\times b}$ as the matrix representation, or reshaping, of the vector $\mathbf{x}$ that respects the grid arrangement of $\mathbf{x}$. We sometimes drop the $\mathbb{M}$ notation for brevity when it is clear that we are referring to the matrix representation of the vector. In addition, we define $|\cdot|_r$ and $|\cdot|_c$ as the number of rows and columns respectively of some matrix with non-zero entries. In other words, $|\mathbb{M}[\mathbf{x}]|_r$ is the number of row vectors in $\mathbb{M}[\mathbf{x}]$ where there exists some column with a non-zero entry; with slight abuse of notation, we will sometimes write $|\mathbf{x}|_r \equiv |\mathbb{M}[\mathbf{x}]|_r$ when the underlying 2D arrangement is obvious. We denote the submatrix of the matrix representation of $\mathbf{x}$ with the set of rows with row indices $A \subseteq [a]$ and columns with column indices $B \subseteq [b]$ as $\mathbb{M}[\mathbf{x}][A, B] = \mathbf{x}[A, B]$. We can similarly let $\mathbf{x}[A] \in \mathbb{F}_2^{|A|}$ for $A \subseteq [n]$ denote the restriction of the column vector $\mathbf{x}$ to the row indices in A. Let $\oplus$ and $\otimes$ denote the direct sum and tensor product, respectively. It should also be clear from context as to whether an addition operation is done modulo 2. When we are simultaneously discussing both classical and quantum codes, we reserve lowercase letters for quantities related to the classical code and uppercase letters for those of the quantum code, unless stated otherwise.

2.2 Classical linear codes and automorphisms

A classical binary $[n, k, d]$ linear code $\mathcal{C}$ is defined to be a k-dimensional subspace of an n-dimensional vector space $\mathbb{F}_2^n$ such that the Hamming weight of any nonzero codeword $\mathbf{c} \in \mathcal{C}$ satisfies $|\mathbf{c}| \geq d$. We typically say that k *logical* bits are encoded in n *physical*, or data, bits. In particular, $\mathcal{C}$ can be identified as the row space of a generator matrix G where the rows of G form a *logical basis* for $\mathcal{C}$. Alternatively, $\mathcal{C}$ can also be identified by a (non-unique) parity-check matrix H such that $\mathcal{C} = \ker H$ i.e., $H\mathbf{c} = 0$ for all $\mathbf{c} \in \mathcal{C}$ and equivalently $HG^{\mathsf{T}} = 0$. An automorphism on a classical code is defined to be a permutation of coordinates, or bit positions, that stabilizes the codespace. When $\mathcal{C}$ is linear, then it implies that the generator matrix G is related to the permuted G' under a linear transformation. To be precise, let σ be an $n \times n$ permutation matrix. Then in order for the rows of $G' = G\sigma$ and G to span the same codespace (row space), there must exist an invertible $k \times k$ matrix V such that $G' = VG$. In other words, we have the following *automorphism condition*:

$$G\sigma = VG\,, \tag{1}$$

with $\sigma \in \mathrm{S}_n$ and $V \in \mathrm{GL}_k(\mathbb{F}_2)$. Conversely, if there exists such a σ and V satisfying (1), we say σ is an automorphism of $\mathcal{C}$. Since V can potentially map codewords to other codewords, we say it enacts a *logical operation* on the code. Since the composition of permutations is another permutation, and the composition of invertible matrices is an invertible matrix, if σ, σ' are two code automorphisms, then so is $\sigma\sigma'$. In addition, since σ and V are both invertible, we can multiply (1) on the right by σ^{-1} and on the left by V^{-1} to get $G\sigma^{-1} = V^{-1}G$, which tells us that σ^{-1} is also a code automorphism. Combined with the associativity of matrix multiplication, we see that the automorphisms $\{\sigma\}$ form a finite subgroup, which we we call the automorphism group or $\mathrm{Aut}(\mathcal{C})$ for short, of the permutation group S_n. Similarly, the collection of corresponding logical operations $\{V\}$ forms a subgroup, which we call the *logical* automorphism group $\mathcal{A}(\mathcal{C})$, within the group of all logical operations $\mathcal{L}$. The automorphism condition (1) induces a correspondence $\phi : \mathrm{Aut}(\mathcal{C}) \mapsto \mathcal{A}(\mathcal{C})$ through the generator matrix G.

Proposition 2.1. $\phi : \mathrm{Aut}(\mathcal{C}) \to \mathcal{A}(\mathcal{C})\,;\, \sigma \mapsto V$ *is a group homomorphism.*

Proof. We first show that ϕ is a well-defined mapping. Since G has full (row) rank by definition, there exists a (non-unique) right-inverse G^{-1} that we can choose for our construction of ϕ. All that remains is to show that for any $\sigma \in \mathrm{Aut}(\mathcal{C})$, we have $\phi(\sigma) \in \{V\} \subseteq \mathcal{A}(\mathcal{C})$. Suppose there exist $V, V' \in \mathcal{A}(\mathcal{C})$. Using the fact described above the proposition statement, we can compose σ with its inverse and use (1) to get $G\sigma\sigma^{-1} = G = V'V^{-1}G$. By multiplying G^{-1} on the right in the previous expression, we obtain $\mathbb{1} = V'V^{-1}$ and hence $V = V'$ as desired. To show that ϕ is a group homomorphism, it suffices to show that $\phi(\sigma\sigma') = \phi(\sigma)\phi(\sigma')$:

$$\phi(\sigma\sigma') = VG\sigma'G^{-1} = VV'GG^{-1} = VV' = \phi(\sigma)\phi(\sigma')\,. \tag{2}$$

□

For a linear code $\mathcal{C}$, we can also construct its orthogonal complement, or dual, $\mathcal{C}^\perp$. A generator matrix of $\mathcal{C}^\perp$ is a parity-check matrix of $\mathcal{C}$. Since σ is an orthogonal transformation satisfying $\sigma\sigma^\mathsf{T} = \mathbb{1}$, if $\sigma \in \mathrm{Aut}(\mathcal{C})$, then we also have that $\sigma \in \mathrm{Aut}(\mathcal{C}^\perp)$ and vice versa, which implies that $\mathrm{Aut}(\mathcal{C})$ and $\mathrm{Aut}(\mathcal{C}^\perp)$ are isomorphic groups, i.e. $\mathrm{Aut}(\mathcal{C}) \simeq \mathrm{Aut}(\mathcal{C}^\perp)$. Thus, all the aforementioned statements about $\mathrm{Aut}(\mathcal{C})$ and $\mathcal{A}(\mathcal{C})$ also hold for $\mathrm{Aut}(\mathcal{C}^\perp)$ and $\mathcal{A}(\mathcal{C}^\perp)$. In particular, we have the dual group homomorphism $\phi^\perp : \mathrm{Aut}(\mathcal{C}^\perp) \mapsto \mathcal{A}(\mathcal{C}^\perp)$ and the dual automorphism condition

$$H\sigma = WH \tag{3}$$

with $W \in \mathrm{GL}_m(\mathbb{F}_2)$.

The quantum codes we are interested in will be defined with respect to the parity-check matrices of the input codes (rather than the generator matrices). As such, a stricter notion of code automorphism called *Tanner graph automorphism* will be useful later, and so we present it here.

Definition 2.2 (Graph automorphism)**.** *Given a (hyper)graph $\mathsf{G} = (V, E)$, an automorphism of G is defined as a permutation of the vertices $\pi(V)$ such that for a collection of vertices $\{v_j\} \subset V$, we have $\pi\big(\{v_j\}\big) \in E$ if and only if $\big(\{v_j\}\big) \in E$.*

In other words, two graphs G and G' are isomorphic when one can be obtained from the other upon relabeling vertices and edges. For a linear code, its parity-check matrix $H \in \mathbb{F}_2^{m\times n}$ gives the vertex-edge incidence matrix of its Tanner graph. An automorphism of the Tanner graph given by H is then given by the relation

$$H = \sigma_\mathrm{e} H \sigma_\mathrm{v}\,, \tag{4}$$

where σ_e and σ_v are permutation matrices whose respective rows and columns encode the edge and vertex relabeling. Comparing (4) with the dual automorphism condition (3), we see that Tanner graph automorphisms are automorphisms where W is a permutation matrix[5]. We denote $\mathcal{T} \subseteq \mathcal{A}$ the subgroup of code automorphisms corresponding to Tanner graph automorphisms. For a given generator matrix G, we can always construct a parity-check matrix for which all code automorphisms are Tanner graph automorphisms:

Proposition 2.3 (All code automorphisms as Tanner graph automorphisms)**.** *Every binary linear code $\mathcal{C}$ admits a parity-check matrix H such that (4) holds for all $\sigma_\mathrm{v} \in \mathrm{Aut}(\mathcal{C})$.*

Proof. The rows of any parity-check matrix of $\mathcal{C}$ must span $\mathcal{C}^\perp$, which has dimension $n-k$. Let $H \in \mathbb{F}_2^{(2^{n-k}-1)\times n}$ be a parity-check matrix whose rows consist of all $2^{n-k}-1$ nonzero vectors in $\mathcal{C}^\perp$. Since $\sigma_\mathrm{v} \in \mathrm{Aut}(\mathcal{C}) \simeq \mathrm{Aut}(C^\perp)$, it must preserve $\mathcal{C}^\perp$ by definition and so can only permute the $2^{n-k}-1$ rows of H (note that the all-zero vector remains invariant and is not included in H). □

Although Proposition 2.3 provides us a method to make all code automorphisms Tanner graph automorphisms, the cost is a dense parity-check matrix with an exponentially large number of rows. In the product constructions we will be interested in, the number of physical qubits as well as the weight of the stabilizer checks will be directly tied to the row and column weights of the input parity-check matrices, and so it will be pertinent to ensure that these weights do not blow up for the sake of fault tolerance.

[5] In [SKW+25], these are also called “matrix automorphisms”.

2.3 Quantum CSS codes and automorphisms

A $[\![n,k,d]\!]$ quantum stabilizer code encodes k logical qubits in n data qubits and can detect errors with weight less than d. The codespace is defined by its stabilizer group $\mathcal{S}$, an abelian subgroup of the Pauli group $\mathcal{P}_n$ on n qubits [Got97]. A generating set of $\mathcal{S}$, containing Pauli check operators, is typically used to detect and correct errors. The codespace is defined as the simultaneous $+1$ eigenspace of these check operators (and hence the entire stabilizer group). It is customary to define a stabilizer code in this "dual" picture, rather than listing out the logical Pauli operators themselves in analogy to a generator matrix, since a logical Pauli may have many different equivalent representations related by an element of the stabilizer group, a phenomenon called stabilizer degeneracy. A quantum Calderbank-Shor-Steane (CSS) code is a stabilizer code whose check operators are purely X-type and Z-type Paulis [CS96, Ste96]. As such, its stabilizer group also factorizes into X-type and Z-type subgroups. For a quantum CSS code, we can write its X-type and Z-type parity checks in terms of a binary $2n \times 2n$ block-diagonal matrix

$$H_{\text{css}} = \begin{pmatrix} H_X & \mathbf{0} \\ \mathbf{0} & H_Z \end{pmatrix}, \tag{5}$$

where H_X and H_Z label the supports of the X-type and Z-type parity checks respectively. The interpretation of (5) is that the first n columns label X-type support and the last n columns Z-type support.

A permutation $\sigma \in \mathrm{S}_n$ of the qubits now corresponds to the operator $\sigma \oplus \sigma = \operatorname{diag}(\sigma, \sigma)$. The (dual) automorphism condition for CSS codes can then be expressed as

$$\begin{pmatrix} W_X & \mathbf{0} \\ \mathbf{0} & W_Z \end{pmatrix} \begin{pmatrix} H_X & \mathbf{0} \\ \mathbf{0} & H_Z \end{pmatrix} = \begin{pmatrix} H_X & \mathbf{0} \\ \mathbf{0} & H_Z \end{pmatrix} \begin{pmatrix} \sigma & \mathbf{0} \\ \mathbf{0} & \sigma \end{pmatrix}, \tag{6}$$

or

$$W_X H_X = H_X \sigma \tag{7a}$$
$$W_Z H_Z = H_Z \sigma\,. \tag{7b}$$

2.4 Chain complexes

In this section, we introduce some basic definitions of chain complexes that are useful for our subsequent analysis.

Definition 2.4 (Chain Complexes)**.** *A chain complex $\mathcal{C}_*$ over a field $\mathbb{F}_q$ consists of a sequence of $\mathbb{F}_q$-vector spaces $(C_i)_{i\in\mathbb{Z}}$. These vector spaces are related to each other by linear boundary maps $\left(\partial_i^{\mathcal{C}} : C_i \to C_{i-1}\right)_{i\in\mathbb{Z}}$ satisfying $\partial_{i-1}^{\mathcal{C}} \circ \partial_i^{\mathcal{C}} = 0$ for all $i \in \mathbb{Z}$. When it is clear from context, we omit the superscript and subscript for the linear boundary maps. By choosing an appropriate basis, we refer to the basis elements of C_i as i-cells and an arbitrary vector in C_i as an i-chain. If there exists bounds $\ell < m \in \mathbb{Z}$ such that for all $i < \ell$ and $i > m$ have $C_i = 0$, then we may truncate the sequence and say that $\mathcal{C}$ is the $(m-\ell+1)$-term chain complex*

$$\mathcal{C}_* = \left(C_m \xrightarrow{\partial_m} C_{m-1} \xrightarrow{\partial_{m-1}} \dots \xrightarrow{\partial_{\ell+1}} C_\ell \right).$$

We furthermore define the following (standard) vector spaces for $i \in \mathbb{Z}$:

$$\textit{the space of } i\textit{-cycles: } Z_i(\mathcal{C}) \coloneqq \ker(\partial_i) \subseteq C_i, \tag{8}$$

$$\text{the space of } i\text{-boundaries: } B_i(\mathcal{C}) \coloneqq \operatorname{im}(\partial_{i+1}) \subseteq C_i, \tag{9}$$

$$\text{the space of } i\text{-homology: } \mathcal{H}_i(\mathcal{C}) \coloneqq Z_i(\mathcal{C}) / B_i(\mathcal{C}). \tag{10}$$

The cochain complex $\mathcal{C}^*$ associated to $\mathcal{C}_*$ is the chain complex with vector spaces $\left(C^i \coloneqq C_i\right)_{i\in\mathbb{Z}}$ and linear boundary maps given by the coboundary maps $\left(\delta_i = \partial_{i+1}^{\intercal} : C^i \to C^{i+1}\right)_{i\in\mathbb{Z}}$ obtained by transposing all the boundary maps of $\mathcal{C}_*$. Similarly, by choosing an appropriate basis, we refer to the basis elements of C^i as the i-cocells and an arbitrary vector in C^i as an i-cochain. Thus, the cochain complex is defined as such:

$$\mathcal{C}^* = \left(C^m \xleftarrow{\delta_{m-1}} C^{m-1} \xleftarrow{\delta_{m-2}} \dots \xleftarrow{\delta_\ell} C^\ell\right).$$

We can analogously define the spaces of cohomology $\mathcal{H}^i(\mathcal{C}) = Z^i(\mathcal{C})/B^i(\mathcal{C})$, cocycles $Z^i(\mathcal{C}) = \ker(\delta_i)$, and coboundaries $B^i(\mathcal{C}) = \operatorname{im}(\delta_{i-1})$.

Classical linear codes can be described by 2-term chain complexes where the two vector spaces are the spaces of bits and checks respectively such that these spaces are related to each other by a linear boundary map that can be written as the check matrix H. It is also a well-known fact that quantum CSS codes can be described with a 3-term chain complex by associating the X stabilizers, qubits, and Z stabilizers with the three consecutive vector spaces in a 3-term chain complex. An easy way to see the connection is to relate the condition $\partial_{i-1} \circ \partial_i = 0$ to $H_X H_Z^{\intercal} = 0$. We now state an important definition that relates the X and Z distances of a quantum CSS code to its associated chain complex.

Definition 2.5. *For a chain complex $\mathcal{C}$, the i-systolic distance $d_i(\mathcal{C})$ and the i-cosystolic distance $d^i(\mathcal{C})$ are defined as*

$$d_i(\mathcal{C}) = \min_{c\in Z_i(\mathcal{C})\setminus B_i(\mathcal{C})} |c|, \qquad d^i(\mathcal{C}) = \min_{c\in Z^i(\mathcal{C})\setminus B^i(\mathcal{C})} |c|.$$

Suppose we associate the X stabilizers, qubits, and Z stabilizers to the $\mathbb{F}_2$-vector spaces C_0, C_1, and C_2, then the X and Z logical operator representatives are given by the basis elements of the 1-cohomology space and 1-homology space respectively. The X and Z distances of the quantum code are then $d^1(\mathcal{C})$ and $d_1(\mathcal{C})$.

The framework of chain complexes provides us with a diagrammatic way to write the (dual) automorphism condition for classical linear codes (3). Interpreting the $m \times n$ parity-check matrix $H : B \mapsto S$ as a linear map from the binary vector spaces of bit flips B to syndromes S, a code automorphism is a pair of matrices $\sigma \in \mathrm{S}_n$ and $W \in \mathrm{GL}_m(\mathbb{F}_2)$ such that the following diagram is commutative:

$$\begin{tikzcd} B \arrow[r, "H"] \arrow[d, "\sigma"'] & S \arrow[d, "W"] \\ B \arrow[r, "H"] & S \end{tikzcd} \tag{11}$$

For quantum CSS codes, we have the binary vector spaces of X-syndromes, qubit errors and Z-syndromes, organized into a 3-term chain complex $S_X \xrightarrow{H_X^{\intercal}} Q \xrightarrow{H_Z} S_Z$. The CSS automorphism condition (7) can then be rephrased as having matrices $\sigma \in \mathrm{S}_n$, $W_X \in \mathrm{GL}_{m_X}(\mathbb{F}_2)$ and $W_Z \in \mathrm{GL}_{m_Z}(\mathbb{F}_2)$ such that the following diagram is commutative:

$$\begin{tikzcd} S_X \arrow[r, "H_X^{\intercal}"] \arrow[d, "W_X"'] & Q \arrow[r, "H_Z"] \arrow[d, "\sigma"] & S_Z \arrow[d, "W_Z"] \\ S_X \arrow[r, "H_X^{\intercal}"] & Q \arrow[r, "H_Z"] & S_Z \end{tikzcd} \tag{12}$$

2.5 Homological product of chain complexes

This section states the basic notions of the homological product of chain complexes. Note that the homological product is also known as the tensor product of chain complexes.

Definition 2.6 (Homological Product)**.** *For chain complexes $\mathcal{A}$ and $\mathcal{B}$, the homological product $\mathcal{C} = \mathcal{A} \otimes \mathcal{B}$ is the chain complex given by the vector spaces*

$$C_i \coloneqq \bigoplus_{j \in \mathbb{Z}} A_j \otimes B_{i-j}$$

and the boundary maps

$$\partial_i^{\mathcal{C}} \coloneqq \bigoplus_{j \in \mathbb{Z}} \left(\partial_j^{\mathcal{A}} \otimes \mathbb{1} + \mathbb{1} \otimes \partial_{i-j}^{\mathcal{B}}\right).$$

We now state a well-known result about the homological product.

Proposition 2.7 (Künneth Formula)**.** *Let $\mathcal{A}$ and $\mathcal{B}$ be chain complexes over a field $\mathbb{F}_q$, each with a finite numbers of nonzero terms. Then for every $i \in \mathbb{Z}$,*

$$\mathcal{H}_i(\mathcal{A} \otimes \mathcal{B}) \cong \bigoplus_{j \in \mathbb{Z}} \mathcal{H}_j(\mathcal{A}) \otimes \mathcal{H}_{i-j}(\mathcal{B}).$$

Furthermore, for $a \in Z_j(\mathcal{A})$ and $b \in Z_{i-j}(\mathcal{B})$, the isomorphism above maps

$$a \otimes b + B_i(\mathcal{A} \otimes \mathcal{B}) \mapsto (a + B_j(\mathcal{A})) \otimes (b + B_{i-j}(\mathcal{B})).$$

The Künneth formula allows us to compute the number of logical qubits in the homological product code based on properties of the input codes. By analyzing the rank of the homology group, we can determine the number of logical qubits and the structure of their logical Pauli operators.

We note that the definitions stated in Sections 2.4 and 2.5 are with respect to the multi-sector theoretic version of chain complexes. Some readers might be familiar with the single-sector theoretic version introduced in [BH14] which contains an unbounded chain complex where every vector space and boundary map is *identical*. While the single-sector theoretic version simplifies some of the homology analysis and allows the associated quantum code to break the $O(\sqrt{n})$ distance barrier, the code ceases to have good distance bounds once we impose the LDPC condition on it [GG24]. In order to have meaningful fault tolerance [Got14] and interesting automorphisms that arise from the inheritance of *distinct* boundary maps, we restrict our study to homological product codes of the multi-sector theoretic version defined in Sections 2.4 and 2.5.

2.6 Hypergraph product codes

One of the first constructions of quantum CSS codes from classical linear codes with large rate and distance is the hypergraph product (HGP) [TZ14]. The HGP construction is no different from taking the homological product of two 2-term chain complexes that correspond to two classical linear codes. Consider two classical linear codes with parameters $[n_1, k_1, d_1]$ and $[n_2, k_2, d_2]$ and parity-check matrices h_1 and h_2, each of dimension $m_i \times n_i$. Taking a Kronecker product of their chain complexes, we obtain the product complex

$$\begin{tikzcd}
 & B_1 \otimes S_2 & & & S_Z \\
B_1 \otimes B_2 \arrow[ur, "\mathbb{1}\otimes h_2"] & & S_1 \otimes S_2 \arrow[ul, "h_1^\top\otimes\mathbb{1}"'] & & Q \arrow[u, "H_Z"] \\
 & S_1 \otimes B_2 \arrow[ul, "h_1^\top\otimes\mathbb{1}"] \arrow[ur, "\mathbb{1}\otimes h_2"'] & & & S_X \arrow[u, "H_X^\top"]
\end{tikzcd} \tag{13}$$

where S_X, Q, S_Z are the binary vector spaces of X-syndromes, qubit errors and Z-syndromes respectively. The CSS parity-check matrices of the corresponding HGP code $\mathrm{HGP}(h_1, h_2)$ can then be read off from (13):

$$H_X = \left(h_1 \otimes \mathbb{1}_{n_2} \mid \mathbb{1}_{m_1} \otimes h_2^\top\right) \tag{14a}$$

$$H_Z = \left(\mathbb{1}_{n_1} \otimes h_2 \mid h_1^\top \otimes \mathbb{1}_{m_2}\right). \tag{14b}$$

The X-type and Z-type stabilizer checks mutually commute because $H_X H_Z^\top = 2h_1 \otimes h_2^\top = 0$ over $\mathbb{F}_2$. The resulting quantum code $\mathrm{HGP}(h_1, h_2)$ has parameters [TZ14]

$$n = n_1 n_2 + m_1 m_2 \tag{15a}$$

$$k = k_1 k_2 + k_1^\top k_2^\top \tag{15b}$$

$$d_Z = \min\left(d_1, d_2^\top\right) \tag{15c}$$

$$d_X = \min\left(d_2, d_1^\top\right), \tag{15d}$$

with the quantum code distance $d = \min(d_X, d_Z)$. Geometrically, the HGP code can be arranged in a rectangular layout; see Fig. 1. Adopting the naming convention from [QC22], we call the n^2 qubits on the left side of (14) *left* qubits, Λ_{L}, and the other m^2 qubits *right* qubits, Λ_{R}. In this form, it can be seen that the Z-type stabilizer checks consist of qubits from a single row of Λ_{L} and a single column of Λ_{R}. X-type stabilizer checks correspondingly have qubits from a single row in Λ_{R} and a single column in Λ_{L}. The 2D surface (toric) code can be viewed as a HGP code of two classical 1D repetition codes, each embedded on a line (ring). When h_1 and h_2 have full rank, their transpose codes are trivial ($k_1^\top = k_2^\top = 0$) and so the above code parameters simplify to $k = k_1 k_2$ and $d = \min(d_1, d_2)$.

From the classical parity-check matrices h_1 and h_2, we can also construct their associated generator matrices g_1 and g_2, which label a basis of their corresponding codewords and satisfy $hg^\top = 0$. In the HGP code, similar to the data qubits, the logical qubits can be partitioned into $k_1 k_2$ left logical qubits and $k_1^\top k_2^\top$ right logical qubits. A canonical basis for the left logical $\overline{X}$ and $\overline{Z}$ operators of the HGP code can then be chosen as [QC22, QWV23]

$$G_{Z,\mathrm{L}} = \left(g_1 \otimes \{\mathbf{e}_i\} \mid \mathbf{0}\right) \tag{16a}$$

$$G_{X,\mathrm{L}} = \left(\{\mathbf{e}_j\} \otimes g_2 \mid \mathbf{0}\right), \tag{16b}$$

where $\{\mathbf{e}_i\} \notin \mathrm{im}(h_2^\top)$ and $\{\mathbf{e}_j\} \notin \mathrm{im}(h_1^\top)$ are n-dimensional unit vectors. This basis choice ensures that our different logical operators are not stabilizer equivalent. Note that $\mathrm{rank}(h_1) = n_1 - k_1$, and so k_1 instances of $\mathbf{e}_j$ are guaranteed to exist (and k_2 likewise for $\mathbf{e}_i$). Hence (16) is a valid Pauli basis for all $k_1 k_2$ left logical qubits. The canonical basis for the right logical qubits follows similarly but with the transpose codes. By putting the classical generator matrices into standard form, i.e.

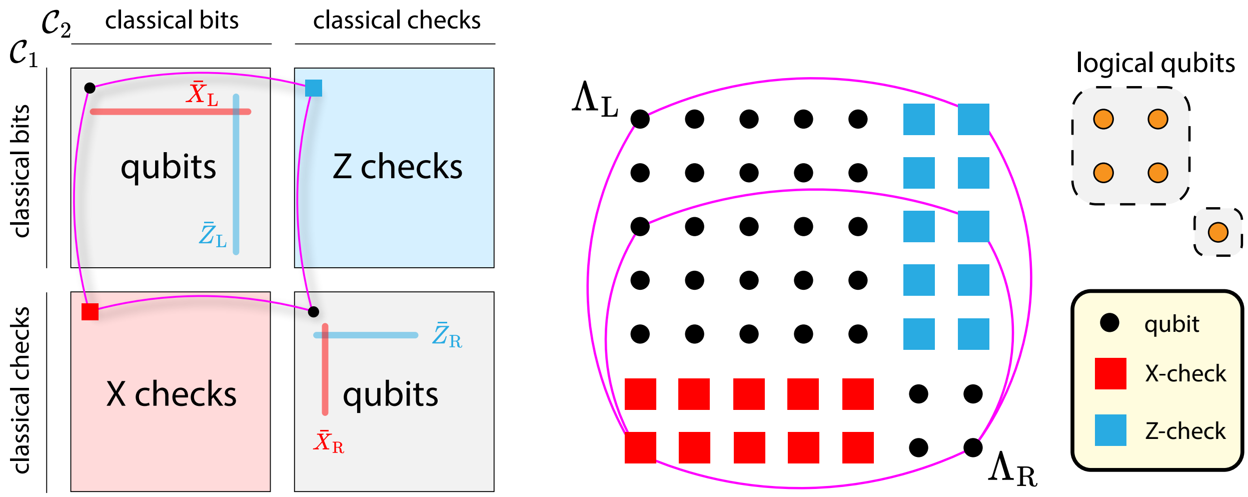

Figure 1: The rectangular layout of a hypergraph product code is depicted. Magenta edges denote a subset of qubit-check connections. The logical qubits are partitioned into those which reside in the left sector $\Lambda_{\rm L}$ or right sector $\Lambda_{\rm R}$ of data qubits.

$g = (\mathbb{1}_k \,|\, A)$, we can further ensure that logical $\overline{X}$ and $\overline{Z}$ operators either have no intersection or intersect at a single qubit indexed by (i, j) in (16); we can use this single qubit intersection to label our k_1k_2 left logical qubits, see the top right corner of Figure 1. Note that this further refinement is not strictly necessary since for any row of (16b), one can always find a corresponding combination of rows of (16a) which anticommutes with it while commuting with all other rows. Observe that the canonical logical $\overline{Z}$ operators are supported on single columns of $\Lambda_{\rm L}$, and the canonical logical $\overline{X}$ operators are supported on single rows of $\Lambda_{\rm L}$. The same holds true for the right logical qubits but on $\Lambda_{\rm R}$. Note that in the canonical basis, left logical operators have minimum weight $\min(d_1, d_2)$ while right logical operators have minimum weight $\min(d_1^{\sf T}, d_2^{\sf T})$. In fact, if we ignore the right logical qubits and treat them as gauge qubits, the code distance simplifies to $d = \min(d_1, d_2)$. This previous statement can be formalized by examining the stabilizer structure on the left and right sectors, which we relegate to Section 5.1.

For the classical codes we will be interested in, often $d^{\sf T}$ will be smaller than d, and so it will be beneficial to ignore the right logical qubits to not diminish code distance, at the cost of a reduced rate. For the remainder of this paper, we will focus on the left logical qubits unless stated otherwise.

2.7 Higher-fold homological product codes

In this section, we discuss the construction of higher-fold homological product codes as well as review some of their well-known facts. We note that there are two distinct constructions of homological product codes:

1. Homological product between a quantum CSS code and a classical linear code
2. Homological product between two quantum CSS codes

We discuss these two constructions in the following two sections separately.

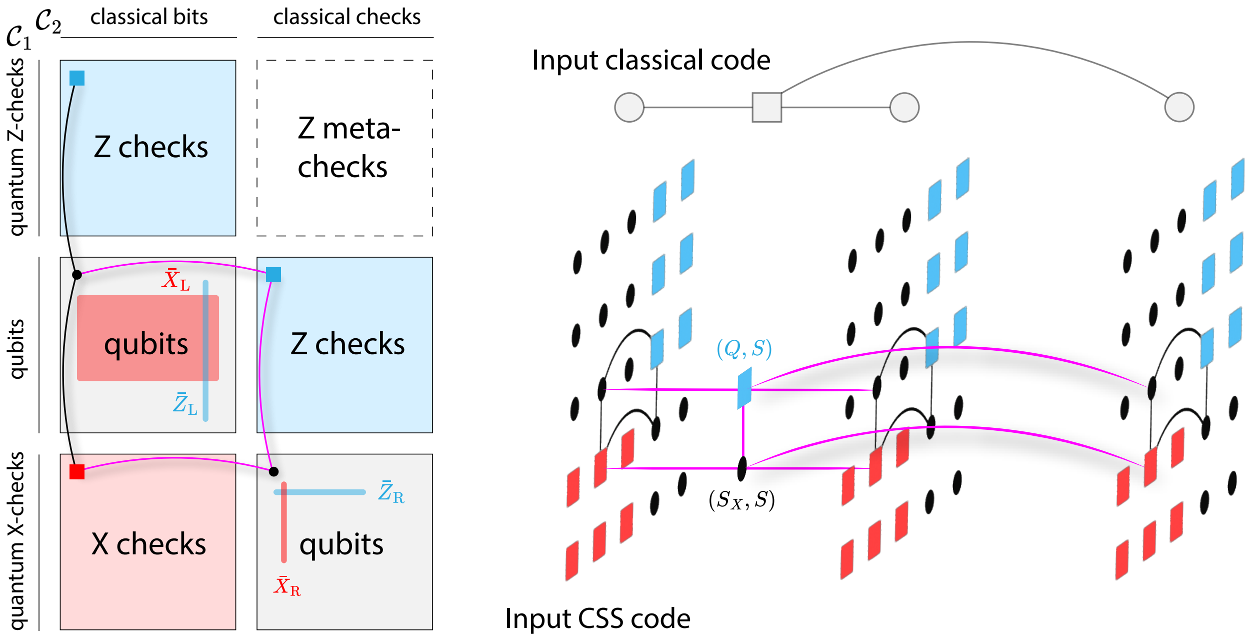

Figure 2: **Left:** The layout of a (quantum × classical) homological product code is depicted. **Right:** New qubits and checks are labeled according to their associated spaces in the product complex (17), and their new connections are colored in magenta.

2.7.1 Homological product: quantum × classical

For an input $[\![n_\mathrm{Q}, k_\mathrm{Q}, (d_X, d_Z)]\!]$ quantum CSS code with parity-check matrices $H_X, H_Z \in \mathbb{F}_2^{m_\mathrm{Q}\times n_\mathrm{Q}}$ and a $[n_\mathrm{c}, k_\mathrm{c}, d_\mathrm{c}]$ classical code with parity-check matrix $h \in \mathbb{F}_2^{m_\mathrm{c}\times n_\mathrm{c}}$, we can take the Kronecker product of their underlying chain complexes and associate the product complex with a *homological product code* [FH14, BH14, ZP19, Cam19], which can be thought of as higher-dimensional/fold generalizations of the usual hypergraph product. For example, the 3D surface code can be interpreted as a homological product of the 2D surface code with a 1D classical repetition code. Alternatively, it could also be viewed as a 3-fold hypergraph product of classical repetition codes i.e., a 3D hypergraph product code. Similar to the freedom in choosing which logical Pauli type extends to a membrane in the 3D surface code, we will have two choices for constructing the homological product code. Without loss of generality, we will restrict to the case where we extend the logical $\overline{Z}$ operators of the quantum code. The $\overline{X}$ case follows suit upon reversing the direction of interpretation of the product complex. The tensor product complex takes the form

$$\begin{tikzcd}
 & (S_Z,S) & & & \tilde{R}_Z \\
(S_Z,B) \arrow[ur, "\mathbb{1}\otimes h"] & & (Q,S) \arrow[ul, "H_Z\otimes\mathbb{1}"'] & & \tilde{S}_Z \arrow[u, "\tilde{M}_Z"'] \\
(Q,B) \arrow[u, "H_Z\otimes\mathbb{1}"] \arrow[urr, "\mathbb{1}\otimes h"] & & (S_X,S) \arrow[u, "H_X^\mathsf{T}\otimes\mathbb{1}"'] & & \tilde{Q} \arrow[u, "\tilde{H}_Z"'] \\
 & (S_X,B) \arrow[ul, "H_X^\mathsf{T}\otimes\mathbb{1}"] \arrow[ur, "\mathbb{1}\otimes h"'] & & & \tilde{S}_X \arrow[u, "\tilde{H}_X^\mathsf{T}"']
\end{tikzcd} \tag{17}$$

and the new CSS parity-check matrices are given by

$$\tilde{H}_X = \left(H_X \otimes \mathbb{1} \,\middle|\, \mathbb{1} \otimes h^\intercal \right) \tag{18a}$$

$$\tilde{H}_Z = \left(\begin{array}{c|c} H_Z \otimes \mathbb{1} & \mathbf{0} \\ \mathbb{1} \otimes h & H_X^\intercal \otimes \mathbb{1} \end{array} \right) . \tag{18b}$$

Geometrically, the Tanner graph of the homological product code resembles that of the Euclidean graph product between the two input Tanner graphs but with nodes relabeled according to (17); see Figure 2 for an illustration. When the input parity-check matrices have full rank, the parameters for the homological product code can be computed through the homology groups [ZP19]:

$$\tilde{n} = n_{\mathrm{Q}} n_{\mathrm{c}} + m_Z m_{\mathrm{c}} \tag{19a}$$

$$\tilde{k} = k_{\mathrm{Q}} k_{\mathrm{c}} \tag{19b}$$

$$\tilde{d}_X = d_X d_{\mathrm{c}} \tag{19c}$$

$$\tilde{d}_Z = d_Z \,. \tag{19d}$$

Note that in Eq. 19b, we have used the Künneth formula stated in Proposition 2.7 to compute $\tilde{k}$. Let us denote the (co)homology groups of the quantum code and classical code with $\mathcal{H}[\mathrm{Q}]$ and $\mathcal{H}[\mathrm{c}]$ respectively. Recall that we have adopted the convention for the logical $\overline{X}$ operator representatives of the CSS code to be given by the cohomology group $\mathcal{H}^1[\mathrm{Q}]$. By allowing the bits and checks of the classical code to be assigned to the 0-cells and 1-cells of the associated 2-term chain complex, we can observe that the codespace of the classical code is given by $\mathcal{H}^0[\mathrm{c}]$. Thus, a straightforward application of the Künneth formula tells us that the resulting product complex has the following cohomology group:

$$\tilde{\mathcal{H}}^1 \cong \mathcal{H}^1[\mathrm{Q}] \otimes \mathcal{H}^0[\mathrm{c}].$$

Since the rank of $\tilde{\mathcal{H}}^1$ gives us the number of logical qubits in the homological product code, we can simply sum the product of the ranks of the cohomology groups of the constituent codes to give us Eq. 19b.

From (18), we see that, similar to the HGP case, the qubits partition into a left and a right sector. Since the input parity-check matrices are full rank, we only have left-sector logical qubits. For these logical qubits, we can write a canonical basis for logical $\overline{X}$ and $\overline{Z}$ operators:

$$\tilde{G}_{Z,\mathrm{L}} = \left(G_Z \otimes \{\mathbf{e}_i\} \,\middle|\, \mathbf{0} \right) \tag{20a}$$

$$\tilde{G}_{X,\mathrm{L}} = \left(G_X \otimes g \,\middle|\, \mathbf{0} \right) , \tag{20b}$$

where G_X and G_Z are a symplectic logical basis for the input quantum code and $\{\mathbf{e}_i\} \notin \mathrm{rs}(h)$. If the input quantum CSS code is a HGP code, then we can designate G_Z and G_X as the canonical logical basis (16) as well and index all logical qubits with 3-dimensional coordinate triples (i, j, k).

We point out that our construction of the homological product code described above has implicitly assumed that the quantum CSS code corresponds to a 3-term chain complex and that the parity-check matrix h of the classical code has full rank. However, in general, it is known that we are able to associate a quantum CSS code to *any* 3 consecutive terms in a chain complex, which in its entirety could consist of more than 3 terms; the same goes for the classical code but with 2 terms. Notable examples include some of the quantum CSS codes that are amenable to single-shot error correction constructed by Campbell [Cam19]. These CSS codes are constructed from the middle 3 terms in a 5-term chain complex as shown below:

$$\mathcal{C} = \left(R_Z \xrightarrow{\partial_3 = M_Z^\intercal} S_Z \xrightarrow{\partial_2 = H_Z^\intercal} Q \xrightarrow{\partial_1 = H_X} S_X \xrightarrow{\partial_0 = M_X} R_X \right)$$

where we have associated the constitutent vector spaces $C_3, C_2, C_1, C_0, C_{-1}$ with R_Z, S_Z, Q, S_X, R_X i.e., the space of relations between Z checks, the Z checks, qubits, X checks, and the relations between X checks. A significant advantage of longer chain complexes is the appearance of "metacheck" matrices M_Z and M_X. These metacheck matrices can be interpreted as additional "codes" on the error syndromes, which can be used to detect and correct syndrome measurement errors. In this case, when we take the homological product between such CSS codes with a classical code, we end up with a 6-term chain complex. When we apply Proposition 2.7 to evaluate $\tilde{\mathcal{H}}^1$ for the new product complex, we now observe contributions from summands other than $\mathcal{H}^1[\mathrm{Q}] \otimes \mathcal{H}^0[\mathrm{c}]$ that include

$$\mathcal{H}^0[\mathrm{Q}] \otimes \mathcal{H}^1[\mathrm{c}] = \frac{\ker \delta_0[\mathrm{Q}]}{\operatorname{im} \delta_{-1}[\mathrm{Q}]} \otimes \frac{\ker \delta_1[\mathrm{c}]}{\operatorname{im} \delta_0[\mathrm{c}]} = \frac{\ker H_X^\mathsf{T}}{\operatorname{im} M_X^\mathsf{T}} \otimes \frac{\ker h^\mathsf{T}}{\operatorname{im} m^\mathsf{T}} \,. \tag{21}$$

Formerly, M_X^T did not exist when we only had a 3-term chain complex and $\operatorname{im} m^\mathsf{T}$ was trivial when the classical code had no redundant checks. The additional contribution from the other summands implies that we have additional logical qubits that arise from these additional cohomology group elements that we call "spurious" cohomologies. Because the structure of these spurious cohomologies depends on the redundancy of checks in both the quantum and classical codes, the cosystolic distances for these spurious cohomologies can vary. By choosing a canonical basis for these spurious cohomologies as well as the cohomologies that we hope to keep, i.e., $\mathcal{H}^1[Q] \otimes \mathcal{H}^0[c]$, we can show that we can treat the spurious cohomologies as gauge logical qubits that do not deprecate the distance of our resulting quantum code even when they dress our other logical operators. The intuition is similar to the case with the HGP: the gauge operators corresponding to the spurious cohomologies are supported in a different subsystem of the data qubits compared to the logical operators. We provide a more detailed analysis for this phenomenon in Lemma 5.16 in Section 5.

2.7.2 Homological product: quantum × quantum

Suppose we are given two quantum CSS codes with parameters that are denoted as $[\![n_\mathrm{Q}, k_\mathrm{Q}, (d_X, d_Z)]\!]$ and $[\![n'_\mathrm{Q}, k'_\mathrm{Q}, (d'_X, d'_Z)]\!]$ and corresponding parity-check matrices H_X, H_Z, H'_X, H'_Z. Similar to the case for the homological product between a quantum CSS code and a classical linear code, we can take the Kronecker product of their underlying chain complexes and associate the product complex with a (quantum × quantum) *homological product code* [FH14, BH14]. Because each quantum CSS code is associated to a 3-term chain complex (from our simplifying assumption discussed in the previous section), the product complex now becomes a 5-term chain complex. The 5-term chain gives us a some flexibility to choose which set of three consecutive vector spaces to place the X-syndromes, qubits, and Z-syndromes. For the purpose of reducing the number of cases that we have to analyze, we assume that the two 3-term chain complexes associated to the two quantum CSS codes have the following form:

$$\mathcal{C} = \left(S_Z \xrightarrow{\partial_1 = H_Z^\mathsf{T}} Q \xrightarrow{\partial_0 = H_X} S_X \right),$$

where C_1, C_0, C_{-1} are the vector spaces for the Z checks, qubits, and X checks respectively. After taking the homological product of the 3-term chain complexes, we let the new Z checks, qubits, and X checks be associated to the vector spaces $\tilde{C}_1, \tilde{C}_0, \tilde{C}_{-1}$ of the product complex $\tilde{\mathcal{C}}$. The product complex takes the form

$$\begin{tikzcd}
 & & (S_Z, S'_Z) & & & & \tilde{R}_Z \\
(S_Z, Q') \arrow[urr, "\mathbb{1}\otimes H'_Z"] & & & & (Q, S'_Z) \arrow[ull, "H_Z\otimes\mathbb{1}"'] & & \tilde{S}_Z \arrow[u, "\tilde{M}_Z"] \\
(S_Z, S'_X) \arrow[u, "\mathbb{1}\otimes H'^{\intercal}_X"] & & (Q, Q') \arrow[ull, "H_Z\otimes\mathbb{1}"'] \arrow[urr, "\mathbb{1}\otimes H'_Z"] & & (S_X, S'_Z) \arrow[u, "H^{\intercal}_X\otimes\mathbb{1}"'] & & \tilde{Q} \arrow[u, "\tilde{H}_Z"] \\
(Q, S'_X) \arrow[u, "H_Z\otimes\mathbb{1}"] \arrow[urr, "\mathbb{1}\otimes H'^{\intercal}_X"'] & & & & (S_X, Q') \arrow[ull, "H^{\intercal}_X\otimes\mathbb{1}"] \arrow[u, "\mathbb{1}\otimes H'_Z"'] & & \tilde{S}_X \arrow[u, "\tilde{H}^{\intercal}_X"] \\
 & & (S_X, S'_X) \arrow[ull, "H^{\intercal}_X\otimes\mathbb{1}"] \arrow[urr, "\mathbb{1}\otimes H'^{\intercal}_X"'] & & & & \tilde{R}_X \arrow[u, "\tilde{M}^{\intercal}_X"]
\end{tikzcd} \tag{22}$$

and the new CSS parity-check matrices are

$$\tilde{H}_X = \left(\begin{array}{c|c|c} H^{\intercal}_Z \otimes \mathbb{1} & \mathbb{1} \otimes H'_X & \mathbf{0} \\ \mathbf{0} & H_X \otimes \mathbb{1} & \mathbb{1} \otimes H'^{\intercal}_Z \end{array}\right), \tag{23a}$$

$$\tilde{H}_Z = \left(\begin{array}{c|c|c} \mathbb{1} \otimes H'^{\intercal}_X & H_Z \otimes \mathbb{1} & \mathbf{0} \\ \mathbf{0} & \mathbb{1} \otimes H'_Z & H^{\intercal}_X \otimes \mathbb{1} \end{array}\right), \tag{23b}$$

where the three partitions in the parity-check matrices correspond to the physical qubits in the subspaces $(S_Z, S'_X), (Q, Q'), (S_X, S'_Z)$ respectively; see Figure 3 for an illustration. We refer to these subsets of physical qubits as left (L), middle (M), and right (R) respectively. Focusing on the middle logical qubits, we can write a canonical basis for logical $\overline{X}$ and $\overline{Z}$ operators:

$$\tilde{G}_{Z,\mathrm{M}} = \left(\,\mathbf{0} \mid G_Z \otimes G'_Z \mid \mathbf{0}\,\right), \tag{24a}$$

$$\tilde{G}_{X,\mathrm{M}} = \left(\,\mathbf{0} \mid G_X \otimes G'_X \mid \mathbf{0}\,\right). \tag{24b}$$

One can straightforwardly verify that this basis is symplectic when the input bases are symplectic. We defer the discussion of the exact code parameters for the homological product code to Section 5.3.1 because the code distances for such codes are not sufficiently well understood in the most general case.

3 Generic bounds and limitations

In this section, we review some limitations of automorphism gates and then derive some broad results that will guide us in tailoring classical codes to our formalism later on in Section 6. We note some of these results are likely "folklore" within the error-correction community, but we state them explicitly for completeness.

3.1 Limitations on automorphisms of classical linear codes

Recall the classical automorphism conditions:

$$H\sigma = WH \tag{25a}$$

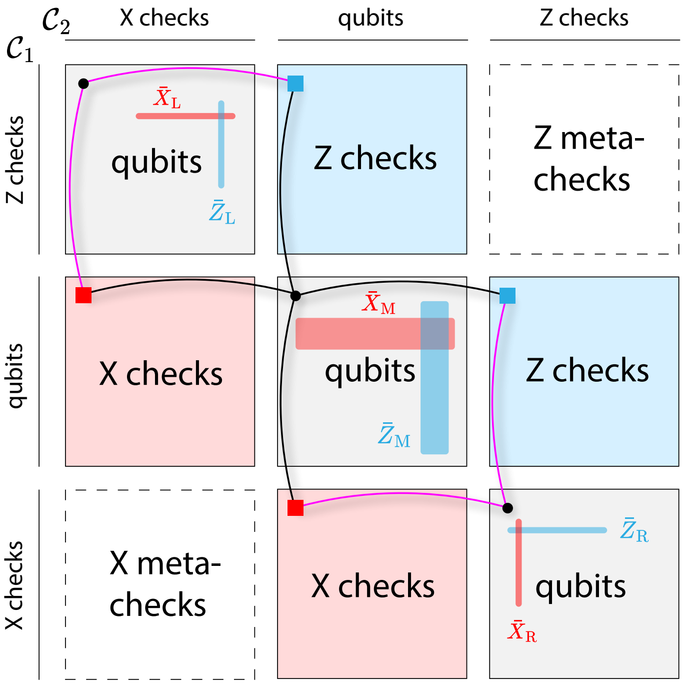

Figure 3: The layout of a (quantum × quantum) homological product code is depicted. New qubit-check connections are colored in magenta.

$$G\sigma = VG \tag{25b}$$

where H is the classical parity check matrix, G is the generator matrix, and σ is a permutation of the physical bits. Also recall that the affine class refers to functions that represent affine transformations over $\mathbb{F}_2$, any of which can be expressed using only the XOR logical gate and constants. From this definition, we can see that the CNOT gate generates all affine transformations [AGS15].

Theorem 3.1 (Logical action restricted to affine class)**.** *Suppose we have a linear code $\mathcal{C}$. The logical action of any element of* Aut($\mathcal{C}$) *is limited to the affine class.*

Proof. Let $G \in \mathbb{F}_2^{k\times n}$ be a generator matrix for $\mathcal{C}$, and let $U \in \mathbb{F}_2^{n\times n}$ be a permutation matrix corresponding to some element in Aut($\mathcal{C}$). Then the definition of Aut($\mathcal{C}$) requires $GU = WG$ for some invertible $W \in \mathbb{F}_2^{k\times k}$. Since W is invertible, its logical action is reversible. Furthermore, W can be decomposed into elementary row operations consisting of row addition and swaps. Row addition is equivalent to logical CNOT gates, with control and target defined by which row is replaced. Row swapping is equivalent to logical SWAPs, which can be decomposed into three CNOT gates. Thus, since W can be fully decomposed into logical CNOT gates, its logical action falls into the affine class of classical reversible gates. □

An immediate corollary is that we cannot achieve a universal set of logical reversible gates on linear codes from physical permutations alone, since the affine class is nonuniversal [AGS15].

Corollary 3.2 (Classical automorphism gates are nonuniversal)**.** *For any linear code $\mathcal{C}$, the group of logical automorphism gates $\mathcal{A}(\mathcal{C})$ cannot form a universal set.*

Another consequence of Theorem 3.1 is that the number of distinct logical operations achieved by physical permutations is upper bounded by the number of invertible matrices W, which is the size of the general linear group of degree k over $\mathbb{F}_2$, $\mathrm{GL}_k(\mathbb{F}_2)$. Note that the determinant map is trivial over $\mathbb{F}_2$, and so we also have $\mathrm{GL}_k(\mathbb{F}_2) \simeq \mathrm{SL}_k(\mathbb{F}_2) \simeq \mathrm{PSL}_k(\mathbb{F}_2)$ since the only scalars are 0 and 1 in $\mathbb{F}_2$.

Corollary 3.3 (Upper bound on number of logical gates)**.** *For any linear code $\mathcal{C}$ encoding k logical bits, we have $|\mathcal{A}| \leq |\mathrm{GL}_k(\mathbb{F}_2)|$.*

Suppose we desire a linear code with $|\mathcal{A}| \propto |\mathrm{GL}_k(\mathbb{F}_2)|$, i.e. approaching the upper bound of Corollary 3.3. Since the total number of distinct permutations of n bits is $|\mathrm{S}_n| = n!$, for a given code dimension k, we can derive a lower bound on the minimum code length n, or alternatively an upper bound on k for given n, by comparing to the size of $\mathrm{GL}_k(\mathbb{F}_2)$.

Corollary 3.4. *For any linear code with $|\mathcal{A}| \propto |\mathrm{GL}_k(\mathbb{F}_2)|$, we must have $k = O\big(\sqrt{n \log n}\big)$.*

Proof. Every $W \in \mathrm{im}(\phi) \simeq \mathcal{A}$ needs to have a distinct preimage in $\mathrm{Aut}(\mathcal{C})$ for ϕ to be a valid homomorphism. Thus, we have $|\mathcal{A}| = |\mathrm{Aut}(\mathcal{C})/\ker(\phi)| \leq |\mathrm{Aut}(\mathcal{C})| \leq |\mathrm{S}_n|$ with the first inequality saturated if and only if ϕ is injective, i.e. $\ker(\phi) = \mathbb{1}$. The asymptotic size of S_n is $n! = 2^{O(n \log n)}$ by Stirling's formula, and the asymptotic size of $\mathrm{GL}_k(\mathbb{F}_2)$ is $\Theta\big(2^{k^2}\big)$. Equating the two group sizes, we arrive at the statement of the Corollary. □

We note that Corollary 3.4 is likely loose, since it has been shown that all equidistant linear codes are sequences of dual-Hamming codes [Bon84], which have $k = O(\log n)$. The equidistant property is a necessary condition for $\mathcal{A} = \mathrm{GL}_k(\mathbb{F}_2)$ since permutations preserve the weight of codewords.

When the code distance is strictly greater than 2, we can show that $\mathcal{A}(\mathcal{C})$ is contained in $\mathcal{A}(\mathcal{C}^\perp)$, which we will denote $\mathcal{A}^\perp$ for short.

Theorem 3.5 (Dual automorphism bound)**.** *For any linear code $\mathcal{C}$ with code distance $d \geq 3$, we have $\mathcal{A} \subseteq \mathcal{A}^\perp \simeq \mathrm{Aut}(\mathcal{C}^\perp)$.*

Proof. Recall that for any linear code, we have $\mathrm{Aut}(\mathcal{C}) \simeq \mathrm{Aut}(\mathcal{C}^\perp) = \mathrm{Aut}(h)$ by default. So in order to show that $\mathcal{A}$ is contained in $\mathcal{A}^\perp$, we essentially need to show that there cannot exist two distinct physical permutations σ, σ' that correspond to the same row-transformation w on h. We will prove this statement by contradiction. Suppose we have two distinct permutations $\sigma, \sigma' \in \mathrm{S}_n$ with $h\sigma = h\sigma' = wh$. Define the permutation matrix $\sigma'' = \sigma' \sigma^{\mathsf{T}} \neq \mathbb{1}$. Then we have $h\sigma'' = w^{-1}wh = h$, and upon rearranging, $h(\mathbb{1} + \sigma'') = 0$, which implies that the columns of $\mathbb{1} + \sigma''$ are in $\ker h = \mathcal{C}$. However, the weight of any column in $\mathbb{1} + \sigma''$ is at most 2, which contradicts our assumption that the code distance $d \geq 2$. Thus, distinct $\sigma, \sigma' \in \mathrm{Aut}(\mathcal{C}^\perp)$ must correspond to distinct $w, w' \in \mathcal{A}^\perp$, which shows that $\mathrm{Aut}(\mathcal{C}^\perp) \subseteq \mathcal{A}^\perp$. Since $\mathcal{A}^\perp \subseteq \mathrm{Aut}(\mathcal{C}^\perp)$ by definition, we conclude that $\mathcal{A}^\perp \simeq \mathrm{Aut}(\mathcal{C}^\perp)$, i.e. $\ker \phi^\perp$ is trivial. Combining this last statement with the fact that $\mathcal{A} \subseteq \mathrm{Aut}(\mathcal{C}) \simeq \mathrm{Aut}(\mathcal{C}^\perp)$, we arrive at the statement of the Theorem. □

By swapping the roles of $\mathcal{C}$ and $\mathcal{C}^\perp$ in Theorem 3.5, we can achieve a strict equality if the dual distance also satisfies $d^\perp \geq 3$. In other words, both $\ker \phi$ and $\ker \phi^\perp$ are trivial.

Corollary 3.6 (Automorphism equivalence)**.** *If both the minimum distance and dual distance are greater than 2, then $\mathrm{Aut}(\mathcal{C}) \simeq \mathcal{A}(\mathcal{C}) \simeq \mathcal{A}(\mathcal{C}^\perp) \simeq \mathrm{Aut}(\mathcal{C}^\perp)$.*

As we will see later in the examples (Section 6), it will sometimes be easier to characterize $\mathcal{A}^\perp$ or $\mathrm{Aut}(\mathcal{C}^\perp)$ than $\mathcal{A}(\mathcal{C})$ itself, which provides us with a convenient way to compute $\mathcal{A}(\mathcal{C})$ when both $d, d^\perp \geq 3$.

3.2 Limitations on automorphisms of simply connected graphs

In Section 2.2, we briefly discussed a subgroup of code automorphisms known as Tanner graph automorphisms, which will be useful for our automorphism gadgets later on in Section 4. In this section, we review a well-known limitation for the automorphisms of simple, connected graphs before developing a bound for the size of the automorphism group of a popular family of simple, connected graphs. These bounds will constrain the automorphism groups of codes whose Tanner graphs are isomorphic to simple, connected graphs. We only work with finite, simple, connected graphs that have vertex degree at least 2.

We begin by stating some important definitions regarding path automorphisms, s-transitive graphs and s-regular graphs.

Definition 3.7 (Path automorphism). *Let paths P and Q of some graph G be defined as two ordered sets:*

$$P = \left\{v_{p_0}, e_{p_1}, v_{p_1}, \dots, e_{p_n}, v_{p_{n+1}}\right\},$$

$$Q = \left\{v_{q_0}, e_{q_1}, v_{w_1}, \dots, e_{q_n}, v_{q_{n+1}}\right\},$$

where $v_{p_i}, v_{q_j} \in V(\mathsf{G})$ and $e_{p_i}, e_{q_j} \in E(\mathsf{G})$. Then, an automorphism of G such that $f(P) = Q$ is defined such that f maps the i^{th} term of P onto the i^{th} term of Q, for each index i.

Definition 3.8 (s-transitive graph [Tut66, Section 7.5]). *A graph G is s-transitive, for a given positive integer s, if it has at least one path of length s and if for each ordered pair (P, Q) of paths of length s of G, there exists an automorphism f of G such that $f(P) = Q$.*

Note that when $s = 0$, the graph G is vertex-transitive i.e., there exists some graph automorphism f such that $f(u) = v$ for any pair of vertices $u, v \in V(\mathsf{G})$. In addition, it is known that an s-transitive graph is also a $(s-1)$-transitive graph for $s \geq 1$ as long as the graph has vertex degree at least 2.

Definition 3.9 (s-regular graph [Tut66, Section 7.7]). *A graph G is s-regular if it is s-transitive and for each ordered pair (P, Q) of paths of length s, there is only one automorphism f of G such that $f(P) = Q$.*

We emphasize that this is different from the conventional definition of regularity in graphs, where k-regular means that each vertex has the same degree k. Here, regularity is a statement on the number of automorphisms per a pair of paths (P, Q).

We are particularly interested in cubic graphs where each vertex has degree 3 because we are interested in the regime where the base code has weight-3 checks so that the quantum code will similarly have small constant-size weight checks when taking products of such codes. In addition, cubic graphs are very well-studied and a lot of work has been done to understand their automorphism groups. The following theorems are useful in limiting the girth and the size of the automorphism group for simply connected graphs with the above properties.

Theorem 3.10 (Girth bound by regularity [Tut47, Theorem III]). *Let G be a s-regular cubic graph. Then, the girth g of G is bounded by the following:*

$$g \geq 2s - 2.$$

Recall that the girth of the graph is related to the distance of the classical code associated to the graph. This implies that the distance of the code is lower bounded by the regularity of its underlying graph.

Theorem 3.11 (Upper bound on transitivity [Big74, Theorem 18.6])**.** *If a graph* G *is simply connected, cubic, and* t*-transitive, then* $t \leq 5$*.*

Theorem 3.12 (Size of automorphism group of s-regular cubic graphs [Tut47, Theorem VI])**.** *Let* G *be a* s*-regular cubic graph with* n *vertices. Then,*

$$|\mathrm{Aut}(\mathsf{G})| = 2^{s-1} \cdot 3n.$$

Given the above two theorems, we now provide a useful theorem for upper bounding the size of the automorphism group of a simple, connected, cubic, s-regular, and t-transitive graph with n vertices. We emphasize that the bound is independent of s and t.

Theorem 3.13 (Upper bound on size of automorphism group of simply connected cubic graphs)**.** *Let* G *be a simply connected, cubic,* s*-regular,* t*-transitive graph with* n *vertices. Then,*

$$|\mathrm{Aut}(\mathsf{G})| \leq 48n.$$

Proof. Thm. 3.11 tells us that the simply connected, cubic, regular graph G can be at most 5-transitive, implying that G can be at most 5-regular by Def. 3.9. From a straightforward application of Thm. 3.12, we obtain the theorem statement. □

Based on Thm. 3.13, we see that the size of the automorphism group of a code with an underlying graph that is cubic, simply connected, s-transitive, and t-transitive can at most scale linearly with the number of checks. This implies that codes constructed from such graphs are not great candidates in the asymptotic limit if we are interested in having a large number of automorphisms.

3.3 Limitations on automorphisms of quantum CSS codes

For a classical linear code, an automorphism of its codespace automatically implies an automorphism of its dual. For quantum stabilizer codes, logical Pauli operators related by an element of the stabilizer act equivalently on the codespace, and so distinct logical operations are grouped into equivalence classes under the code's stabilizer group. As a consequence, for a given choice of logical basis operators, there is not a unique "generator matrix" and so the classical automorphism condition (1), which is defined with respect to a generator matrix, is not a useful test for automorphism. What is useful, on the other hand, is the dual automorphism condition (3), which naturally takes into account stabilizer degeneracy. There are some additional subtleties related to notions of strong and weak automorphisms for which we refer the reader to [Hao21] for more details. In this work, we focus on CSS codes, whose analyses is much simpler and does not contain such subtleties. Since physical permutations only map Pauli strings to other Pauli strings, it is clear that the logical automorphism group for any quantum stabilizer code is strictly contained in the Clifford group. Furthermore, since permutations cannot change the Pauli type, for a CSS code the logical automorphism group is contained within the affine group, and so Corollary 3.3 naturally applies as well.

A recent work by Guyot and Jaques provides generic upper bounds and limitations on the number of distinct automorphism gates $|\mathcal{A}|$ for any CSS code [GJ25]. When we use a CSS code as input to the homological product, the number of inherited automorphism gadgets will be constrained by their result. We quote their relevant results here and encourage the reader to explore their paper for the technical details and proofs.

Theorem 3.14 (Upper bound on distinct automorphism gates for CSS codes; Theorem 2 of [GJ25])**.** *Let $C = \mathrm{CSS}(A, B)$ be an $[\![n, k, d]\!]$ code with $d \geq 3$. The number of distinct logical operations that can be implemented by permuting qubits in the code is upper-bounded by $|\mathcal{A}| \leq \frac{n!}{k_{\max}!}$, where $k_{\max} = \max(\dim(A), \dim(B))$.*

Corollary 3.15 (Corollary 5 of [GJ25])**.** *The following logical gates are not permutation addressable on CSS codes with the following rates:*

- *SWAP gates for codes with an asymptotic rate greater than $1/3$*
- *Any 2-qubit gate for codes with with an asymptotic rate greater than $3/4$*
- *CNOT gates for codes with an asymptotic rate of $\Omega\left(\sqrt{\frac{\log n}{n}}\right)$*

Since all the product constructions that we use produce CSS codes, they will all be constrained by these upper bounds. However, we will see that this generic upper bound can be loose for the product constructions, which demand additional constraints on the input codes for there to be nontrivial code automorphisms.

4 Automorphism gadgets

In this section, we derive our automorphism gadgets for all the homological product constructions mentioned in Section 2.6 and 2.7. We will determine both the physical actions on the data qubits as well as the encoded actions on the logical qubits. Note that for CSS codes, we only need to analyze the X-sector because commutation conditions will completely determine the corresponding transformation on the Z-sector. Specifically, for an invertible matrix $U \in \mathrm{GL}_n(\mathbb{F}_2)$ whose columns tabulate the transformation of all n X-type Paulis, $(U^{-1})^{\mathsf{T}} = (U^{\mathsf{T}})^{-1} \equiv U^{-\mathsf{T}}$ determines the corresponding action on the n Z-type Paulis. This is because any Clifford transformation should preserve the symplectic norm $\mathit{\Omega} = \begin{pmatrix} \mathbf{0} & \mathbb{1}_n \\ \mathbb{1}_n & \mathbf{0} \end{pmatrix}$ i.e. $A\mathit{\Omega}A^{\mathsf{T}} = \mathit{\Omega}$, where A is a $2n \times 2n$ matrix with U and $U^{-\mathsf{T}}$ on the block diagonal. Preserving the symplectic norm $\mathit{\Omega}$ is equivalent to preserving the Pauli algebra on the qubits.

4.1 Hypergraph product codes

We begin by reviewing the construction in ([HMKL24], Appendix D), which introduced the concept of logical gate inheritance for hypergraph product codes. At a high level, the idea is to leverage the tensor product structure of the HGP to derive logical gadgets stemming from those of the classical input codes. For our purposes, we will focus on the gadgets corresponding to automorphisms of the classical input codes. Denoting $\oplus$ as the direct sum of matrices (as opposed to Kronecker sum), we define the unitary operators

$$U_1 \equiv (\sigma_1 \otimes \mathbb{1}) \oplus (w_1 \otimes \mathbb{1}) = \begin{pmatrix} \sigma_1 \otimes \mathbb{1} & \mathbf{0} \\ \mathbf{0} & w_1 \otimes \mathbb{1} \end{pmatrix}, \tag{26a}$$

$$U_2 \equiv (\mathbb{1} \otimes \sigma_2) \oplus (\mathbb{1} \otimes w_2^{-\mathsf{T}}) = \begin{pmatrix} \mathbb{1} \otimes \sigma_2 & \mathbf{0} \\ \mathbf{0} & \mathbb{1} \otimes w_2^{-\mathsf{T}} \end{pmatrix}. \tag{26b}$$

We can interpret U_1 as acting $\sigma_1 \otimes \mathbb{1}$ on the left qubits and $w_1 \otimes \mathbb{1}$ on the right qubits where σ_1 is some permutation that satisfies $h_1\sigma_1 = w_1 h_1$. Likewise, σ_2 is chosen to be some permutation for

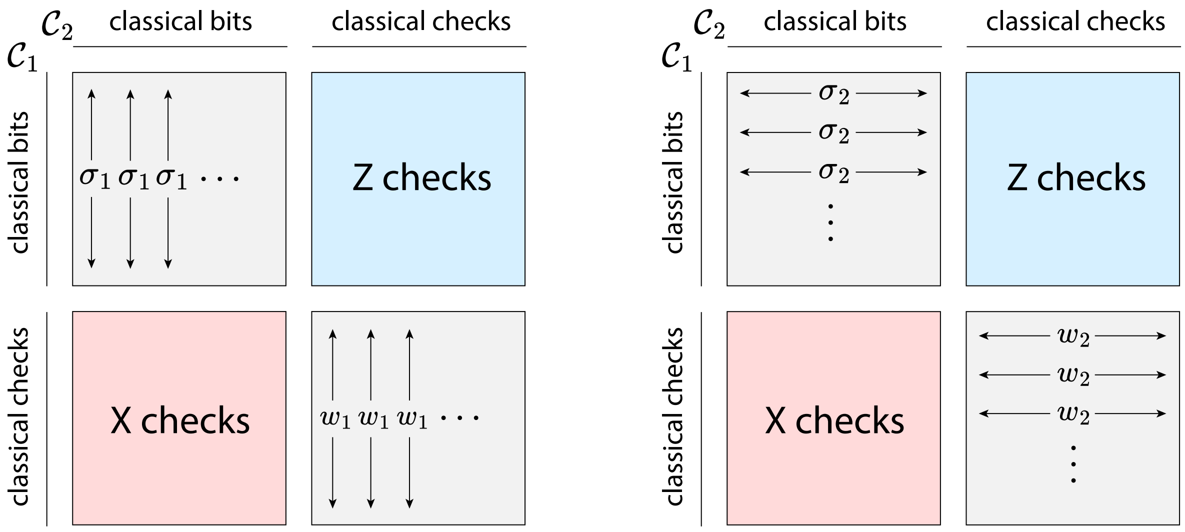

Figure 4: The physical implementations of the HGP automorphism gadgets (26) are depicted. An automorphism of the first input code $\mathcal{C}_1$ results in a gadget acting within columns of the HGP code, and an automorphism of the second input code $\mathcal{C}_2$ results in a gadget acting within the rows.

U_2 such that $h_2\sigma_2 = w_2h_2$. Interpreting the tensor product as (rows $\otimes$ columns), we see that U_1 acts the permutation σ_1 on all left rows and w_1 on all right rows of the HGP code in parallel. U_2 follows suit but on the columns; see Figure 4 for an illustration. To check that U_1 preserves the stabilizer group of the HGP code, it suffices to examine its actions on H_X and H_Z. For H_X we have

$$\begin{aligned} H_XU_1 &= \big(h_1\sigma\otimes\mathbb{1}\mid w_1\otimes h_2^{\top}\big)\\ &= \big(w_1h_1\otimes\mathbb{1}\mid w_1\otimes h_2^{\top}\big)\\ &= (w_1\otimes\mathbb{1})\big(h_1\otimes\mathbb{1}\mid\mathbb{1}\otimes h_2^{\top}\big)\\ &= W_1H_X\,, \end{aligned}\tag{27}$$

where $W_1\equiv w_1\otimes\mathbb{1}$. The action on Z-type Pauli operators is given by $U_1^{-\top}$, and so for H_Z we have

$$\begin{aligned} H_ZU_1^{-\top} &= \big(\sigma_1\otimes h_2\mid h_1^{\top}w_1^{-\top}\otimes\mathbb{1}\big)\\ &= \big(\sigma_1\otimes h_2\mid (w_1^{-1}h_1)^{\top}\otimes\mathbb{1}\big)\\ &= \big(\sigma_1\otimes h_2\mid (h_1\sigma_1^{-1})^{\top}\otimes\mathbb{1}\big)\\ &= \big(\sigma_1\otimes h_2\mid \sigma_1h_1^{\top}\otimes\mathbb{1}\big)\\ &= (\sigma_1\otimes\mathbb{1})\big(\mathbb{1}\otimes h_2\mid h_1^{\top}\otimes\mathbb{1}\big)\\ &= W_1'H_Z\,, \end{aligned}\tag{28}$$

where $W_1'\equiv\sigma_1\otimes\mathbb{1}$, and in the first and fourth line we have used the fact that $\sigma_1^{-1}=\sigma_1^{\top}$ for a permutation matrix. (27) and (28) show that the row spaces of H_X and H_Z remain invariant under U_1 (26a), and thus the stabilizer group is preserved. The proof for U_2 follows suit upon swapping the sides of the tensor product.

The logical action of U_1 (26a) can be obtained by examining its action on our canonical logical basis (16). For now, we focus on the action on the left logical qubits. The logical $\overline{Z}$ operators (16a) transform according to

$$\begin{aligned} G_Z U_1^{-\mathsf{T}} &= \big(g_1\sigma_1 \otimes \{\mathbf{e}_j\} \;\big|\; \mathbf{0}\big) \\ &= \big(v_1 g_1 \otimes \{\mathbf{e}_j\} \;\big|\; \mathbf{0}\big) \\ &= (v_1 \otimes \mathbb{1})\, G_Z \\ &= V_1' G_Z\,. \end{aligned} \tag{29}$$

Hence, we see that within each canonical column of logical qubits, the logical $\overline{Z}$ operators transform the same way as the classical codewords, according to v. The action on the logical $\overline{X}$ operators is

$$G_X U_1 = \big(\{\mathbf{e}_i\}\sigma_1 \otimes g_2 \;\big|\; \mathbf{0}\big) = \big(\{\mathbf{e}_{\sigma_1(i)}\} \otimes g_2 \;\big|\; \mathbf{0}\big) \equiv G_X'\,. \tag{30}$$

It is not so clear from the above equation what the relation is between G_X' and G_X, but we emphasize that on the logical level it is completely determined by (29) due to commutation relations between $\overline{X}$s and $\overline{Z}$s. Denote $\overline{V} \in \mathbb{F}_2^{k^2 \times k^2}$ as the logical action of V after accounting for stabilizer degeneracy. The Pauli algebra (i.e. the symplectic condition) demands $\overline{V}_1'\overline{V}_1^{\mathsf{T}} = \mathbb{1}$, which fixes $\overline{V}_1' = \overline{V}_1^{-\mathsf{T}}$. The analysis for U_2 follows similarly but amongst the rows of logical qubits.

Overall, the logical $\overline{X}$ actions are given by

$$\overline{V}_1 = v_1^{-\mathsf{T}} \otimes \mathbb{1} \tag{31a}$$

$$\overline{V}_2 = \mathbb{1} \otimes v_2\,. \tag{31b}$$

Since $[\overline{V}_1, \overline{V}_2] = 0$, the group of inherited logical automorphism gates is $\mathcal{A}_1 \times \mathcal{A}_2$. We comment that although the group generated by (31) is basis-invariant, the particular structure of these logical gates on the logical qubits will depend on the presentation of the generator matrices of the classical input codes.

Note that $w \in \mathrm{GL}_k(\mathbb{F}_2)$ (25) is in general not a permutation matrix, and so our U_1 (26a) and U_2 (26b) do not necessarily belong to $\mathrm{Aut}\big(\mathrm{HGP}(\mathcal{C})\big)$. In the special instances where $w \in \mathrm{S}_k$ is a permutation matrix, then we do indeed have an exact automorphism of the HGP code; i.e. $W_1, W_1', W_2, W_2' \in \mathrm{S}_n$. These special instances are precisely the Tanner graph automorphisms of Def. 2.2. Nonetheless, in the general case, we can decompose an arbitrary $w \in \mathrm{GL}_k(\mathbb{F}_2)$ into elementary operations consisting of permutations and CNOT gates.

Theorem 4.1 (Number of inherited logical gates for HGP codes)**.** *Let* $\mathrm{HGP}(h_1, h_2)$ *be a HGP code, and let* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *be the automorphism groups of the classical input codes. The number of distinct automorphism gadgets, according to* (26) *and* (31)*, is* $|\mathcal{A}_1||\mathcal{A}_2|$.

Theorem 4.2 (Lower bound on automorphism group of HGP codes)**.** *Let* $\mathrm{HGP}(h_1, h_2)$ *be a HGP code, and let* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *be the Tanner graph automorphism groups of* h_1 *and* h_2 *respectively. Then we have* $\mathcal{T}_1 \times \mathcal{T}_2 \subset \mathrm{Aut}\big(\mathrm{HGP}(h_1, h_2)\big)$.

So far, we have focused on automorphism gadgets that act as physical permutations on the left sector of data qubits. A complementary pair of automorphism gadgets can be derived using the transpose of the input codes, which corresponds to physical permutations that satisfy the automorphism condition (3) on the transposed parity-check matrices. These "right-sector" automorphism gadgets generically act as a permutation on the right sector and a circuit on the left sector. For brevity, we will only focus on the left-sector automorphism gadgets, as we typically have $k_1 k_2 \gg k_1^{\mathsf{T}} k_2^{\mathsf{T}}$ in (15); however see Section 7.2 for an example where we use the right-sector automorphism gadgets. If we restrict to the subgroup of Tanner graph automorphisms in the input codes, then the left-sector and right-sector automorphism gadgets are isomorphic.

4.2 Homological product codes: quantum × classical

Having shown the expressions for automorphism gate inheritance in the hypergraph product, we now inductively derive analogous expressions for higher-fold homological products, comprised of a quantum CSS code and another classical code. The resulting homological product code can then be subsequently tensored again with another classical code to produce another homological product code and so forth.

Suppose the input CSS code possesses a logical gadget of the form $H_X U = W H_X$ and $H_Z U^{-\mathsf{T}} = W' H_Z$, and the input classical code possesses an automorphism of the form $h\sigma = wh$. Then their homological product code will inherit the logical gadgets given by the actions of

$$\tilde{U}_{\mathrm{c}} \equiv \big(\mathbb{1} \otimes \sigma\big) \oplus \big(\mathbb{1} \otimes w^{-\mathsf{T}}\big) \tag{32a}$$
$$\tilde{U}_{\mathrm{Q}} \equiv \big(U \otimes \mathbb{1}\big) \oplus \big(W \otimes \mathbb{1}\big) \tag{32b}$$

on the Pauli X operators, where the subscripts Q and c label the logical gadgets inherited from those of the input quantum and classical codes respectively. We will now show that the above physical transformations (32) preserve the stabilizer group given by (18). Starting with $\tilde{U}_{\mathrm{c}}$ (32a), its action on $\tilde{H}_X$ (18a) is

$$\begin{aligned}\tilde{H}_X \tilde{U}_{\mathrm{c}} &= \big(\, H_X \otimes \sigma \,\big|\, \mathbb{1} \otimes h^{\mathsf{T}} w^{-\mathsf{T}} \,\big) \\ &= \big(\, H_X \otimes \sigma \,\big|\, \mathbb{1} \otimes \sigma h^{\mathsf{T}} \,\big) \\ &= (\mathbb{1} \otimes \sigma)\, \tilde{H}_X \\ &= \tilde{W}_{\mathrm{c}} \tilde{H}_X \,,\end{aligned} \tag{33}$$

where in the last line we have defined $\tilde{W}_{\mathrm{c}} \equiv \mathbb{1} \otimes \sigma$. The action on $\tilde{H}_Z$ (18b) is

$$\begin{aligned}\tilde{H}_Z \tilde{U}_{\mathrm{c}}^{-\mathsf{T}} &= \left(\begin{array}{c|c} H_Z \otimes \sigma & \mathbf{0} \\ \mathbb{1} \otimes h\sigma & H_X^{\mathsf{T}} \otimes w \end{array}\right) \\ &= \left(\begin{array}{c|c} H_Z \otimes \sigma & \mathbf{0} \\ \mathbb{1} \otimes wh & H_X^{\mathsf{T}} \otimes w \end{array}\right) \\ &= (\mathbb{1} \otimes \sigma \mid \mathbb{1} \otimes w)\, \tilde{H}_Z \\ &= \tilde{W}'_{\mathrm{c}} \tilde{H}_Z \,,\end{aligned} \tag{34}$$

where in the last line we have defined $\tilde{W}'_{\mathrm{c}} \equiv (\mathbb{1} \otimes \sigma \mid \mathbb{1} \otimes w)$. The combination of (33) and (34) demonstrate that $\tilde{U}_{\mathrm{c}}$ (32a) preserves the codespace.

Now we move onto $\tilde{U}_{\mathrm{Q}}$ (32b). Its action on $\tilde{H}_X$ (18a) is

$$\begin{aligned}\tilde{H}_X \tilde{U}_{\mathrm{Q}} &= \big(\, H_X U \otimes \mathbb{1} \,\big|\, W \otimes h^{\mathsf{T}} \,\big) \\ &= \big(\, W H_X \otimes \mathbb{1} \,\big|\, W \otimes h^{\mathsf{T}} \,\big) \\ &= \big(W \otimes \mathbb{1}\big)\, \tilde{H}_X \\ &= \tilde{W}_{\mathrm{Q}} \tilde{H}_X\end{aligned} \tag{35}$$

where in the last line we have defined $\tilde{W}_{\mathrm{Q}} \equiv W \otimes \mathbb{1}$. The action on $\tilde{H}_Z$ (18b) is

$$\tilde{H}_Z \tilde{U}_{\mathrm{Q}}^{-\mathsf{T}} = \left(\begin{array}{c|c} H_Z U^{-\mathsf{T}} \otimes \mathbb{1} & \mathbf{0} \\ U^{-\mathsf{T}} \otimes h & H_X^{\mathsf{T}} W^{-\mathsf{T}} \otimes \mathbb{1} \end{array}\right)$$

$$
\begin{aligned}
&= \left(\begin{array}{c|c} W' H_Z \otimes \mathbb{1} & \mathbf{0} \\ U^{-\mathsf{T}} \otimes h & U^{-\mathsf{T}} H_X^{\mathsf{T}} \otimes \mathbb{1} \end{array} \right) \\
&= \left(W' \otimes \mathbb{1} \,\middle|\, U^{-\mathsf{T}} \otimes \mathbb{1} \right) \tilde{H}_Z \\
&= \tilde{W}'_{\mathrm{Q}} \tilde{H}_Z \,, \qquad (36)
\end{aligned}
$$

where in the last line we have defined $\tilde{W}'_{\mathrm{Q}} \equiv \left(W' \otimes \mathbb{1} \,\middle|\, U^{-\mathsf{T}} \otimes \mathbb{1} \right)$. The combination of (35) and (36) show that $\tilde{U}_{\mathrm{Q}}$ (32b) preserves the codespace.

Let us summarize the structure of all the left-action $\tilde{W}$s we have derived thus far:

$$\tilde{W}_{\mathrm{c}} = \mathbb{1} \otimes \sigma \qquad (37a)$$

$$\tilde{W}'_{\mathrm{c}} = (\mathbb{1} \otimes \sigma \mid \mathbb{1} \otimes w) \qquad (37b)$$

$$\tilde{W}_{\mathrm{Q}} = W \otimes \mathbb{1} \qquad (37c)$$

$$\tilde{W}'_{\mathrm{Q}} = \left(W' \otimes \mathbb{1} \,\middle|\, U^{-\mathsf{T}} \otimes \mathbb{1} \right) \qquad (37d)$$

Examining the above equations, we observe that $\tilde{W}_{\mathrm{c}}$ is a Kronecker product of permutation matrices and so is a permutation matrix itself. Additionally, if w, W, U are constant-depth circuits, then so are $\tilde{W}'_{\mathrm{c}}, \tilde{W}_{\mathrm{Q}}, \tilde{W}'_{\mathrm{Q}}$ since they all involve Kronecker products of the former with the identity operator. In particular, if our input quantum CSS code is a HGP code from the previous section, then we can use either $W = W_1 = w_0 \otimes \mathbb{1}$ or $W = W_2 = \mathbb{1} \otimes \sigma_0$ (subscript 0 to not be confused with the new w and σ defined above). If the input classical code has a low-depth implementation for some w, then the inherited automorphism gadget in the resulting homological product code can be implemented in the same depth as well. Furthermore, Tanner graph automorphisms of the input classical code can combine with automorphisms of the input quantum code to produce exact automorphisms of the resulting homological product code.

Analogous to the HGP scenario, we can derive the corresponding logical $\overline{X}$ actions by examining the action of (32) on our canonical logical basis (20):

$$\overline{\tilde{V}}_C = \mathbb{1} \otimes v \qquad (38a)$$

$$\overline{\tilde{V}}_Q = \overline{V} \otimes \mathbb{1} \,. \qquad (38b)$$

Since $\overline{\tilde{V}}_C$ and $\overline{\tilde{V}}_Q$ commute, the group of logical gates is a direct product of those of the input quantum and classical codes.

4.3 Homological product codes: quantum × quantum

We can just as easily extend the analysis to the case where we take the homological product of two CSS codes. As noted in (22), the resulting quantum code can be represented as a 3-term chain complex out of the larger 5-term chain complex. One can also obtain a 5-term chain complex through the repeated homological product, i.e. classical × classical × classical × classical; however, that construction does not yield all instances achievable by quantum × quantum. In particular, it may be the case that the two quantum codes are not themselves product codes.

Suppose we are given two quantum CSS codes $\mathcal{C}, \mathcal{C}'$ defined by parity check matrices H_X, H_Z and H'_X, H'_Z, respectively, which possess logical gadgets of the form

$$H_X U_1 = W_1 H_X \qquad (39a)$$

$$H_Z U_1^{-\mathsf{T}} = \hat{W}_1 H_Z \qquad (39b)$$

$$H'_X U_2 = W_2 H'_X \tag{39c}$$

$$H'_Z U_2^{-\intercal} = \hat{W}_2 H'_Z \tag{39d}$$

Then the quantum code resulting from the homological product of $\mathcal{C}$ and $\mathcal{C}'$ will inherit logical automorphism gadgets defined as

$$\tilde{U}_1 \equiv \big(\hat{W}_1^{-\intercal} \otimes \mathbb{1}\big) \oplus \big(U_1 \otimes \mathbb{1}\big) \oplus \big(W_1 \otimes \mathbb{1}\big) \tag{40a}$$

$$\tilde{U}_2 \equiv \big(\mathbb{1} \otimes W_2\big) \oplus \big(\mathbb{1} \otimes U_2\big) \oplus \big(\mathbb{1} \otimes \hat{W}_2^{-\intercal}\big). \tag{40b}$$

Once again, we will show that these unitaries preserve the stabilizer group given by (23):

$$\begin{aligned}
\tilde{H}_X \tilde{U}_1 &= \left(\begin{array}{c|c|c} H_Z^\intercal \hat{W}_1^{-\intercal} \otimes \mathbb{1} & U_1 \otimes H'_X & \mathbf{0} \\ \mathbf{0} & H_X U_1 \otimes \mathbb{1} & W_1 \otimes H'^\intercal_Z \end{array} \right) \\
&= \left(\begin{array}{c|c|c} U_1 H_Z^\intercal \otimes \mathbb{1} & U_1 \otimes H'_X & \mathbf{0} \\ \mathbf{0} & W_1 H_X \otimes \mathbb{1} & W_1 \otimes H'^\intercal_Z \end{array} \right) \\
&= \big(U \otimes \mathbb{1} \,\big|\, W \otimes \mathbb{1} \big)\, \tilde{H}_X \\
&= \tilde{W}_1 \tilde{H}_X \,,
\end{aligned} \tag{41a}$$

$$\begin{aligned}
\tilde{H}_Z \tilde{U}_1^{-\intercal} &= \left(\begin{array}{c|c|c} \hat{W}_1 \otimes H'^\intercal_X & H_Z U_1^{-\intercal} \otimes \mathbb{1} & \mathbf{0} \\ \mathbf{0} & U_1^{-\intercal} \otimes H'_Z & H_X^\intercal W_1^{-\intercal} \otimes \mathbb{1} \end{array} \right) \\
&= \left(\begin{array}{c|c|c} \hat{W}_1 \otimes H'^\intercal_X & \hat{W}_1 H_Z \otimes \mathbb{1} & \mathbf{0} \\ \mathbf{0} & U_1^{-\intercal} \otimes H'_Z & U_1^{-\intercal} H_X^\intercal \otimes \mathbb{1} \end{array} \right) \\
&= \big(\hat{W}_1 \otimes \mathbb{1} \,\big|\, U_1^{-\intercal} \otimes \mathbb{1} \big)\, \tilde{H}_Z \\
&= \tilde{W}'_1 \tilde{H}_Z \,,
\end{aligned} \tag{41b}$$

The combination of (41a) and (41b) shows that $\tilde{U}_1$ (40a) preserves the codespace. We now look at $\tilde{U}_2$ (40b), the action of which yields

$$\begin{aligned}
\tilde{H}_X \tilde{U}_2 &= \left(\begin{array}{c|c|c} H_Z^\intercal \otimes W_2 & \mathbb{1} \otimes H'_X U_2 & \mathbf{0} \\ \mathbf{0} & H_X \otimes U_2 & \mathbb{1} \otimes H'^\intercal_Z \hat{W}_2^{-\intercal} \end{array} \right) \\
&= \left(\begin{array}{c|c|c} H_Z^\intercal \otimes W_2 & \mathbb{1} \otimes W_2 H'_X & \mathbf{0} \\ \mathbf{0} & H_X \otimes U_2 & \mathbb{1} \otimes U_2 H'^\intercal_Z \end{array} \right) \\
&= \big(\mathbb{1} \otimes W_2 \,\big|\, \mathbb{1} \otimes U_2 \big)\, \tilde{H}_X \\
&= \tilde{W}_2 \tilde{H}_X \,,
\end{aligned} \tag{42a}$$

$$\begin{aligned}
\tilde{H}_Z \tilde{U}_2^{-\intercal} &= \left(\begin{array}{c|c|c} \mathbb{1} \otimes H'^\intercal_X W_2^{-\intercal} & H_Z \otimes U_2^{-\intercal} & \mathbf{0} \\ \mathbf{0} & \mathbb{1} \otimes H'_Z U_2^{-\intercal} & H_X^\intercal \otimes \hat{W}_2 \end{array} \right) \\
&= \left(\begin{array}{c|c|c} \mathbb{1} \otimes U_2^{-\intercal} H'^\intercal_X & H_Z \otimes U_2^{-\intercal} & \mathbf{0} \\ \mathbf{0} & \mathbb{1} \otimes \hat{W}_2 H'_Z & H_X^\intercal \otimes \hat{W}_2 \end{array} \right) \\
&= \big(\mathbb{1} \otimes U_2^{-\intercal} \,\big|\, \mathbb{1} \otimes \hat{W}_2 \big)\, \tilde{H}_Z \\
&= \tilde{W}'_2 \tilde{H}_Z \,,
\end{aligned} \tag{42b}$$

And so with (42a) and (42b) we have shown that $\tilde{U}_{Q'}$ (40b) preserves the codespace. We can make similar statements about the resulting $\tilde{W}$s as we did in the quantum × classical case, namely that they preserve circuit depth and are also permutations if the input gadgets are Tanner graph automorphisms.

The action of the automorphism gadgets (40) on the canonical logical basis of the middle sector (24) follows an analogous tensor product structure compared to both the HGP (31) and the (quantum × classical) homological product (38) cases:

$$\tilde{\overline{V}}_1 = \overline{V}_1 \otimes \mathbb{1} \tag{43a}$$

$$\tilde{\overline{V}}_2 = \mathbb{1} \otimes \overline{V}_2 \,. \tag{43b}$$

As a reminder, both $\tilde{\overline{V}}_1, \tilde{\overline{V}}_2 \in \tilde{\mathcal{A}}$ are elements of the logical automorphism gadget group.

5 Fault tolerance of automorphism gadgets

In this section, we analyze the fault tolerance of our logical automorphism gadgets constructed in Section 4. Because we have described three different types of constructions for homological product codes, i.e. classical × classical, quantum × classical, and quantum × quantum, we divide this section into three different subsections that analyze the fault tolerance of the automorphism gadgets for these product codes. For a vector $\mathbf{x} \in \mathbb{F}_2^n$, let $\mathbf{x}_L$ and $\mathbf{x}_R$ denote the restrictions of $\mathbf{x}$ to the physical qubits in the left and right sectors respectively. For the case of the quantum × quantum homological product code, we let $\mathbf{x}_M$ to denote the restriction to the physical qubits in the middle sector. For example, the middle physical qubit sector of a quantum × quantum homological product code corresponds to the qubit subspace $(\mathrm{Q}, \mathrm{Q}')$ and can be laid out in a grid fashion that is represented by a matrix in $\mathbb{F}_2^{n_\mathrm{Q} \times n'_\mathrm{Q}}$. Refer to Figure 6 for a diagrammatic representation of these grids and matrices.

For our noise model of interest, we will assume that all single-qubit gates (including idling), multi-qubit gates and measurements are subject to failure. On the other hand, we will assume that physical permutations do *not* spread errors. This last assumption is based on the observation that for architectures with movable qubits, the SWAP gates can be performed by physical motion rather than 2-qubit gates that can cause correlated errors. Often, we can simply relabel the qubits in software rather than perform the physical permutations themselves, as has been typically done in recent experiments [BDSB24, HDSHL24, RAC+24]. The physical permutations only need to be performed when performing other fault-tolerant gadgets such as an interblock transversal gate or an intrablock fold-transversal gate; see Sec. 5.4 for more details. Combining the above assumptions on the noise with the structure of logical operators in homological product codes, we will show that our automorphism gadgets possess a form of inherent fault tolerance with respect to circuit-level noise.

5.1 Fault tolerance for hypergraph product codes

For simplicity, we will only analyze logical qubits living in the left sector of physical qubits, as these are the only ones we care about for computation. We will leverage the following results in the literature regarding the structure of logical operators in hypergraph product codes. Let $|\cdot|_\mathrm{L}$ denote the weight on the left sector Λ_L of data qubits.

Lemma 5.1 (Hypergraph product logical sector weight; Prop. 2 of [QC22])**.** *Suppose we have a HGP code with parity-check matrices H_X, H_Z of the form (14) from the hypergraph product of $[n_1, k_1, d_1]$ and $[n_2, k_2, d_2]$ classical linear codes. Let $\mathbf{x}_\mathrm{L} \in \ker H_Z \setminus \operatorname{rs} H_X$ denote the support of a nontrivial left-sector logical $\overline{X}$ operator, and let $\mathbf{z}_\mathrm{L} \in \ker H_Z \setminus \operatorname{rs} H_X$ denote the support of* *a nontrivial left-sector logical $\overline{Z}$ operator. Let $\hat{\mathbf{x}}_{\mathrm{L}}$ and $\hat{\mathbf{z}}_{\mathrm{L}}$ denote their canonical representations according to* (16)*. Then we have*

$$|\mathbf{x}_{\mathrm{L}}|_{\mathrm{L}} \geq d_2 \quad , \quad |\mathbf{z}_{\mathrm{L}}|_{\mathrm{L}} \geq d_1 \,. \tag{44}$$

A consequence of Lemma 5.1 is that the weight of any logical operator on the left sector cannot be decreased upon appending stabilizers. From the HGP code parameters (15) as well as the description of the HGP automorphism gadgets (26), we note that we will sometimes want to ignore the right-sector logical qubits and treat them as gauge qubits which hold no logical information. When we ignore the right-sector logical qubits, our original HGP code, which was a CSS stabilizer code, will now become a CSS *subsystem* code.

Definition 5.2 (Left-sector hypergraph product code)**.** *Given a HGP code with canonical logical operators* (16)*, the* left-sector *HGP code is the CSS subsystem code obtained upon designating the right-sector logical qubits as gauge qubits.*

Since we are simply ignoring the right-sector logical qubits, the number of data qubits remains the same. The number of logical qubits, however, now decreases to $k = k_1 k_2$ (we have lost the $k_1^{\mathsf{T}} k_2^{\mathsf{T}}$ right-sector logical qubits). It is not hard to see that Lemma 5.1 remains unchanged upon ignoring right-sector logical qubits because their canonical operators do not have any support in the left-sector.

Lemma 5.3 (Extension of Lemma 5.1)**.** *For the case where the classical input codes have rank-deficient parity-check matrices, Lemma 5.1 applies to all logical qubits in the left-sector HGP code given by Definition 5.2.*

Proof. Similar to the left-sector logical qubits (16), a canonical Pauli basis for the right-sector logical qubits can be written as

$$G_{Z,\mathrm{R}} = \left(\mathbf{0} \,\middle|\, \{\mathbf{e}_i\} \otimes g_{2,\mathsf{T}} \right) \tag{45a}$$

$$G_{X,\mathrm{R}} = \left(\mathbf{0} \,\middle|\, g_{1,\mathsf{T}} \otimes \{\mathbf{e}_j\} \right) , \tag{45b}$$

where $g_{1,\mathsf{T}}$ and $g_{2,\mathsf{T}}$ are generator matrices for the transpose codes (with parity-check matrices h_1^{T} and h_2^{T}), and $\{\mathbf{e}_i\} \notin \mathrm{rs}\, h_1^{\mathsf{T}}$ and $\{\mathbf{e}_j\} \notin \mathrm{rs}\, h_2^{\mathsf{T}}$. When we designate the right-sector logical qubits as gauge operators, the above logical operators can "dress" our left-sector logical operators in addition to the code stabilizers. A generic logical $\overline{X}$ operator now takes the form

$$\mathbf{x} = \hat{\mathbf{x}}_{\mathrm{L}} + \mathbf{s}_X + \hat{\mathbf{x}}_{\mathrm{R}} \,, \tag{46}$$

where $\hat{\mathbf{x}}_{\mathrm{L}}$ is a canonical left-sector logical $\overline{X}$ operator, $\mathbf{s}_X$ is an element of the X-stabilizer subgroup, and $\hat{\mathbf{x}}_{\mathrm{R}}$ is a canonical right-sector logical $\overline{X}$ operator. Now, since $\hat{\mathbf{x}}_{\mathrm{R}}$ has no support on the left sector, i.e. $|\hat{\mathbf{x}}_{\mathrm{R}}|_{\mathrm{L}} = 0$, we have that

$$|\mathbf{x}|_{\mathrm{L}} = |\hat{\mathbf{x}}_{\mathrm{L}} + \mathbf{s}_X|_{\mathrm{L}} \geq d_2 \,, \tag{47}$$

where we used Lemma 5.1 in the last inequality. An analogous argument holds for the logical $\overline{Z}$ operators. □

Proposition 5.4 (Distances of left-sector hypergraph product codes)**.** *For a left-sector HGP code with CSS parity-check matrices* (14) *and classical input code parameters $[n_1, k_1, d_1]$ and $[n_2, k_2, d_2]$, the X and Z distances are given by*

$$d_X = d_2 \quad , \quad d_Z = d_1 \,. \tag{48}$$

Proof. Lemma 5.3 implies that $d_X \geq d_2$ and $d_Z \geq d_1$. At the same time, our canonical logical basis (16) tells us that there exists a logical $\overline{X}$ with weight d_2 and a logical $\overline{Z}$ with weight d_1, and so we also have $d_X \leq d_2$ and $d_Z \leq d_1$. Thus, we arrive at (48). □

We are now ready to show that our automorphism gadgets preserve the full distance of left-sector HGP codes under our noise assumptions.

Theorem 5.5 (Effective distance preservation of automorphism gadgets in left-sector hypergraph product codes)**.** *Suppose physical qubit permutations do not spread errors, and we have a* $[\![n,k,d]\!]$ *left-sector HGP code according to Definition 5.2 and Proposition 5.4 with automorphism gadgets given by* (26)*. Then the effective fault distance of the automorphism gadgets is* $d_{\text{eff}} = d$*. In other words,* d *elementary faults are required to effect a nontrivial logical operation in the code.*

Proof. The proof essentially follows from Lemma 5.3 and Proposition 5.4, which say that the logical support on the left sector of physical qubits cannot be less than the code distance. As a consequence, at least d elementary faults must occur on the left sector in order to effect a nontrivial logical operation on the code. Since our HGP automorphism gadgets (26) only permute the left sector, they do not spread errors within the left sector and thus preserve the distance of the code. □

5.2 Fault tolerance for homological product codes: quantum × classical

For the case of classical × quantum homological product codes, the analysis is largely similar. We first begin by stating a recent result of Tan and Stambler [TS24] before connecting it with the fault tolerance of our automorphism gadget for the quantum × classical homological product code.

Lemma 5.6 (Homological product logical sector weight; Lemma 26 of [TS24])**.** *Suppose we have a (quantum × classical) homological product code with a classical input code with distance* d_{c} *and a quantum CSS input code with distances* (d_X, d_Z)*. In addition, suppose that the parity-check matrix of the classical input code has full rank; i.e. there are no redundant checks. Let* $\mathbf{x} \in \ker \tilde{H}_Z \setminus \operatorname{rs} \tilde{H}_X$ *denote the support of a nontrivial logical* $\overline{X}$ *operator, and let* $\mathbf{z} \in \ker \tilde{H}_Z \setminus \operatorname{rs} \tilde{H}_X$ *denote the support of a nontrivial logical* $\overline{Z}$ *operator. Then we have*

$$|\mathbf{x}|_{\mathrm{L}} \geq d_X d_{\mathrm{c}} \quad , \quad |\mathbf{z}|_{\mathrm{L}} \geq d_Z \,. \tag{49}$$

Again, we observe that the weight of any logical operator on the left sector cannot be decreased upon appending stabilizers. A technical assumption of the above lemma is that the classical input code has a full-rank parity-check matrix. For our purposes, often this will not be the case, i.e. there are linearly dependent parity checks; see the later Section 6 for such examples. As a consequence, one might worry about additional logical qubits resulting from the spurious (co)homologies in the product construction that could potentially spoil the logical sector weight property of Lemma 5.6. Fortunately, and similar to Definition 5.2 in the HGP case, we will now show that these additional logical qubits can be chosen to live on the right sector and do not affect Lemma 5.6 when restricted to the left-sector.

Definition 5.7 (Left-sector homological product code)**.** *Given a (quantum × classical) homological product code with canonical logical operators* (20)*, the* left-sector *homological product code is the CSS subsystem code obtained upon designating the right-sector logical qubits as gauge qubits.*

Lemma 5.8 (Extension of Lemma 5.6)**.** *For the case where the classical input code has a rank-deficient parity-check matrix, Lemma 5.6 still holds for the left-sector homological product code given by Definition 5.7.*

Proof. We begin by constructing a canonical Pauli basis for the right-sector logical operators. From the Künneth formula, we know that there will be $k_{\mathrm{c}}^{\mathsf{T}}\left(n_{\mathrm{Q}} - \operatorname{rank} H_X^{\mathsf{T}}\right)$ spurious logical qubits in this right sector. Define

$$\tilde{G}_{Z,\mathrm{R}} = \left(\, \mathbf{0} \mid \{\mathbf{e}_i\} \otimes g_{\mathsf{T}} \,\right) \tag{50a}$$

$$\tilde{G}_{X,\mathrm{R}} = \left(\, \mathbf{0} \mid G_{X,\mathsf{T}} \otimes \{\mathbf{e}_j\} \,\right) , \tag{50b}$$

where $\{\mathbf{e}_i\} \notin \operatorname{rs} H_X^{\mathsf{T}}$ and $\{\mathbf{e}_j\} \notin \operatorname{rs} h^{\mathsf{T}}$, and g_{T} and $G_{X,\mathsf{T}}$ are the generator matrices of the transpose codes that satisfy $h^{\mathsf{T}} g_{\mathsf{T}}^{\mathsf{T}} = 0$ and $H_X^{\mathsf{T}} G_{X,\mathsf{T}}^{\mathsf{T}} = 0$ respectively. By construction, (50a) and (50b) form a canonical Pauli basis for the spurious logical qubits. The statement of the Lemma then follows by the simple observation that the left-sector and right-sector logical qubits' canonical bases act on disjoint data qubits, and so Lemma 5.6 is unaffected even when we dress our left-sector logical qubits with the right-sector gauge qubits. □

Proposition 5.9 (Distances of left-sector homological product code)**.** *For a left-sector homological product code with CSS parity-check matrices* (18) *and input code parameters* $[\![n_{\mathrm{Q}}, k_{\mathrm{Q}}, (d_X, d_Z)]\!]$ *and* $[n_{\mathrm{c}}, k_{\mathrm{c}}, d_{\mathrm{c}}]$*, the* X *and* Z *distances are given by*

$$\tilde{d}_X = d_X d_{\mathrm{c}} \quad , \quad \tilde{d}_Z = d_Z \,. \tag{51}$$

Proof. Lemma 5.8 implies that $\tilde{d}_X \geq d_X d_{\mathrm{c}}$ and $\tilde{d}_Z \geq d_Z$. At the same time, our canonical logical basis (20) tells us that there exists a logical $\overline{X}$ with weight $d_X d_{\mathrm{c}}$ and a logical $\overline{Z}$ with weight d_Z, and so we also have $\tilde{d}_X \leq d_X d_{\mathrm{c}}$ and $\tilde{d}_Z \leq d_Z$. Thus, we arrive at (51). □

Lemma 5.6 essentially gives us the fault tolerance of our automorphism gadgets acting on the left logical qubits, when the input gadgets are code automorphisms.

Theorem 5.10 (Effective distance preservation of automorphism gadgets in homological product codes, part 1)**.** *Suppose physical qubit permutations do not spread errors, and we have a (quantum* × *classical) left-sector homological product code according to Definition* 5.7 *with automorphism gadgets given by* (32)*. If the input gadgets of the input quantum CSS code are code automorphisms, i.e.* $U \in \mathrm{S}_{n_{\mathrm{Q}}}$ *in* (32)*, then the resulting automorphism gadgets will be distance-preserving.*

Proof. We will analyze the two automorphism gadgets (32) separately. For $\tilde{U}_{\mathrm{c}}$ (32a), note that its action on the left sector of physical qubits is a permutation. Since Lemma 5.8 says that the left-sector logical weight cannot be less than the code distance, the fault distance of $\tilde{U}_{\mathrm{c}}$ is equal to the code distance. The situation is analogous for $\tilde{U}_{\mathrm{Q}}$ (32b) because the $U \otimes \mathbb{1}$ acting on the left-sector is also a permutation. Again, Lemma 5.8 tells us that the left-sector logical weight cannot decrease, and so at least $\tilde{d}_X = d_{\mathrm{Q}} d_{\mathrm{c}}$ faults are required to effect a nontrivial logical $\overline{X}$, and at least $\tilde{d}_Z = d_{\mathrm{Q}}$ faults are required to effect a nontrivial logical $\overline{Z}$. □

The trickier case is when the input gadget of the input CSS code is not a strict code automorphism but rather a hybrid gadget ($U \notin \mathrm{S}_{n_{\mathrm{Q}}}$) such the automorphism gadgets in Section 4. Recall that our automorphism gadgets generically have physical operations that include both a permutation and a circuit, and so they fall outside of the assumptions of Theorem 5.10. If our input CSS code was a HGP code or another homological product code with automorphism gadgets, it would be nice if we could lift those gadgets to the combined homological product code. Fortunately, we can leverage the product structure once again to show that this is indeed the case. In fact, our proof technique is closely related to that of Theorem 4.2 in Ref. [EKZ22]. Let $|\cdot|_{\Lambda_{\mathrm{L}}}$ denote the weight on the $\Lambda_{\mathrm{L}} \otimes \mathbb{1} \subset \tilde{\Lambda}_{\mathrm{L}}$ subsystem, corresponding to the rows belonging to Λ_{L} of the input quantum code.

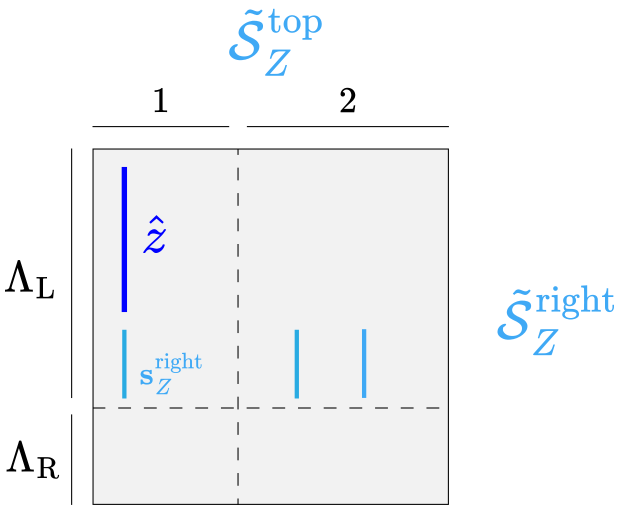
$\tilde{\mathcal{S}}_Z^{\text{top}}$
1
2
Λ_{L}
$\hat{z}$
$\mathbf{s}_Z^{\text{right}}$
$\tilde{\mathcal{S}}_Z^{\text{right}}$
Λ_{R}

Figure 5: The partitioning of $\tilde{\Lambda}_{\mathrm{L}}$ used for the proof of Proposition 5.11. $\hat{\mathbf{z}}$ is the support of a canonical logical $\overline{Z}$ operator, and $\mathbf{s}_Z^{\text{right}}$ is the support of an element of $\tilde{\mathcal{S}}_Z^{\text{right}}$.

Proposition 5.11 (Homological product restricted logical sector weight)**.** *Suppose we have a (quantum × classical) homological product code* (18) *where the input CSS code is either a left-sector HGP code by Definition* 5.2 *or a left-sector (quantum × classical) homological product code by Definition* 5.7*; denote this input left sector as* Λ_{L}*. Suppose the input CSS code has distances* (d_X, d_Z)*, and the input classical code has distance* d_{c}*. Let* $\mathbf{x} \in \ker \tilde{H}_Z \setminus \operatorname{rs} \tilde{H}_X$ *denote the support of a nontrivial logical* $\overline{X}$ *operator, and let* $\mathbf{z} \in \ker \tilde{H}_Z \setminus \operatorname{rs} \tilde{H}_X$ *denote the support of a nontrivial logical* $\overline{Z}$ *operator. Then we have*

$$|\mathbf{x}|_{\Lambda_{\mathrm{L}}} \geq d_X d_{\mathrm{c}} \quad , \quad |\mathbf{z}|_{\Lambda_{\mathrm{L}}} \geq d_Z \,. \tag{52}$$

Proof. Before we proceed, let $\tilde{\Lambda}_{\mathrm{L}}$ denote the left sector of the resulting homological product code, and let Λ_{L} denote the left sector of the input HGP or homological product code. Let $\tilde{\mathbf{r}}(\cdot) \in \mathbb{F}_2^{n_{\mathrm{Q}}}$ denote the row support in $\tilde{\Lambda}_{\mathrm{L}}$, and, within $\tilde{\Lambda}_{\mathrm{L}}$, let $\tilde{\mathbf{r}}_{\Lambda_{\mathrm{L}}}(\cdot)$ denote the *restricted* row support on the Λ_{L} rows of $\tilde{\Lambda}_{\mathrm{L}}$, corresponding to the left sector of the input HGP or homological product code (recall that there are $n_{\mathrm{Q}} = |\Lambda_{\mathrm{L}}| + |\Lambda_{\mathrm{R}}|$ total rows in $\tilde{\Lambda}_{\mathrm{L}}$).

We first analyze the logical $\overline{Z}$ operators. Suppose we have a nontrivial logical $\overline{Z}$ operator with support $\mathbf{z}$ and canonical support $\hat{\mathbf{z}}$ given in (20a). Note that there are two "types" of Z-checks that we need to analyze, given by the two row-blocks in (18b): a "top-type" Z-check and a "right-type". Note that any $\mathbf{s}_Z \in \operatorname{rs} \tilde{H}_Z$ can be decomposed into a linear combination of top-type and right-type checks: $\mathbf{s}_Z = \mathbf{s}_Z^{\text{top}} + \mathbf{s}_Z^{\text{right}}$. Furthermore, we will decompose the top-type Z-checks into those that act in the columns indexed by $\{\mathbf{e}_i\} \notin \operatorname{rs} h$ (region 1) and its complement (region 2); see Figure 5 for a sketch of the above partitioning. By linearity we can analyze the total contribution of the stabilizer element by summing the independent contributions from each constituent type; as such we have that $\mathbf{z} = \hat{\mathbf{z}} + \mathbf{s}_Z^{\text{top}_1} + \mathbf{s}_Z^{\text{top}_2} + \mathbf{s}_Z^{\text{right}}$. We begin with the contribution from $\mathbf{s}_Z^{\text{top}_1}$. Recall that the top-type Z-checks act as $H_Z \otimes \mathbb{1}$, i.e. independent copies of H_Z among the columns of $\tilde{\Lambda}_{\mathrm{L}}$. As a result, Lemma 5.8 tells us that $|\tilde{\mathbf{r}}_{\Lambda_{\mathrm{L}}}(\hat{\mathbf{z}} + \mathbf{s}_Z^{\text{top}_1})| \geq |\tilde{\mathbf{r}}_{\Lambda_{\mathrm{L}}}(\hat{\mathbf{z}})| \geq d_Z$. Now, notice that $\hat{\mathbf{z}} + \mathbf{s}_Z^{\text{top}_1}$ is still completely supported in the columns of region 1. Since these columns are not in $\operatorname{rs} h$ by definition, whenever we try to reduce the weight in any column in region 1 by appending a right-type Z-check, we must spawn additional support in new columns in region 2. As a result, the row weight cannot be decreased from these right-type Z-checks; i.e. we have $|\tilde{\mathbf{r}}_{\Lambda_\mathrm{L}}(\hat{\mathbf{z}} + \mathbf{s}_Z^{\mathrm{top}_1} + \mathbf{s}_Z^{\mathrm{right}})| \geq |\tilde{\mathbf{r}}_{\Lambda_\mathrm{L}}(\hat{\mathbf{z}} + \mathbf{s}_Z^{\mathrm{top}_1})| \geq |\tilde{\mathbf{r}}_{\Lambda_\mathrm{L}}(\hat{\mathbf{z}})| \geq d_Z$. Finally, we will deal with $\mathbf{s}_Z^{\mathrm{top}_2}$. Note that after the previous step of including $\mathbf{s}_Z^{\mathrm{right}}$, if the row weight in region 1 is greater or equal to d_Z, then we are done because $\mathbf{s}_Z^{\mathrm{top}_2}$ has no support in region 1. So we only need to handle the case where the row weight in region 1 is decreased below d_Z due to $\mathbf{s}_Z^{\mathrm{right}}$. By the previous argument, any row weight missing from region 1 must be compensated in region 2. Now, suppose $\mathbf{s}_Z^{\mathrm{top}}$ was able to remove this new row weight in region 2 such that the total row weight $|\tilde{\mathbf{r}}_{\Lambda_\mathrm{L}}(\hat{\mathbf{z}} + \mathbf{s}_Z^{\mathrm{top}_1} + \mathbf{s}_Z^{\mathrm{right}} + \mathbf{s}_Z^{\mathrm{top}_2})| = |\tilde{\mathbf{r}}_{\Lambda_\mathrm{L}}(\mathbf{z})| < d_Z$. However, due to the parallel structure of $H_Z \otimes \mathbb{1}$ in the top-type checks, it immediately implies the existence of a $\mathbf{s}_Z^{\prime\mathrm{top}_1}$ in region 1 such that $|\tilde{\mathbf{r}}_{\Lambda_\mathrm{L}}(\hat{\mathbf{z}} + \mathbf{s}_Z^{\mathrm{top}_1} + \mathbf{s}_Z^{\prime\mathrm{top}_1})| \equiv |\tilde{\mathbf{r}}_{\Lambda_\mathrm{L}}(\mathbf{z}')| < d_Z$: simply take the checks in $\mathbf{s}_Z^{\prime\mathrm{top}_1}$ as well as their copies in the columns of region 1 and eliminate the row support in region 1 one column at a time, which is guaranteed if $\mathbf{s}_Z^{\prime\mathrm{top}_2}$ was able to remove row weight in region 2. Note that $\mathbf{z}' \equiv \hat{\mathbf{z}} + \mathbf{s}_Z^{\mathrm{top}_1} + \mathbf{s}_Z^{\prime\mathrm{top}_1}$ is related to $\hat{\mathbf{z}}$ by an element $\mathbf{s}_Z^{\mathrm{top}_1} + \mathbf{s}_Z^{\prime\mathrm{top}_1} \in \mathrm{rs}\,\tilde{H}_Z$ and so is a nontrivial logical $\overline{Z}$ operator itself. However, $|\tilde{\mathbf{r}}_{\Lambda_\mathrm{L}}(\mathbf{z}')| < d_Z$ is in contradiction with Lemma 5.8, and so we must have $|\mathbf{z}|_{\Lambda_\mathrm{L}} \geq |\tilde{\mathbf{r}}_{\Lambda_\mathrm{L}}(\mathbf{z})| \geq d_Z$.

For the X sector, observe that all X-checks act as $H_X \otimes \mathbb{1}$ in $\tilde{\Lambda}_\mathrm{L}$ and hence as independent copies of H_X among the columns, as illustrated in Figure 2. Since each column is effectively an independent copy of the HGP code from the perspective of the X-checks, we can perform the analysis column by column. From (20b), recall that a nontrivial canonical $\overline{X}$ operator has support $\hat{\mathbf{x}}$ among d_c columns. Within each of these columns, $\hat{\mathbf{x}}$ is supported on at least d_X rows. Now, Lemma 5.6 tells us that $|\tilde{\mathbf{r}}_{\Lambda_\mathrm{L}}(\hat{\mathbf{x}} + \mathbf{s}_X)| \geq d_X$ within each column for any $\mathbf{s}_X \in \mathrm{rs}\,\tilde{H}_X$. Since this inequality holds for each column independently, we conclude that $|\mathbf{x}|_{\Lambda_\mathrm{L}} \geq d_X d_\mathrm{c}$. □

Theorem 5.12 (Effective distance preservation of automorphism gadgets in homological product codes, part 2)**.** *Suppose physical qubit permutations do not spread errors, and we have a (quantum × classical) left-sector homological product code according to Definition 5.7 with automorphism gadgets given by (32). If the input quantum CSS code is a left-sector HGP code according to Definition 5.2, with distance-preserving automorphism gadgets given by (26), then the inherited automorphism gadgets in the homological product code will still be distance-preserving.*

Proof. We will restrict our attention to $\tilde{U}_\mathrm{Q}$ in (32), since $\tilde{U}_\mathrm{c}$ falls within the scope of Theorem 5.10. In (32), note that the left-sector action of $\tilde{U}_\mathrm{Q}$ is given by $U \otimes \mathbb{1}$, which has the interpretation of applying U among the left-sector columns in parallel. If U is an automorphism gadget of the input HGP code, then it further factorizes according to (26). Let $\mathbf{z} \in \ker \tilde{H}_X \backslash \mathrm{rs}\,\tilde{H}_Z$ $\big(\mathbf{x} \in \ker \tilde{H}_Z \backslash \mathrm{rs}\,\tilde{H}_X\big)$ be the support of any nontrivial logical $\overline{Z}$ ($\overline{X}$) operator of the left-sector homological product code. Then Propositions 5.9 and 5.11 tell us that

$$|\mathbf{z}|_{\Lambda_\mathrm{L}} \geq \tilde{d}_Z \quad , \quad |\mathbf{x}|_{\Lambda_\mathrm{L}} \geq \tilde{d}_X \, . \tag{53}$$

Since the input automorphism gadget only acts as a permutation within the Λ_L rows of $\tilde{\Lambda}_\mathrm{L}$, at least $\tilde{d}_Z$ elementary faults are required to incur a logical $\overline{Z}$ error. Similarly, at least $\tilde{d}_X$ elementary faults are required to incur a logical $\overline{X}$ error. □

5.3 Fault tolerance for homological product codes: quantum × quantum

Before we begin our analysis on the fault tolerance of our automorphism gadgets for (quantum × quantum) homological product codes, we first discuss the code parameters of such homological product codes. These parameters would directly set the baseline for how we can expect our gadgets to perform fault-tolerantly.

We first state a few well-known techniques to modify the parity-check matrices of classical linear codes that we use to prove some facts regarding the parameters for our (quantum × quantum) homological product codes.

Definition 5.13 (Punctured code)**.** *Let $H \in \mathbb{F}_2^{m \times n}$ be the parity-check matrix of a classical binary linear code $\mathcal{C}$. In addition, let $B \subseteq [n]$ be a set of column indices. Then, we say that the classical linear code $\mathcal{C}$ is punctured with respect to B when its parity-check matrix is now given by the submatrix $H|_B := H[[m], B] \in \mathbb{F}_2^{m \times |B|}$.*

In other words, when we puncture a code with respect to a set of column indices, we are dropping all the columns in the matrix other than the columns corresponding to the set of indices.

Definition 5.14 (Shortened code)**.** *Let $G \in \mathbb{F}_2^{k \times n}$ be the generator matrix of a classical binary linear code $\mathcal{C}$. In addition, let $B \subseteq [n]$ be a set of column indices. Then, we say that the classical linear code $\mathcal{C}$ is shortened with respect to B when its generator is now given by the submatrix $G|^B := G[A, B] \in \mathbb{F}_2^{|A| \times |B|}$ for some $A \subseteq [k]$ such that $G[\{a\}, [n] \setminus B] = \mathbf{0}$ for any $a \in A$.*

Because we mainly work with parity-check matrices instead of generator matrices, we abuse the definition and replace G with H in Definition 5.14 when we say we shorten H with respect to some set of column indices. Recall that if H is the parity-check matrix of the classical code $\mathcal{C}$ that has a generator matrix G, then H is the generator matrix of the classical code $\mathcal{C}^\perp$ that is dual to $\mathcal{C}$. Thus, one can also consider shortening H as the equivalent of shortening the dual code $\mathcal{C}^\perp$.

Lastly, we define how we can augment the set of parity-checks imposed by a parity-check matrix H for a classical linear code $\mathcal{C}$.

Definition 5.15 (Augmented checks)**.** *Let $H \in \mathbb{F}_2^{m \times n}$ be the parity-check matrix of a classical binary linear code $\mathcal{C}$. Let $S = \left\{\mathbf{v}^\mathsf{T} \mid \mathbf{v} \in \mathbb{F}_2^n\right\}$ be a set of row vectors that correspond to new additional parity-checks that act on n bits. Then, the augmented parity-check matrix H^{+S} is the new parity-check matrix of a classical linear code $\mathcal{C}'$ with the set of rows S added to the set of rows in H. Unless otherwise stated, the set of rows S is appended at the bottom of the matrix H i.e.,*

$$H^{+S} := \begin{pmatrix} H \\ S \end{pmatrix}.$$

5.3.1 Code parameters for homological product codes: quantum × quantum

Recall that we are only interested in the middle logical qubits of the quantum × quantum homological product code by our construction in Section 2.7.2. To be more explicit, we are interested in keeping the logical qubits corresponding to the cohomology $\mathcal{H}^0[\mathrm{Q}] \otimes \mathcal{H}^0[\mathrm{Q}']$ for quantum codes Q and Q′ and treating the other cohomologies that correspond to $\mathcal{H}^i[\mathrm{Q}] \otimes \mathcal{H}^{-i}[\mathrm{Q}']$ as gauge qubits for $i \in \mathbb{Z} \setminus \{0\}$. By a simple evaluation of the rank of the cohomology $\mathcal{H}^0[\mathrm{Q}] \otimes \mathcal{H}^0[\mathrm{Q}']$, we see that the number of logical qubits for the homological product code $\tilde{\mathrm{Q}}$ is given by

$$\operatorname{rank}\left(\frac{\ker H_Z}{\mathrm{rs} H_X}\right) \cdot \operatorname{rank}\left(\frac{\ker H'_Z}{\mathrm{rs} H'_X}\right) = k_{\mathrm{Q}} \cdot k'_{\mathrm{Q}}.$$

Assuming that Q and Q′ are both associated to 3-term chain complexes, we have determined that this "middle-sector" homological product code $\tilde{\mathrm{Q}}$ corresponding to Q and Q′ has the following parameters:

$$\tilde{n} = n_{\mathrm{Q}} \cdot n'_{\mathrm{Q}} + m'_Z \cdot m_X + m_Z \cdot m'_X \tag{54}$$

$$\tilde{k} = k_{\mathrm{Q}} \cdot k'_{\mathrm{Q}} \tag{55}$$

Firstly, we make an argument in the following lemma that dressing our $\overline{X}$ and $\overline{Z}$ logical operators for our middle-sector homological product code with its gauge operators does not reduce the Hamming weight of our logical operators:

Lemma 5.16 (Dressed logical operators)**.** *Suppose we have a* $[\![n_{\mathrm{Q}}, k_{\mathrm{Q}}, (d_X, d_Z)]\!]$ *quantum code* Q *and a* $[\![n'_{\mathrm{Q}}, k'_{\mathrm{Q}}, (d'_X, d'_Z)]\!]$ *quantum code* Q′ *with parity-check matrices* H_X, H_Z, H'_X, H'_Z *respectively. Let* $\tilde{\mathrm{Q}}$ *be a middle-sector CSS code constructed by taking the homological product code between* Q *and* Q′*. Then, dressing the* $\overline{X}$ *logical operators from* $\mathcal{H}^0[\mathrm{Q}] \otimes \mathcal{H}^0[\mathrm{Q}']$ *and the* $\overline{Z}$ *logical operators from* $\mathcal{H}_0[\mathrm{Q}] \otimes \mathcal{H}_0[\mathrm{Q}']$ *strictly increases their Hamming weights.*

Proof. We begin by formulating a canonical logical operator basis for the X gauge operators. Recall that our set of X gauge operators is the set of spurious cohomologies $\{\mathcal{H}^i[\mathrm{Q}] \otimes \mathcal{H}^{-i}[\mathrm{Q}']\}_{i\in\mathbb{Z}\setminus\{0\}}$. Each of these spurious cohomologies is supported on the physical qubits that lie in the vector space $\mathrm{Q}^i \otimes \mathrm{Q}'^{-i}$ for $i \neq 0$ respectively. Note that Q_i is simply the i^{th} $\mathbb{F}_2$-vector space in the chain complex Q. In other words, when written in their canonical forms, these X gauge operators do not share support with the logical $\overline{X}$ operators since the logical $\overline{X}$ operators are completely supported on the qubits in the vector space $\mathrm{Q}^0 \otimes \mathrm{Q}'^0$. Therefore, dressing the logical $\overline{X}$ operators with an arbitrary X gauge operator only increases the support of the logical operator on a disjoint set of qubits. The argument for the dressed logical $\overline{Z}$ operators follows in the same way. □

Now, we proceed to state the following theorem that bounds the minimum distances of the (dressed) logical operators of the middle-sector homological product code $\tilde{\mathrm{Q}}$. In the theorem statement and its proof, we use $\tilde{\mathrm{Q}}$ to represent the quantum code and its associated chain complex interchangeably. It should be clear from the context which object we are referring to. Our proof of Proposition 5.17 uses techniques introduced in Refs. [TZ14, ZP19, TS24].

Proposition 5.17 (Distance bounds for middle-sector homological product codes)**.** *Suppose we have a* $[\![n_{\mathrm{Q}}, k_{\mathrm{Q}}, (d_X, d_Z)]\!]$ *quantum code* Q *and a* $[\![n'_{\mathrm{Q}}, k'_{\mathrm{Q}}, (d'_X, d'_Z)]\!]$ *quantum code* Q′ *with parity-check matrices* H_X, H_Z, H'_X, H'_Z *respectively. Let* $\tilde{\mathrm{Q}}$ *be a subsystem CSS code constructed by taking the homological product code between* Q *and* Q′*. Then, the 0-cosystolic and 0-systolic distances of the middle-sector homological product code are bounded by:*

$$\max(d_X, d'_X) \le d^0(\tilde{\mathrm{Q}}) \le d_X \cdot d'_X,$$

$$\max(d_Z, d'_Z) \le d_0(\tilde{\mathrm{Q}}) \le d_Z \cdot d'_Z$$

when we disregard the 0-cohomologies and 0-homologies introduced by the $\overline{X}$ *and* $\overline{Z}$ *gauge operators. In addition, let* $\mathbf{x}$ *and* $\mathbf{z}$ *be arbitrary non-trivial dressed* $\overline{X}$ *and* $\overline{Z}$ *logical operators for* $\tilde{\mathrm{Q}}$ *respectively. Then, we have:*

$$|\mathbf{x}_{\mathrm{M}}| \ge \max(d_X, d'_X),$$

$$|\mathbf{z}_{\mathrm{M}}| \ge \max(d_Z, d'_Z).$$

Proof. To prove the proposition's statement, we focus on establishing the distance bounds for the cohomology operators i.e., $\overline{X}$ logical operators. The arguments put forward will also hold for the homology operators i.e., $\overline{Z}$ logical operators. Recall that the logical operator basis for the middle-sector homological product code $\tilde{\mathrm{Q}}$ is given by Eq. 24b which we restate below for convenience:

$$\tilde{G}_{X,\mathrm{M}} = \left(\, \mathbf{0} \;\middle|\; G_X \otimes G'_X \;\middle|\; \mathbf{0} \,\right) = \left(\, \mathbf{0} \;\middle|\; \frac{\ker H_Z}{\operatorname{rs} H_X} \otimes \frac{\ker H'_Z}{\operatorname{rs} H'_X} \;\middle|\; \mathbf{0} \,\right).$$

Take will have support on at least $d_X \cdot d'_X$ physical qubits from the expression above. Thus, we trivially obtain the upper bound $d^0(\tilde{\mathrm{Q}}) \leq d_X \cdot d'_X$.

Next, we proceed to show that $|\mathbf{x}_{\mathrm{M}}| \geq \max(d_X, d'_X)$ for an arbitrary nontrivial $\overline{X}$ logical operator $\mathbf{x}$. Note that this directly implies that $d^0(\tilde{\mathrm{Q}}) \geq \max(d_X, d'_X)$ since $|\mathbf{x}_{\mathrm{M}}| \leq d^0(\tilde{\mathrm{Q}})$ in the worst case. To show that $|\mathbf{x}_M| \geq \max(d_X, d'_X)$, we prove that $|\mathbf{x}_M|_r \geq d_X$ and $|\mathbf{x}_M|_c \geq d'_X$ for a non-trivial $\overline{X}$ logical operator.

Suppose $|\mathbf{x}_M|_r < d_X$. Let $R_M \subseteq [n_{\mathrm{Q}}]$ such that for all $r \in R_M$, there exists some $c \in [n'_{\mathrm{Q}}]$ such that $\mathbf{x}_M[\{r\}, \{c\}]$ is non-trivial. Note that $|\mathbf{x}_M|_r = |R_M|$. Now, let us puncture H_Z with respect to R_M to obtain $H_Z|_{R_M}$. Recall that this means that we have dropped all columns of H_Z with indices that lie outside of R_M. We also shorten H_X with respect to R_M to obtain $H_X|^{R_M}$. Recall that this means that we have dropped all rows of H_X with non-trivial support on columns in R_M before dropping all columns in the resulting matrix with indices that lie outside of R_M. Suppose we have dropped ℓ rows from H_X in the process of shortening. Then, we add S, i.e. ℓ rows of all zeros, to $H_X|^{R_M}$ at the indices where they were dropped to obtain a shortened, augmented check matrix $\left(H_X|^{R_M}\right)^{+S} \in \mathbb{F}_2^{m_X \times |R_M|}$. Note that the orthogonality condition is still satisfied i.e., $H_X|^{R_M} H_Z|_{R_M}^{\mathsf{T}} = \left(H_X|^{R_M}\right)^{+S} H_Z|_{R_M}^{\mathsf{T}} = 0$.

Let us construct a chain complex (CSS code) $\mathrm{Q}[R_M]$ with the boundary maps $\left(H_X|^{R_M}\right)^{+S}$ and $H_Z|_{R_M}$. Because $|R_M| = |\mathbf{x}_M|_r < d_X$, we have a trivial 0-cohomology $\frac{\ker H_Z|_{R_M}}{\mathrm{rs}\, H_X|^{R_M}}$ for $\mathrm{Q}[R_M]$. Since $\mathrm{rs}\, H_X|^{R_M} = \left(H_X|^{R_M}\right)^{+S}$, we also obtain a trivial 0-cohomology group $\frac{\ker H_Z|_{R_M}}{\mathrm{rs}\left(H_X|^{R_M}\right)^{+S}}$ for our quantum code $\mathrm{Q}[R_M]$. Now, let us construct a new subsystem homological product code $\tilde{\mathrm{Q}}[R_M]$ from $\mathrm{Q}[R_M]$ and Q' that has check matrices $\tilde{H}_X[R_M]$ and $\tilde{H}_Z[R_M]$. We point out that our non-trivial $\overline{X}$ logical operator $\mathbf{x}$ is well-defined on the physical qubits of $\tilde{\mathrm{Q}}[R_M]$ since the space of 0-cells in both $\tilde{\mathrm{Q}}$ and $\tilde{\mathrm{Q}}[R_M]$ are completely identical. In other words, the qubit vector space can be decomposed into the same sets of qubit subspaces. In addition, we have $\tilde{H}_Z[R_M]\mathbf{x} = 0$.

Next, we claim that the subsystem homological product code $\tilde{\mathrm{Q}}[R_M]$ encodes no logical qubit. Using Proposition 2.7 and discarding all spurious cohomologies other than $\mathcal{H}^0[\mathrm{Q}[R_M]] \otimes \mathcal{H}^0[\mathrm{Q}']$, we see that

$$\operatorname{rank} \mathcal{H}^0[\tilde{\mathrm{Q}}[R_M]] = \operatorname{rank} \mathcal{H}^0[\mathrm{Q}[R_M]] \cdot \operatorname{rank} \mathcal{H}^0[\mathrm{Q}'] = 0 \cdot k' = 0.$$

In other words, $\mathbf{x}$ can be obtained from a linear combination of checks from $\tilde{H}_X[R_M]$ and X gauge operators from the discarded spurious 0-cohomologies. Then,

$$\mathbf{x}^{\mathsf{T}} = \alpha^{\mathsf{T}} \tilde{H}_X[R_M] + \mathbf{x}_{\text{gauge}}^{\mathsf{T}}$$

for some $\alpha \in \mathbb{F}_2^{|R_M| m'_X + m_X n'_{\mathrm{Q}}}$ and some $\mathbf{x}_{\text{gauge}}$ that is some linear combination of $\overline{X}$ gauge operators in the canonical basis. Note that there are many choices of α since we have introduced ℓ rows of zeros when we augmented the shortened check matrix $H_X|^{R_M}$. We choose α to be any such vector with trivial support in the rows with indices that directly interact with the row indices of the shortened, augmented check matrix for those ℓ rows in the larger $\tilde{H}_X[R_M]$ check matrix. Using Lemma 5.16, we know that our spurious 0-cohomologies can be written in a canonical basis that has trivial support on the qubit vector subspace $\mathrm{Q}[R_M]^0 \otimes \mathrm{Q}'^0$. This implies that $\mathbf{x}_M$ is obtained completely from the linear combination of X stabilizer generators in $\tilde{H}_X[R_M]$ i.e., $\mathbf{x}_M^{\mathsf{T}} = \alpha^{\mathsf{T}} \tilde{H}_X[R_M]$. Let us decompose α into $\begin{pmatrix} \alpha[T] \\ \alpha[B] \end{pmatrix}$ for $\alpha[T] \in \mathbb{F}_2^{m_X n'_{\mathrm{Q}}}$ and $\alpha[B] \in \mathbb{F}_2^{|R_M| m'_X}$. Using Eq. 23a, we have

$$\alpha[B]^{\mathsf{T}} (\mathbb{1}_{|R_M|} \otimes H'_X) + \alpha[T]^{\mathsf{T}} (\left(H_X|^{R_M}\right)^{+S} \otimes \mathbb{1}_{n'_{\mathrm{Q}}}) = \mathbf{x}_M^{\mathsf{T}}.$$

Let us denote $\alpha' \in \mathbb{F}_2^{n_{\mathrm{Q}} m'_X + m_X n'_{\mathrm{Q}}}$ that has the decomposition $\begin{pmatrix} \alpha[T] \\ \alpha'[B] \end{pmatrix}$ for the same $\alpha[T]$ and $\alpha'[B] \in \mathbb{F}_2^{n_{\mathrm{Q}} m'_X}$ is obtained by padding $\alpha[B]$ appropriately with zeros at the row indices re-introduced when we increase the dimension from $|R_M|$ to n_{Q}. It is not too hard to see that by our construction of the shortened, augmented check matrix $\left(H_X|^{R_M}\right)^{+S}$ and α, we have

$$\alpha'^{\mathsf{T}} \tilde{H}_X = \alpha'[B]^{\mathsf{T}} (\mathbb{1}_{n_{\mathrm{Q}}} \otimes H'_X) + \alpha[T]^{\mathsf{T}} (H_X \otimes \mathbb{1}_{n'_{\mathrm{Q}}}) = \mathbf{x}_M^{\mathsf{T}},$$

$$\alpha'^{\mathsf{T}} \tilde{H}_X + \mathbf{x}_{\text{gauge}}^{\mathsf{T}} = \mathbf{x}^{\mathsf{T}}.$$

Since $\mathbf{x}$ can be constructed from a linear combination of X stabilizer generators and gauge operators when $|\mathbf{x}_M|_r < d_X$, we have shown that $|\mathbf{x}_M|_r \geq d_X$ for $\mathbf{x}$ to be a non-trivial $\overline{X}$ logical operator. The same proof strategy would also work for showing that $|\mathbf{x}_M|_c \geq d'_X$. This gives us $|\mathbf{x}_M| \geq \max(d_X, d'_X)$ which naturally implies that $d^0(\tilde{\mathrm{Q}}) \geq \max(d_X, d'_X)$ when we consider the relevant 0-cohomologies in our middle-sector homological product code. The arguments above can also be used to show the same bounds for the dressed $\overline{Z}$ logical operators i.e., 0-homologies. □

Proposition 5.17 is related to the result described in Ref. [ZP20]. In the work done by Zeng and Pryadko, they provided upper and lower bounds for the (co)systolic distances of the tensor product of chain complexes. However, they only stated the bounds for the case where one of the chain complexes only has two non-trivial vector spaces and explored the case where some of the qubit vector spaces are projected away. Projecting away these qubit vector subspaces is necessary in their case because the shortening of H_X would have decreased the dimension of the qubit subspaces eg. $\mathrm{Q}^1 \otimes \mathrm{Q}'^{-1}$ which could have potentially rendered the nontrivial logical $\overline{X}$ operator to be undefined in that qubit subspace if it was supported on the physical qubits that have now disappeared as a result of the shortening.

In our result, we have kept all the qubit vector spaces and considered chain complexes of arbitrary lengths; however, we have also chosen to discard the spurious (co)homologies. We were able to sidestep the technical issue that Zeng and Pryadko had by considering the augmentation of the H_X parity-check matrix to ensure that the dimensions of all qubit vector subspaces are preserved so that the nontrivial logical operator is still well-defined for the proof strategy to go through. Our result can also be understood as using two separate applications of Theorem 5.10 from Ref. [TS24]. In Figure 6, we have shaded two sets of six squares where one of the set of squares is shaded in blue and the other set of squares is shaded in red. Each set of six squares imposes a requirement on the qubits in M to have either column weight or row weight of d_Z or d'_Z in order for a Pauli Z operator to be a nontrivial logical operator of the middle-sector homological product code. This intuition guided the proof of Proposition 5.17. While the presence of two separate square blocks of checks that act row-wise and column-wise on the a single square block of qubits was able to constrain any non-trivial logical operator such that its support on that single block of qubits has weight at least $d_Z \cdot d'_Z$ for the quantum × classical homological product code, this is not the case for the quantum × quantum case because we also have two separate square blocks of checks that can potentially "erase" the support on the qubits in that single block by propagating them to the other left and right qubit blocks in an adversarial way as shown in Figure 6. Therefore, it is rather challenging to remove the gap between the lower and upper bound on the (co)systolic distances for the middle-sector homological product code in the most general case.

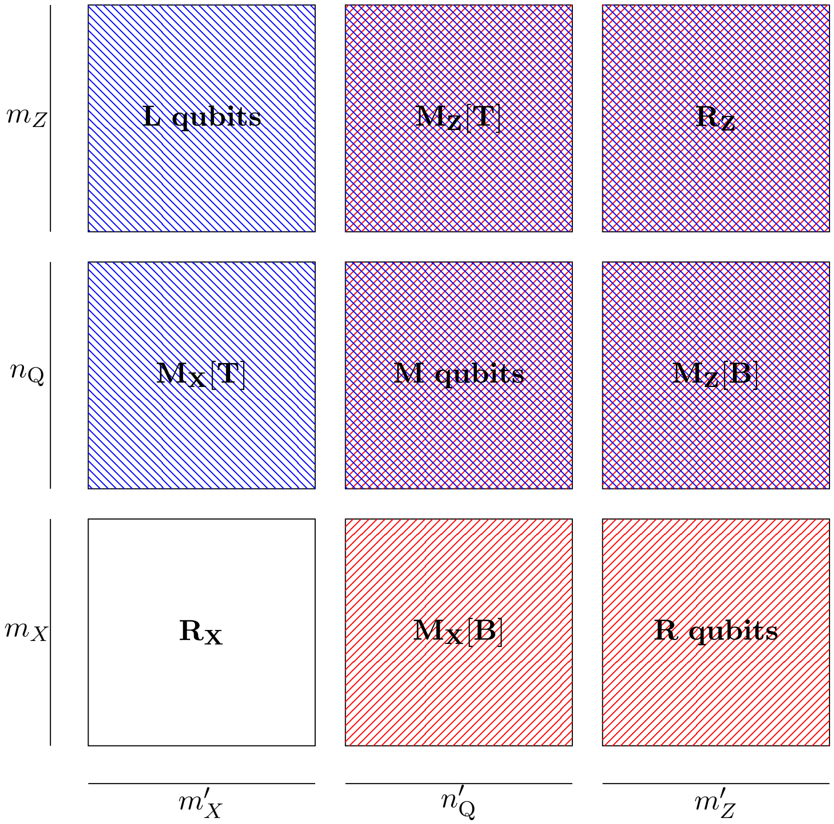

Figure 6: Schematic for the quantum × quantum homological product code constructed from quantum CSS codes with 3-term chain complexes. For each of the square grid in the figure, we can identify the relationship between the square grid and its adjacent neighbors from the linear (co)boundary maps inherited from the constitutent quantum codes. For example, the grid labeled with M qubits contains $n_{\mathrm{Q}} \cdot n'_{\mathrm{Q}}$ physical qubits. For each of the n_{Q} rows, we have X checks from $M_X[T]$ and Z checks from $M_Z[B]$ acting on a row of n'_{Q} physical qubits in M. There are thus effectively n_{Q} copies of the quantum code Q' that can be identified with these n_{Q} rows. $M_X[T]$ (corr. $M_Z[B]$) refers to the X (corr. Z) stabilizer generators that form the top (corr. bottom) row of the generator matrix in Eq. 23a (corr. Eq. 23b). The blue and red shaded squares in the figure allows us to "approximate" some of the interactions between the vector spaces of the product chain complex with two separate instances of a quantum × classical homological product code which each has six vector spaces.

5.3.2 Discussion on fault tolerance for quantum × quantum homological product codes

Based on the discussion in Section 5.3.1, we know that any non-trivial dressed $\overline{X}$ logical operator of a middle-sector homological product code has to have support of weight at least $\max(d_X, d'_X)$ on the M qubits i.e., the vector subspace $\mathrm{Q}^0 \otimes \mathrm{Q}'^0$. In the event where the dressed distance of the middle-sector homological product code is indeed $\max(d_X, d'_X)$, then we achieve the same kind of fault-tolerance discussed in the previous sections since the small circuits that can introduce faults only act on physical qubits that are not in the vector subspace $\mathrm{Q}^0 \otimes \mathrm{Q}'^0$. On the other hand, if the dressed distance of the middle-sector homological product code saturates the upper bound of $d_X \cdot d'_X$, then it is possible for our logical operators to have significant support outside of the vector subspace $\mathrm{Q}^0 \otimes \mathrm{Q}'^0$. This means that faults from the small circuits constructed in Section 4.3 that act on these other qubit subspaces can potentially contribute to logical errors in a way that is not inherently fault-tolerant.

We now use the bounds obtained in Proposition 5.17 to explicitly bound the extent of the fault tolerance of our automorphism gadgets of our (quantum × quantum) middle-sector homological product code. We state it in the following theorem and proof for completeness.

Theorem 5.18 (Effective distance bounds for automorphism gadgets in (quantum × quantum) middle-sector homological product codes)**.** *Suppose physical qubit permutations are noise-free, and we have a $[\![n, k, d]\!]$ middle-sector homological product code with automorphism gadgets given by (40). If the input gadgets of the input quantum CSS codes are distance-preserving, then the new automorphism gadgets of the middle-sector homological product code preserve the X and Z fault distances up to $\max(d_X, d'_X)$ and $\max(d_Z, d'_Z)$ respectively.*

Proof. We will analyze one of the two automorphism gadgets (40) for the X fault distance because the same argument would hold for the other automorphism gadget as well as the Z fault distance. Consider the gadget $\tilde{U}_{\mathrm{Q}}$ stated in (40a). The $U_1 \otimes \mathbb{1}$ acting on the middle physical qubits may not strictly be a permutation. Again, Proposition 5.17 tells us that the middle-sector logical weight is bounded from below by $\max(d_X, d'_X)$, and so we just need to also check that the $U \otimes \mathbb{1}$ acting on the "columns" of the quantum CSS code does not spread errors in a dangerous way. By assumption, U_1 does not decrease the fault distance of the input quantum CSS code below its code distance. In other words, at least $\max(d_X, d'_X)$ elementary faults are still required in the automorphism gadget to generate any dressed $\overline{X}$ logical operators (24) living in columns of the middle sector. Thus, at least $\max(d_X, d'_X)$ faults are required to effect a nontrivial logical $\overline{X}$, and at least $\max(d_Z, d'_Z)$ faults are required to effect a nontrivial logical $\overline{Z}$. □

In Ref. [ZP20], the following conjecture was proposed:

Conjecture 5.1 (Systolic distances of tensor product of bounded chain complexes [ZP20, Restatement of Conjecture 18])**.** *The i-systolic distance $d_i(\mathcal{A} \otimes \mathcal{B})$ in a tensor product of any pair of bounded chain complexes $\mathcal{A}$ and $\mathcal{B}$ of vector spaces over a finite field is given by*

$$d_i(\mathcal{A} \otimes \mathcal{B}) = \min_{j \in \mathbb{Z}} d_j(\mathcal{A}) d_{i-j}(\mathcal{B}).$$

When we consider the middle-sector version of our homological product code, we observe that the conjectured 0-(co)systolic distances in Ref. [ZP20] matched the upper bound for the distance of our middle-sector homological product code as stated in Proposition 5.17. In addition, Zeng and Pryadko noted in their paper that they performed extensive numerical simulations and were unable to find a single instance where Conjecture 5.1 was not true. Thus, it could very much just be a proof technique issue that is preventing us from showing $d^0(\tilde{\mathrm{Q}}) = d_X \cdot d'_X$ and $d_0(\tilde{\mathrm{Q}}) = d_Z \cdot d'_Z$ in Proposition 5.17. Because the proof that lower bounds the weight of any dressed $\overline{X}$ and $\overline{Z}$ logical operator in the $\mathrm{Q}^0 \otimes \mathrm{Q}'^0$ physical qubit vector space is closely related to the lower bound on the effective distance as a result of the automorphism gadget for our quantum× quantum homological product code, a proof for $d^0(\tilde{\mathrm{Q}}) = d_X \cdot d'_X$ and $d_0(\tilde{\mathrm{Q}}) = d_Z \cdot d'_Z$ could potentially imply that any dressed logical operator would also have weight at least $d_X \cdot d'_X$ or $d_Z \cdot d'_Z$ in the $\mathrm{Q}^0 \otimes \mathrm{Q}'^0$ physical qubit vector space. Thus, we conjecture the following:

Conjecture 5.2 (Effective distance preservation of automorphism gadgets in (quantum × quantum) middle-sector homological product codes)**.** *Suppose physical qubit permutations do not spread errors, and we have a* $[\![n, k, d]\!]$ *middle-sector homological product code with automorphism gadgets given by* (40)*. If the gadgets of the input quantum CSS codes preserve the effective distance, then so will the new automorphism gadgets of the homological product code.*

5.4 Alleviating correlated errors

Theorems 5.5, 5.10, 5.12 and 5.18 are statements regarding the fault tolerance of our automorphism gadgets. Although, in the most general case, we have to implement a quantum circuit on a subset of the physical qubits, for HGP and left-sector homological product codes this subsystem circuit does not spread errors in a dangerous way that decreases the effective circuit-level distance of the code, under the assumption of free permutations. Of course, in an actual quantum processor, physical qubit permutations may introduce noise due to crosstalk and heating from movement operations. In addition, idling errors that were negligible for short times may accumulate and become non-negligible when accounting for the time to implement the subsystem circuit portion of our automorphism gadgets. Because of these reasons, we emphasize that the effective distance preservation stated in Theorems 5.5 and 5.10 is *only relevant* when our simplified noise model can well-approximate the actual noise in the machines. In addition to idling errors, a potentially deep circuit on the right sector may cause small errors to propagate to large ones and confuse our classical decoding algorithm. Although our gadgets are distance-preserving, and so these large errors are *detectable*, it could be the case that there exists a logical operator with a large fraction of its support on the right sector. If the right sector has a large density of correlated errors, a simple minimum-weight decoder may potentially infer a wrong correction and incur a logical error.

The first, and perhaps most pressing, question to ask is when do we need to actually implement the subsystem circuit W? Suppose, for instance, that our fault-tolerant architecture is comprised of multiple code blocks that store all of our logical qubits for computation, and we are performing logical gates using a combination of automorphism gadgets, measurement (e.g. lattice surgery) gadgets, state-injection gadgets and transversal gates. For simplicity, we will examine the following cases where an automorphism gadget on a code block is immediately proceeded by:

1. A transversal measurement of the entire code block
2. A logical Pauli measurement
3. An interblock transversal gate

In case 1, a transversal measurement involves measuring every data qubit in the block in the X or Z basis. Such a measurement can be used to fault-tolerantly measure all logical qubits in the block in the $\overline{X}$ or $\overline{Z}$ bases and works for any CSS code. The statistics of the transversal measurement is both invariant to any permutation as well as any CNOT circuit. The key observation is that any permutation or CNOT circuit does not change the Pauli type of an observable, and so the outcome of any observable consisting of Pauli Xs or Pauli Zs before the automorphism gadget can be reconstructed by tracking how the observable would change through the gadget and combining the appropriate single-qubit measurement outcomes. For left-sector logical Pauli operators in our homological product codes, this tracking would involve a simple relabeling. So up to an in-software relabeling of the data qubits, we do not need to physically implement the automorphism gadget at all prior to the transversal measurement; in a sense, the transversal measurement "absorbs" the automorphism gadget.

For case 2, we would like to fault-tolerantly measure a logical Pauli operator, such as a single $\overline{X}_i$ or a joint $\overline{X}_i\overline{X}_j$, without disturbing the uninvolved logical qubits. Such an addressable logical Pauli measurement can be used to design addressable logical Clifford gates or perform targeted state injection. For intrablock logical operators, we can use either the general-purpose "adapters" of [SJOY25] or "extractors" of [HCWY25]; for interblock logical operators, we can also use the related "bridge" systems presented in both references. For these schemes, the specific ancillary system that is attached to the code block depends on the weight(s) of the logical operator(s) being measured as well as the structure of the code stabilizers within its neighborhood. Similar to the first case, when we have an automorphism gadget preceding our logical measurement, we can simply track the support of our logical operator through the permutation (recall that the gadget only acts as a permutation on the left sector) and attach the ancillary system to its new support. Equivalently, we can perform the physical permutation on the left sector and ignore the W on the right sector, which would perform the desired logical transformation (in the canonical basis) but change the underlying stabilizer code. The weight of the logical operator does not change through the permutation, and so the only adjustment to the ancillary system would come from the modified stabilizer structure. The only reason to implement the subsystem circuit W would be if one desires to use the *same* ancillary system to measure the logical operators both prior to and following the automorphism gadget.

For case 3, we envision that we have multiple blocks of the same code, and we would like to perform a transversal gate between the blocks. There are two situations to consider: when the other blocks are in known states versus unknown states. When the other blocks are in known states, as is the case with state injection, we can "offload" the automorphism gadget to the ancillary blocks as follows. Similar to the previous case, we can perform the relabeling part of the automorphism gadget on the left sector and ignore the circuit part on the right sector; however, when we do this relabeling, we will deform the underlying stabilizer code. In order for the transversal gate to work properly, the ancillary blocks will also need to be initialized (or distilled) according to the deformed code. Since the deformed code's logical operators have the same weight as the original code, we do not anticipate the distillation overhead on the deformed code to be substantially greater than that on the original code. For the second situation where the other blocks are in unknown states, as is the case when they are data/computational blocks themselves, the above trick no longer works. We will then generally need to perform the full automorphism gadget in order to properly realize the action of the transversal gate.

Now suppose that implementing the subsystem circuit W becomes unavoidable for reasons such as those mentioned above. We briefly mention a few ways to mitigate potential error propagation when W is not shallow. The proposed methods all come with their own space and time overheads, and any particular choice will boil down to the specific choice of code and automorphism gadget. The first is to introduce additional "flag" qubits that detect the spread of correlated errors and adjust our decoder accordingly, as has been done for fault-tolerant syndrome extraction of non-LDPC codes [CR18b]. With this method, one will also need to carefully schedule the flag qubits so that their inclusion does not decrease the effective distance. The second method involves finding a low-depth collection of $\{w\} \subseteq \mathcal{A}^\perp$ which generates the entire group $\mathcal{A}^\perp$; recall that $\mathcal{A}^\perp \simeq \mathcal{A}$ when

$d, d^{\perp} > 2$ from Corollary 3.6. Any element of $\mathcal{A}^{\perp}$ can then be compiled into a sequence of low-depth components interleaved with intermediate error correction (see Section 6.4 on Hamming codes for an explicit example). Since each component has low depth, the "light cone" of correlated errors is bounded, and so we can catch and correct them during the intermediate error correction cycles between the components. The cost of this second method is the additional time that comes with this compilation scheme. A third method also involves intermediate error correction, but we relax the requirement that the intermediate steps return back to the original codespace. Instead, we will keep track of how the stabilizer checks (and hence code) change through W and perform error correction using the "deformed" codes, a procedure known as pieceable fault tolerance [HFWH13, YTC16]; note that since W is Clifford, the deformed codes are still CSS. We can then perform intermediate error correction with respect to the new stabilizer checks using correlated decoding [CZZ+24]. The drawback with this approach is that we have no control over the weight of these new stabilizer checks, which could become rather large in the bulk of the W circuit.

Lastly, we briefly mention a potential decoding strategy that can complement the methods mentioned above. The strategy involves measuring an enlarged set of checks, or stabilizer generators, that lets us distinguish between elementary errors on the left sector and correlated errors on the right sector. We will analyze the case for HGP codes, but we note that an analogous transformation can be made for the homological product codes. Given a parity-check matrix $H \in \mathbb{F}_2^{m\times n}$ and a matrix $B \in \mathbb{F}_2^{\ell\times m}$, we can construct a new parity-check matrix $H' \equiv BH \in \mathbb{F}_2^{\ell\times n}$ with $\operatorname{rs} H' \subseteq \operatorname{rs} H$. The new parity checks are related to the original checks through B. For the HGP H_X matrix (14a), we will choose $B = \mathbb{1} \otimes g_2$. Then we have

$$H'_X = BH_X = \left(h_1 \otimes g_2 \,|\, \mathbb{1} \otimes g_2 h_2^{\top} \right) = \left(h_1 \otimes g_2 \,|\, \mathbf{0} \right), \tag{56}$$

where we have used the fact that $g_2 h_2^{\top} = 0$ by definition of g_2. A similar procedure can be used to construct new Z-checks using $B = g_1 \otimes \mathbb{1}$. Notice that these new checks are strictly supported in the left sector of the HGP code, and their extra syndrome information could potentially be used to distinguish between left-sector and right-sector errors in addition to the original checks in H_X. In fact, these new checks are precisely the stabilizer generators for the subsystem HGP [BC06] and subsystem homological product codes [ZP20], which can be viewed as a projection of the original HGP or homological product code onto the left sector of data qubits. However, the weight of these new checks are proportional to the classical code distances due to presence of g_1 (g_2) in H'_X (H'_Z), and as such additional resources will be required to extract their syndromes in a fault-tolerant manner.

6 Classical codes with automorphisms

In the previous section, we saw that classical and quantum code automorphisms induce fault-tolerant logical operations in the corresponding homological product codes. Due to the flexibility of the homological product, *any* choice of classical or quantum input codes can be tensored together, and so constructing a suitably symmetric homological product code effectively boils down to looking at symmetric input codes. In this section, we will explore various families of classical codes with rich automorphism structures that could potentially be used in our framework. Note that computing the automorphism group of a generic linear code is related to the Permutation Code Equivalence problem, which admits a polynomial-time reduction to Graph Isomorphism [PR97], for which the current fastest algorithm runs in quasipolynomial time [Bab16, HBD17]. Nonetheless, when a code admits special structure, such as a Reed-Muller code, its automorphism group can be efficiently computed.

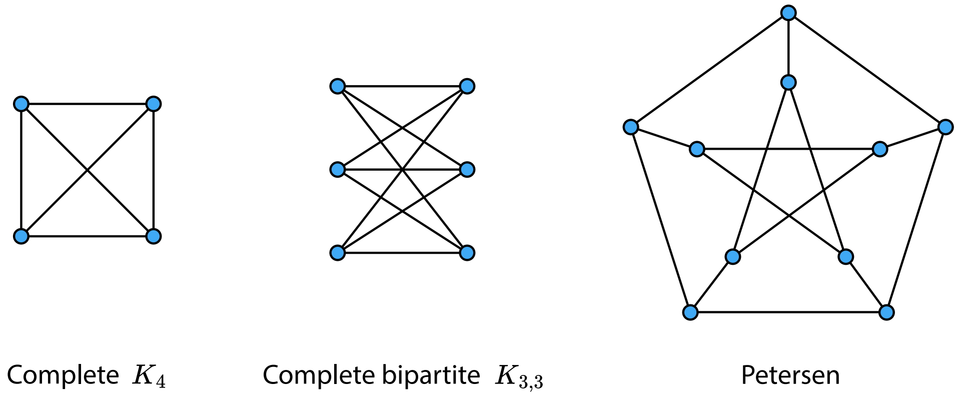

Figure 7: The graphs of the cycle codes in Table 2 are drawn.

6.1 Cycle codes of graphs

Given a simple, connected graph $\mathsf{G} = (V, E)$, the cycle code $\mathcal{C}(\mathsf{G})$ is defined as the linear code whose parity-check matrix $H \in \mathbb{F}_2^{|V|\times|E|}$ is the edge-vertex incidence matrix of G [Kas61, HB68]. In particular, we place bits on the edges of G and parity checks on the vertices. Using the fact that each edge must be attached to two vertices, the connectedness of G implies that the set of edges $\{e_j^{(i)}\}_j$ that lie in the neighborhood of any vertex $v_i \in V$ can be generated by "adding (modulo 2)" all the edges $\{e_\ell^{(k)}\}_\ell$ that lie in the neighborhood of all other vertices $v_k \in V \setminus \{v_i\}$. In other words, the cycle code $\mathcal{C}(\mathsf{G})$ has $|V| - 1$ linearly independent checks since any check can be obtained from a linear combination of the other checks.

Because a cycle code is defined according to a graph, many of its parameters can be efficiently computed from properties of this graph. If $\varDelta$ is the maximum degree of G, then H is $(2, \varDelta)$-LDPC, and we have $|E| \leq \varDelta|V|/2$ with equality if G is regular. The transpose code with parity checks on the edges and bits on the vertices is simply the repetition code, which is another way to see the global linear dependency among the rows of H. It follows that the code dimension is $k = |E| - |V| + 1$, which is also the rank of the fundamental group $\pi_1(\mathsf{G})$ of the graph. Logical codewords have support on closed loops or cycles in G. It follows that the code distance is given by the girth, or minimum cycle length of G, which can be efficiently computed in $O\big(|V||E|\big)$ operations via breadth-first search. Furthermore, since the Tanner graph of $\mathcal{C}$ is essentially given by G, automorphisms of G are automatically Tanner graph automorphisms of $\mathcal{C}$; in other words we have $\mathrm{Aut}(\mathsf{G}) \simeq \mathcal{T}(\mathcal{C}) \subseteq \mathrm{Aut}(\mathcal{C})$. Unfortunately, finding the automorphism group of a generic graph can only be done in quasipolynomial time [Bab16, HBD17] and so may become computationally intractable at large n. For some specific graphs, however, there can be simple arguments to deduce their automorphism groups.

As an example, a parity-check matrix for the complete graph on 4 vertices K_4 (see the leftmost graph in Figure 7 for an illustration) is given by

$$H_{K_4} = \underbrace{\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}}_{\text{edges}} \left.\vphantom{\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}}\right\} \text{vertices}\,. \tag{57}$$

Note that each column contains two 1s since each edge must be attached to two vertices, which

Cubic graphs	$[n,k,d]$	$d^{\perp}$	$\mathrm{Aut}(\mathsf{G}) \simeq \mathcal{T}$	$\lvert\mathrm{Aut}(\mathsf{G})\rvert$	$\mathrm{Aut}(\mathcal{C})$
Complete K_4	$[6,3,3]$	3	S_4	24	S_4
Complete bipartite $K_{3,3}$	$[9,4,4]$	3	$\mathrm{S}_3 \times \mathrm{S}_3 \rtimes \mathbb{Z}_2$	72	$\mathrm{S}_3 \times \mathrm{S}_3 \rtimes \mathbb{Z}_2$
Petersen	$[15,6,5]$	3	S_5	120	S_5

Table 2: Cycle code parameters for a selection of small cubic graphs with the largest known automorphism groups (for their size) are displayed. For these examples, the graph automorphism groups are the full code automorphism groups, i.e. $\mathcal{T} \simeq \mathrm{Aut}(\mathcal{C}) \simeq \mathcal{A}$, verified using the GUAVA package in GAP [Joy05].

also means that the sum of all rows is 0 (mod 2), i.e. each row can be obtained by summing all other rows. Thus, there are three linearly independent checks. Since K_4 is a complete graph, every vertex is connected to every other vertex, and so any permutation of the vertices leaves the graph unchanged. That is to say, the graph automorphism condition (4) is satisfied for any 4×4 permutation matrix acting to the left of (57).

Since we are interested in highly symmetric graphs with large automorphism groups, we will typically restrict to Δ-regular graphs; note that two vertices need to have the same degree in order to map to each other through an automorphism. For a Δ-regular graph with n edges, there are $2n/\Delta$ vertices, and so the corresponding cycle code has rate $k/n = 1 - 2/\Delta + 1/n$, which is nonvanishing for $n \to \infty$ as long as $\Delta > 2$. Note that $\Delta = 2$ corresponds to the ring graph which gives the $[n,1,n]$ repetition code. Unfortunately, as soon as $\Delta > 2$, the Moore bound on girth tells us that

$$g = d \leq 2 \log_{\Delta-1} n + O(1) \,. \tag{58}$$

As a consequence, if we try to construct a constant-rate LDPC cycle code, our code distance will be at most logarithmic in n. We know that (58) is close to being tight since the family of Ramanujan spectral expanders [LPS88] obey $g \geq \frac{4}{3} \log_{\Delta-1} n + O(1)$. Although the distance of constant-rate cycle codes are not quite satisfactory from an asymptotic standpoint, they can be rather competitive at small code sizes. Table 2 lists three examples with $n \leq 15$, taken from the Forster census of symmetric graphs with uniform degree 3 (cubic) up to 512 vertices [FB88]. Each of the listed codes also has $d^{\perp} > 2$, and so Corollary 3.6 tells us that each graph automorphism corresponds to a distinct logical operation.

For concreteness, we briefly discuss how we computed the graph automorphisms for the complete bipartite $K_{3,3}$ graph and the Petersen graph. For the complete bipartite $K_{3,3}$ graph, we can permute the three vertices on each side of the bipartition independently, giving us $\mathrm{S}_3 \times \mathrm{S}_3$. In addition, we can reflect the graph along the bipartition which effectively swaps the vertices of the two halves. Because reflections do not commute with the permutations that may take place on each side of the bipartition, we end up with a semi-direct product between $\mathrm{S}_3 \times \mathrm{S}_3$ and $\mathbb{Z}_2$. In the case of the Petersen graph, consider a bijective mapping that maps each of the 10 vertices in the graph to some unique ordered pair of indices in $\{1,2,3,4,5\}$ such that a vertex only shares an edge with another vertex if and only if their pairs of indices do not share any index. For example, the vertex that is associated with $(2,3)$ does not share an edge with the vertex that is mapped to $(3,5)$ but shares an edge with the vertices that are associated with $(1,4), (1,5)$ and $(4,5)$. It is not too hard to see that any permutation in S_5 preserves this relationship since we utilize all possible pairs of indices in $\{1,2,3,4,5\}$ with our 10 vertices. With a bit more work, one can show that there are no additional automorphisms beyond S_5 [FB88].

6.2 Group-algebra codes

Perhaps the most straightforward way to construct classical LDPC codes with automorphism groups is with the help of a group algebra. Specifically, given a finite group $\mathcal{G}$ of order $|\mathcal{G}| = n$, an element of the (binary) group algebra $a \in \mathbb{F}_2[\mathcal{G}]$ takes the form

$$a = \sum_{i=1}^{n} c_i g_i \,, \quad c_i \in \mathbb{F}_2 \,, \; g_i \in \mathcal{G} \,. \tag{59}$$

An intuitive way to think of the group algebra is that the coefficients c_i interact through addition while the group elements g_i control multiplication. To get a parity-check matrix out of (59), we simply replace each g_i with a binary matrix representation $\mathbb{B}[g_i]$, which is typically taken to be the (left) regular representation; the classical parity-check matrix is then given by $H \equiv \mathbb{B}[a]$. In the regular representation, each $\mathbb{B}[g_i]$ is an $n \times n$ permutation matrix whose rows and columns index the group elements and whose entries tabulate the group's Cayley multiplication table. Because the row and column weight of a permutation matrix is always 1, the row and column weights of H will be upper-bounded by the number of nonzero terms in the sum (59).

Suppose $\mathcal{G}$ is abelian. Then it is easy to see that for any classical group-algebra code $\mathcal{C}$ built from $\mathbb{F}_2[\mathcal{G}]$, we have $\mathcal{G} \subset \mathrm{Aut}(\mathcal{C})$ since for any $g_j \in \mathcal{G}$,

$$a g_j = \left(\sum_{i=1}^{n} c_i g_i \right) g_j = g_j \sum_{i=1}^{n} c_i g_i = g_j a \tag{60}$$

using the distributive property and the fact that $g_i g_j = g_j g_i$ for an abelian group. Plugging in the regular representation $H = \mathbb{B}[a]$ into (60), we immediately see that

$$H \mathbb{B}[g_j] = \mathbb{B}[g_j] H \,, \tag{61}$$

and upon comparing with the automorphism condition (25), we conclude that $\sigma = W = \mathbb{B}[g_j]$. Since $W = \sigma$ is also a permutation matrix, these transformations are graph automorphisms (Def. 2.2) and hence lift to exact automorphisms of the HGP code.

A commonly used family of abelian group-algebra codes are the cyclic codes, where we take $\mathcal{G} = \mathbb{Z}_n$. Since the elements of $\mathbb{Z}_n$ can all be generated by the unit translation x, it is customary to express (59) in terms of a degree $\leq n$ polynomial in x. For example, a parity-check matrix for the repetition code is given by the polynomial $a = 1 + x$. Taking $\sigma = W = \mathbb{B}[x]$, we see that the HGP operators (26) act as unit translations of all qubits in the $\hat{x}$ and $\hat{y}$ directions (with periodic boundaries). This particular automorphism "translation" gadget has been recently used to lower the spacetime overhead for fault-tolerant computation using HGP codes [XZZ+24]. In our framework, we see that this automorphism gadget arises from Tanner graph automorphisms associated with group-algebra codes over $\mathbb{F}_2[\mathbb{Z}_n]$.

For non-abelian $\mathcal{G}$, it is not immediately clear how we can get an analogue of (61) since we will not be able to commute $\mathbb{B}[g_j]$ through H in general. However, we can employ the following trick which has been previously used to construct both small-scale [LP24] as well as large-scale [PK22, DEL+22] LDPC codes. Since $\mathcal{G}$ is non-abelian, we need to distinguish between the left-regular and right-regular representations, corresponding to left and right group-multiplication respectively. Denote $L[\cdot]$ and $R[\cdot]$ the left-regular and right-regular representations accordingly. Then we can still use (59), but we define $h = L[a]$ to be the left-regular representation of a. Then $R[g]$ for any $g \in \mathcal{G}$ commutes with h due to the associativity of group multiplication, i.e. for $g_1 g_2 g_3$ the order of multiplying g_1 (g_3) on the left (right) does not matter. Table 3 lists group-algebra codes arising from the abelian group $\mathbb{Z}_7$ as well as the non-abelian groups D_6 and D_8.

Base group $\mathcal{G}$	H	$[n = \lvert\mathcal{G}\rvert, k, d]$	$d^{\perp}$	$\mathrm{Aut}(\mathcal{C})$	$\lvert\mathrm{Aut}(\mathcal{C})\rvert$
$\mathbb{Z}_7$	$1 + x + x^3$	$[7, 3, 4]$	3	$\mathrm{GL}_3(\mathbb{F}_2)$	168
$\mathrm{D}_6 \simeq \mathbb{Z}_6 \rtimes \mathbb{Z}_2$	$1 + r + sr^{-1}$	$[12, 4, 6]$	3	$(\mathrm{A}_4^2 \rtimes \mathbb{Z}_2) \rtimes \mathbb{Z}_2$	576
$\mathrm{D}_8 \simeq \mathbb{Z}_8 \rtimes \mathbb{Z}_2$	$1 + r^2 + r^3 + sr^{-1}$	$[16, 6, 6]$	4	$\mathbb{Z}_2^4 \rtimes \mathrm{S}_6$	11520

Table 3: Parameters for a few small abelian and non-abelian group-algebra codes are displayed. The first code is the dual Hamming or simplex code of length 7. For the dihedral group D_ℓ of order 2ℓ, we use the left-regular representation according to the presentation $\mathrm{D}_\ell = \langle r, s \mid r^\ell = s^2 = (rs)^2 = 1\rangle$. Automorphism groups are computed using the GUAVA package in GAP [Joy05].

Although their construction is relatively straightforward, group-algebra codes have their number of physical bits equal to the size of the group of interest, $n = |\mathcal{G}|$, as a consequence of the regular representation. For example, if one wishes to use $\mathcal{G} = \mathrm{S}_\ell$, then the number of physical bits is $n = \ell!$. As we will see in the later examples, there exist other classical code constructions where the number of physical bits can be much less than the size of the group.

6.3 Lifted group-algebra codes

We can generalize group-algebra codes by using a matrix of polynomial instead of a single (scalar) polynomial, yielding lifted group-algebra codes. Lifting is a common technique used in the literature on classical coding theory to generate large LDPC codes from smaller known LDPC codes. Given an arbitrary Tanner graph that corresponds to some classical LDPC code $\mathcal{C}$, we can lift it to an LDPC code with block length that is ℓ times larger. The resulting Tanner graph is called an *ℓ-lift* or ℓ-fold *cover graph* for the base Tanner graph. An easy way to interpret the result of the lift is to consider replacing each vertex v of the base graph with ℓ copies $v_1, \ldots, v_\ell$ of the same vertex and each edge e that connects v and v' with ℓ copies of the same edge $e_1, \ldots, e_\ell$. In the *ℓ-lift*, the edge e_i connects the vertices v_i and $v'_{\pi(i)}$ for some permutation $\pi \in S_\ell$. We emphasize that the permutation π does not need to be the same for the different edges in the base graph.

Consider *shift ℓ-lifts*, which are lifts based on a cyclic subgroup $\mathit{\Gamma}_\ell$ generated by the permutation $(1, 2, \ldots, \ell) \in S_\ell$. Before we elaborate on the algebraic structure of the parity check matrices that are obtained as a result of a shift ℓ-lift, we first define what an $\ell \times \ell$ circulant matrix is. We reuse the notation provided in Ref. [PK21b].

Definition 6.1 ($\ell \times \ell$ Circulant Matrix)**.** *An $\ell \times \ell$ circulant matrix M over the field $\mathbb{F}_q$ is the following:*

$$M = \begin{pmatrix} a_0 & a_{\ell-1} & \ldots & a_1 \\ a_1 & a_0 & \ldots & a_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{\ell-1} & a_{\ell-2} & \ldots & a_0 \end{pmatrix},$$

where $a_i \in \mathbb{F}_q$ for all $i \in [\ell - 1] \cup \{0\}$. The matrix M can also be decomposed into the following form:

$$M = a_0 \mathbb{1}_\ell + a_1 P + \ldots + a_{\ell-1} P^{\ell-1},$$

where $\mathbb{1}_\ell$ is the $\ell \times \ell$ identity matrix and P is the permutation matrix representing the right cyclic *shift* $(1, 2, \ldots, \ell) \in S_\ell$ *i.e.,*

$$P = \begin{pmatrix} 0 & 0 & \ldots & 1 \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 \end{pmatrix}.$$

It is not too hard to see that the ring of $\ell \times \ell$ circulant matrices over $\mathbb{F}_q$ is isomorphic to the ring of polynomials over $\mathbb{F}_q$ modulo the polynomial $x^\ell - 1$ i.e., $\mathbb{F}_q[x]/(x^\ell - 1)$, given $P^\ell = \mathbb{1}_\ell$. Thus, we can succinctly express the matrix M in the form of an $(\ell - 1)$-degree $\mathbb{F}_q$-polynomial:

$$m = a_0 + a_1 x + \ldots + a_{\ell - q} x^{\ell - 1}.$$

We denote $M \coloneqq \mathbb{B}(m)$ where $\mathbb{B}$ simply denotes the block matrix form of the polynomial. We also abuse notation by allowing $\mathbb{B}$ to act on matrices such that it changes the polynomial in each matrix entry into its block matrix form.

Now, suppose we are given a parity check matrix $H_0 \in \mathbb{F}_2^{c \times n}$. We can obtain a new parity check matrix $H \in \mathbb{F}_2^{c\ell \times n\ell}$ that corresponds to the shift ℓ-lift of the Tanner graph for H_0. The matrix H is a quasi-cyclic matrix i.e., a block matrix where each block is an $\ell \times \ell$ circulant matrix as defined in Def. 6.1. It is thus convenient to express the block matrices of H as the following: To construct H, we can first construct the matrix H_{poly} over the ring of polynomials $\mathbb{F}_2[x]/(x^\ell - 1)$ from H_0. To do that, we simply multiply each entry $(H_0)_{ij}$ of H_0 with some polynomial $m_{ij} \in \mathbb{F}_2[x]/(x^\ell - 1)$ to obtain $H_{poly} \in \left(\mathbb{F}_2[x]/(x^\ell - 1)\right)^{c \times n}$. We then use the isomorphism between $\mathbb{F}_2[x]/(x^\ell - 1)$ and the $\ell \times \ell$ circulant matrices described in Def. 6.1 to construct the individual block matrices $\mathbb{B}\left((H_0)_{ij}\, m_{ij}\right)$ that replaces each polynomial $(H_0)_{ij}\, m_{ij}$. This allows us to obtain $\mathbb{B}\left(H_{poly}\right) = H$.

We include an example below where we set $\ell = 4$:

$$H_0 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \tag{62}$$

$$m = \begin{pmatrix} x & 1 + x^2 & x^3 \\ 0 & x & x^2 \end{pmatrix}, \tag{63}$$

$$H_{\mathsf{poly}} = \begin{pmatrix} x & 1 + x^2 & 0 \\ 0 & 0 & x^2 \end{pmatrix}, \tag{64}$$

$$H = \mathbb{B}\left(H_{\mathsf{poly}}\right) = \left(\begin{array}{cccc|cccc|cccc} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right). \tag{65}$$

Without assuming any underlying automorphisms for H_0, the shift ℓ-lift for H_0, that is, H, is equipped with a set of automorphisms that arise from the lift. Consider the permutation of the following form:

$$\pi = \bigoplus_{i=1}^{n} \sigma,\ \sigma \in \mathit{\Gamma}_\ell.$$

In other words, the permutation π is a direct sum of permutations of the subgroup $\mathit{\Gamma}_\ell \subseteq S_\ell$ generated from $(1, 2, 3, \ldots, \ell)$. The matrix representation for π is the following:

$$\pi = \begin{pmatrix} \sigma & 0 & \ldots & 0 \\ 0 & \sigma & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma \end{pmatrix} = \begin{pmatrix} x^s & 0 & \ldots & 0 \\ 0 & x^s & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & x^s \end{pmatrix},$$

for some $s \in [\ell - 1] \cup \{0\}$. It is not difficult to see that

$$H\pi = \left(\bigoplus_{i=1}^{c} \sigma \right) H =: WH$$

is due to the fact that $\mathit{\Gamma}_\ell$ is an abelian group. And similar to the case for the group-algebra codes, because W is also a permutation matrix, these transformations are graph automorphisms (Def. 2.2) and hence lift to exact automorphisms of the HGP code.

It is useful to consider the case where the group $\mathcal{G}$ is non-abelian. In fact, non-abelian lifts were essential for the construction of asymptotically good quantum LDPC codes [PK22]. Similar to the case for group-algebra codes, we face the same obstacle that prevents us from commuting π through H when $\mathcal{G}$ is non-abelian. However, it is easy to see that the trick used to circumvent the obstacle for the group-algebra codes can also be used in the case for the lifted codes. We simply utilize both the left and right actions of $\mathcal{G}$ which trivially commute by the associativity of group multiplication.

6.4 $[2^r - 1, 2^r - r - 1, 3]$ Hamming codes

Hamming codes are one of the first linear codes studied after the simple repetition code. They have minimum distance $d = 3$ and dimension $k = n - \log_2(n+1)$. They are known as perfect codes because they saturate the Hamming (sphere-packing) bound and thus partition the physical space $\mathbb{F}_2^n$ into sets (balls) of correctable errors. The parity-check matrix of a Hamming code of level r consists of all $2^r - 1$ nonzero bitstrings of length r in $\mathbb{F}_2^r$. As a result, each row has weight $(n+1)/2$ and the maximum column weight is r. As an example, consider the $[7, 4, 3]$ $r = 3$ Hamming code with parity-check matrix

$$H_{[7,4,3]} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}. \tag{66}$$

One can see in the above example that the 7 columns of H comprise all 7 nonzero bitstrings on 3 bits. Using the structure of H for the Hamming codes, one can easily show that $\mathcal{A}^\perp = \mathrm{GL}_r(\mathbb{F}_2)$ as follows. For any $W \in \mathcal{A}^\perp$, observe that the columns of WH transform according to W. Since W is invertible and H contains all nonzero bitstrings, the action of W simply permutes the columns of H, which implicitly defines a corresponding permutation matrix $\sigma \in \mathrm{S}_n$ such that we obtain the automorphism condition $WH = H\sigma$. So we have $\mathrm{GL}_r(\mathbb{F}_2) \subset \mathcal{A}^\perp$; at the same time, since H has rank r, we must also have $\mathcal{A}^\perp \subset \mathrm{GL}_r(\mathbb{F}_2)$. Thus, $\mathcal{A}^\perp = \mathrm{GL}_r(\mathbb{F}_2)$. In addition, since both the distance and dual distance are greater than 2, Corollary 3.6 tells us that the corresponding logical gate group $\mathcal{A} \simeq \mathcal{A}^\perp = \mathrm{GL}_r(\mathbb{F}_2)$. Furthermore, since S_r is a subgroup of $\mathrm{GL}_r(\mathbb{F}_2)$, we also have $\mathcal{T}(H) = \mathrm{S}_r$, which is maximal.

Note that the physical implementation of a generic matrix $W \in \mathcal{A}^\perp = \mathrm{GL}_r(\mathbb{F}_2)$ may involve multiqubit gates and a superconstant circuit depth. However, we can always decompose any W

into elementary column operations that physically involve simple SWAPs and CNOTs, the number of which is at most $r(r-1)$. Furthermore, since each elementary operation also belongs to $\mathcal{A} \simeq \mathrm{GL}_r(\mathbb{F}_2)$, each step in the decomposition is a code automorphism, and so we can interleave error correction along the way. To summarize, for the Hamming code family with the standard form of the parity-check matrix, we have $\mathcal{A} \simeq \mathrm{GL}_r(\mathbb{F}_2)$ with Tanner graph automorphism subgroup $\mathcal{T} \simeq \mathrm{S}_r$. An arbitrary $W \in \mathcal{A}^\perp$ can be decomposed into at most $r(r-1) = O(\log^2 n)$ steps $W = W_1 W_2, W_3, \ldots$ comprised of permutations $\mathcal{T}$ and constant-depth circuits.

6.5 $[2^r - 1, r, 2^{r-1}]$ simplex codes

The dual of the $[2^r - 1, 2^r - r - 1, 3]$ Hamming codes are known as the $[2^r - 1, r, 2^{r-1}]$ simplex codes. They have minimum distance $d = (n+1)/2$ and dimension $k = \log_2(n+1)$, which saturates the Plotkin bound. In determining the automorphism group of Hamming codes, we first calculated $\mathcal{A}^\perp$ and then applied Corollary 3.6. This $\mathcal{A}^\perp$ is $\mathcal{A}$ for the corresponding simplex code. Hence we again have that the corresponding logical gate group $\mathcal{A} \simeq \mathcal{A}^\perp \simeq \mathrm{GL}_r(\mathbb{F}_2)$. Since we have that $r = k$, we see that simplex codes saturate the bound of Corollary 3.3; they have the largest possible automorphism group for a given number of logical bits.

For the Hamming codes, we had $W \in \mathrm{GL}_r(\mathbb{F}_2)$, and so we were able to easily characterize the graph automorphism subgroup as well as decompose any element into elementary steps that could be implemented in constant depth interleaved with error correction. For the simplex codes, the Ws generate the same group but embedded in a larger matrix group $\mathrm{GL}_m(\mathbb{F}_2)$ where $m \geq 2^r - 1 - r > r$ for $r \geq 3$. For practical considerations, we would like to choose a parity-check matrix H which minimizes the (gate) overhead for implementing each W on the physical level. One choice of H is to leverage the fact that all binary Hamming codes can be written as cyclic codes. For a length $n = 2^r - 1$ Hamming code, its codewords can be generated by any primitive polynomial of degree r in $\mathbb{F}_2[x]/(x^n - 1)$ [MS77]. For the simplex codes, this means we can build H using the group-algebra construction with $\mathcal{G} = \mathbb{Z}_n$ and a primitive polynomial which is typically a trinomial[6]. As examples, the $[7, 3, 4]$ simplex code has parity-check matrix $H = \mathbb{B}[1 + x + x^3]$, and the $[15, 4, 8]$ simplex code has parity-check matrix $H = \mathbb{B}[1 + x + x^4]$. By writing the parity-check matrices of the simplex codes in cyclic form, we can ensure that we have $\mathbb{Z}_n$ as a subgroup of their Tanner graph automorphism groups.

6.6 Other Reed-Muller codes

Some of the previously mentioned codes can be considered instantiations of Reed Muller codes up to puncturing. In general, Reed Muller $\mathrm{RM}(r, m)$ codes [Mul54, ASY20] are defined by evaluating multivariate polynomials over $\mathbb{F}_2$. Consider the polynomial ring with m variables $\mathbb{F}_2[x_1, x_2, ..., x_m]$. For a polynomial $f \in \mathbb{F}_2[x_1, x_2, ..., x_m]$ and a vector $y = (y_1, y_2, ..., y_m) \in \mathbb{F}_2^m$, we can compute the evaluation of y, $f(y_1, y_2, ..., y_m)$. Now define $\mathrm{Eval}(f)$ to be the bitstring obtained from evaluating $f(y)$ each of the 2^m vectors in $\mathbb{F}_2^m$. The code $\mathrm{RM}(m, r)$ is then defined as:

$$\mathrm{RM}(r, m) = \big\{\mathrm{Eval}(f) \mid f \in \mathbb{F}_2[x_1, ..., x_m],\ \deg(f) \leq r\big\}. \tag{67}$$

Note that over $\mathbb{F}_2$, $x^n = x$, and so a degree r polynomial in $\mathbb{F}_2[x_1, ..., x_m]$ has monomials that look like $x_1 x_2 ... x_r$. From (67) it can be seen that $n = 2^m$. The number of logical bits k is simply the

[6] The constraint on the polynomial can be relaxed to only having a primitive greatest common divisor with $x^n + 1$ [MGF+25].

number of polynomials with degree less than or equal to r:

$$k = \sum_{i=0}^{r} \binom{m}{i}, \tag{68}$$

and the distance is $d = 2^{m-r}$. When $r = 0$, we obtain the $[2^m, 1, 2^m]$ repetition code, and when $r = m$, we obtain the $[2^m, 2^m, 1]$ trivial code. Additionally, other codes can be derived by interpolating between these values. A small example is the $[8, 7, 2]$ code $\mathrm{RM}(2, 3)$, which has the following generator matrix:

$$G = \begin{pmatrix} \mathrm{Eval}(1) \\ \mathrm{Eval}(x_1) \\ \mathrm{Eval}(x_2) \\ \mathrm{Eval}(x_3) \\ \mathrm{Eval}(x_1x_2) \\ \mathrm{Eval}(x_1x_3) \\ \mathrm{Eval}(x_2x_3) \end{pmatrix} = \begin{array}{c} \begin{array}{cccccccc} 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \tag{69}$$

Here, rows correspond to the input basis vectors, while columns correspond to the evaluations of the variables $Z_1, Z_2 \ldots Z_m$.

In general, the size of the automorphism group of $\mathrm{RM}(r, m)$ is the general affine group $\mathrm{GA}_m(\mathbb{F}_2) \simeq \mathbb{F}_2^m \rtimes \mathrm{GL}_m(\mathbb{F}_2)$. This can be seen easily from the code construction: each evaluation point of the variables x_i corresponds to a bit position. In total there are 2^m bit positions. An affine transformation on the variables corresponds to a permutation of the bit positions, thus any affine transformation is a valid automorphism.

Note that there are three groups to compare: $\mathrm{GL}_k(\mathbb{F}_2)$, $\mathrm{GL}_m(\mathbb{F}_2)$ and S_n:

- When $r = 1$, then $k = m + 1$. Here $\mathrm{GL}_m(\mathbb{F}_2)$ is roughly the same as $\mathrm{GL}_k(\mathbb{F}_2)$, while the set of permutations S_n is much bigger.

- When $r = m - 1$, then $k = 2^m - 1$. For this case, $\mathrm{GL}_k(\mathbb{F}_2) = \mathrm{GL}_{n-1}(\mathbb{F}_2)$, and while the automorphism group is S_n, this is still much smaller than $\mathrm{GL}_k(\mathbb{F}_2)$. This is the repetition code like case.

- For $r = m/2$, then $k = n/2$. For this case, the size of the automorphism group is $|\mathrm{GL}_m(\mathbb{F}_2)| \approx (0.29)2^{m^2} = \Theta\big(n^{\log n}\big)$.

Alternatively, we can consider punctured RM codes:

6.6.1 Punctured Reed Muller codes as cyclic codes

Reed Muller codes are not cyclic in general, however, they can be punctured to obtain cyclic codes.

Definition 6.2. *A* ***punctured RM*** *code is denoted* $\mathrm{RM}^*(r, m)$ *and is defined by taking an* $\mathrm{RM}(r, m)$ *code and deleting the coordinate corresponding to* $Z_1, \ldots Z_m = 0$ *from all codewords.*

As given in [Mac64], we have the following theorem:

Theorem 6.3. *Punctured RM code $RM(r,m)^*$ corresponds to a cyclic code, which has zeros α^s that satisfy:*

$$1 \leq w_2(s) \leq m-r-1 \qquad 1 \leq s \leq 2^m - 2 \tag{70}$$

where s labels the cyclotomic cosets of the field $GF(2^m)$ that partition the integers as:

$$\{0,1,\ldots 2^m\} = \bigcup_s C_s \tag{71}$$

where s runs through the coset representatives $\mod 2^m$.

The minimal polynomial $M_s(x)$ of α^s is:

$$M^{(s)}(x) = \prod_{i \in C_s} (x - \alpha^i) \tag{72}$$

where α denotes the primitive element of the field.

The generator and check polynomials can be defined in terms of the minimal polynomials $M^{(s)}(x)$ of α^s.

$$g(x) = \prod_{1 \leq w_2(s) \leq m-r-1, 1 \leq s \leq 2^m-2} M^s(x) \tag{73}$$

$$h(x) = (x+1) \prod_{m-r \leq w_2(s) \leq m-1, 1 \leq s \leq 2^m-2} M^s(x) \tag{74}$$

It is worth noting that the automorphism group for punctured RM code $\mathrm{RM}^*(r,m)$ is $\mathrm{GL}_m(\mathbb{F}_2)$. We highlight two specific examples which have interesting properties. The first example contains the punctured $\mathrm{RM}^*(1,m)$. These codes correspond to dual Hamming or simplex codes. In particular, take $\mathrm{RM}(1,m)$ and drop the first bit and the first row corresponding to all ones. The resulting $[2^m - 1, m, 2^{m-1}]$ is the dual of the Hamming code with parameters $[2^m - 1, 2^m - m - 1, 3]$. The generator polynomial for $RM^*(1,3)$ can be found to be:

$$g(x) = 1 + x^2 + x^3 + x^4, \tag{75}$$

which corresponds to the check polynomial $h(x)$ of the $[7,4,3]$ Hamming code.

The second example contains some punctured families of RM codes that can also be thought of as Bose–Chaudhuri–Hocquenghem (BCH) codes [BRC60a, BRC60b, Hoc59], a special subclass of cyclic codes. As an example, let us look at $\mathrm{RM}(2,5)$. By puncturing the code (say at the zero position) one obtains the cyclic BCH code $[31, 16, \geq 7]$ code. The generator polynomial for the above BCH code is given as:

$$g(x) = x^{15} + x^{14} + x^{13} + x^{12} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1. \tag{76}$$

7 Examples of quantum codes with automorphism gadgets

In this section, we explicitly construct several hypergraph product codes, analyze their parameters, and provide some brief applications of their automorphism gadgets using the formalism from Section 4.

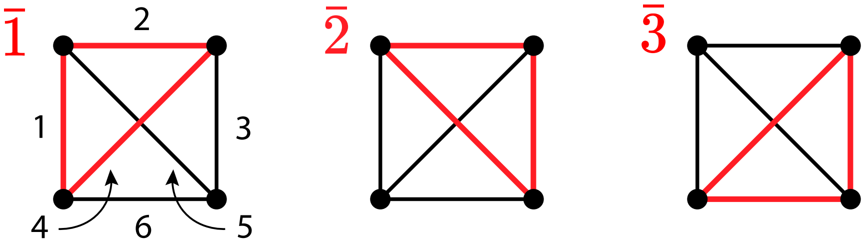

Figure 8: Codeword generators of the K_4 cycle code corresponding to the three rows of (77). Black numbers label the physical bits on edges, and red numbers label the logical bits.

7.1 $[\![52, 10, 3]\!]$ HGP code with $\mathcal{A} \simeq \mathrm{S}_4 \times \mathrm{S}_4$

For the first example, we will take the $[6, 3, 3]$ cycle code on the complete graph on 4 vertices (K_4). The parity-check matrix was defined in (57). Consider the following generator matrix:

$$g = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}, \tag{77}$$

which corresponds to three out of the four "triangular" cycles in K_4; see Figure 8 for an illustration. Using cycle notation for permutations, it is easy to verify that logical transposition $(\bar{1}\bar{2})$ is achieved via the physical permutation $(15)(34)$, and the logical transposition $(\bar{2}\bar{3})$ is achieved via $(24)(56)$. These two logical transpositions generate the entire logical permutation group S_3 on all three logical qubits. In addition to all logical permutations, we also have the logical transformation

$$v = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{78}$$

corresponding to the physical permutation $(25)(46)$. The rows of (78) describe the multiqubit logical gate $\mathrm{C}_{\bar{1}}\mathrm{NOT}_{\bar{2}\bar{3}} = \mathrm{C}_{\bar{1}}\mathrm{NOT}_{\bar{2}} \cdot \mathrm{C}_{\bar{1}}\mathrm{NOT}_{\bar{3}}$ where logical qubit $\bar{1}$ is controlled and logical qubits $\bar{2}$ and $\bar{3}$ are targeted. The logical permutations in combination with (78) generate the (Tanner graph) automorphism group S_4 for the K_4 cycle code. For each physical permutation σ described above, we also have a corresponding row-operation w given by $wh = h\sigma$. Since we are using a cycle code, every w will also be a permutation; for example, the w corresponding to (78) is the permutation (23). We take the HGP (14) of (57) with itself to obtain a $[\![52, 10, 3]\!]$ HGP code, with 36 data qubits and 9 logical qubits in the left sector, and 16 data qubits and 1 logical qubit in the right sector. The CSS parity checks have weight 5, and all qubits participate in either 2 or 3 X-type checks or Z-type checks. It will be convenient to represent the logical state of the 9 left logical qubits as a 3×3 block

$$|\overline{\Psi}_{\mathrm{L}}\rangle = \left| \begin{matrix} \overline{\psi}_1 & \overline{\psi}_2 & \overline{\psi}_3 \\ \overline{\psi}_4 & \overline{\psi}_5 & \overline{\psi}_6 \\ \overline{\psi}_7 & \overline{\psi}_8 & \overline{\psi}_9 \end{matrix} \right\rangle \tag{79}$$

akin to the geometric picture of Figure 1. The logical permutations of the classical code now lift to permutations of the "rows" and "columns" in (79) which corresponds to the group $\mathrm{S}_3 \times \mathrm{S}_3$. On the physical level, we will also be permuting rows and columns of both the left sector and right sector of data qubits, using σ for the left sector and w for the right sector. Note that the right logical qubit is invariant to all these automorphisms since its logical operators are repetition codewords (from the transpose of a cycle code). Adding in any "diagonal" logical SWAP such as $(\bar{1}\bar{5})$ from other means[7] enlarges this permutation subgroup $(\mathrm{S}_3 \times \mathrm{S}_3)$ to the full permutation group on all 9 left logical qubits. The inclusion of any two-qubit logical CNOT gate then enables us to compile the entire class of affine gates[8] on these 9 logical qubits.

7.2 $[\![48, 6, 3]\!]$ HGP code with $\mathcal{A} \simeq \mathrm{S}_4 \times \mathrm{S}_4$ and transversal CZ gates

For the second example, we will also take the K_4 cycle code with parity-check matrix (57) and generator matrix (77). But this time, we take the HGP of the K_4 cycle code with its transpose, the $[4, 1, 4]$ repetition code with bits on the 4 vertices of K_4 and parity checks on the 6 edges. The resulting HGP code has parameters $[\![48, 6, 3]\!]$ with 3 left logical qubits and 3 right logical qubits; note that there is a symmetry between the left and right sectors now. The X-type and Z-type checks have weights 6 and 4 respectively, and each qubit participates in 2 X-type and 3 Z-type checks. For this example, we will care about the right logical qubits, and so for clarity we write down the full canonical logical bases for both left and right sectors:

$$G_{Z,\mathrm{L}} = (\, g \otimes \mathbf{e}_1 \mid \mathbf{0} \,) \tag{80a}$$

$$G_{X,\mathrm{L}} = (\, E_1 \otimes \mathbf{1} \mid \mathbf{0} \,) \tag{80b}$$

$$G_{Z,\mathrm{R}} = (\, \mathbf{0} \mid \mathbf{e}_1 \otimes g \,) \tag{80c}$$

$$G_{X,\mathrm{R}} = (\, \mathbf{0} \mid \mathbf{1} \otimes E_1 \,)\,, \tag{80d}$$

where g is given by (77), $\mathbf{e}_1 = (1, 0, 0, 0) \notin \ker h^{\mathsf{T}}$, $\mathbf{1} = (1, 1, 1, 1)$ is the repetition codeword, and $E_1 \in \mathbb{F}_2^{3\times 6}$ is a matrix whose rows are not in $\ker h$. From the above logical basis, we notice that the first left automorphism gadget $U_{1,\mathrm{L}} = (\sigma \otimes \mathbb{1}) \oplus (\sigma' \otimes \mathbb{1})$ (here $\sigma' \equiv w$) acts nontrivially on the 3 left logical qubits and trivially on the 3 right logical qubits. Likewise, the second right automorphism gadget $U_{2,\mathrm{R}} = (\mathbb{1} \otimes \sigma') \oplus (\mathbb{1} \otimes \sigma)$ acts nontrivially on the right logical qubits and trivially on the left logical qubits. Note that for this example, $U_{2,\mathrm{L}}$ and $U_{2,\mathrm{R}}$ are redundant, similarly for $U_{1,\mathrm{L}}$ and $U_{1,\mathrm{R}}$. Thus, the group of logical automorphisms generated from $U_{1,\mathrm{L}}$ and $U_{2,\mathrm{R}}$ is isomorphic to $\mathrm{S}_4 \times \mathrm{S}_4$. The actions on the left and right logical qubits are identical to those in the classical K_4 cycle codes: all logical permutations as well as the multitargeted CNOT (78). Importantly, the '×' here represents independent operations between the left and right sectors, whereas for the first example everything was performed solely on the left sector.

There are several reasons to choose the transpose code as the second input code. Recall that the K_4 cycle code defines bits on edges and parity checks on the vertices of K_4. When we take the HGP of the K_4 cycle code with its transpose, the resulting HGP code can also be defined according to a graph: the Cartesian graph product $K_4 \times K_4 = K_4^2$. Qubits are located on the edges, X-checks on the vertices, and Z-checks on the square "faces" of the product graph K_4^2 spanned by an edge in one copy of K_4 and an edge in the second copy. As a consequence, the X-syndromes for Z errors can be decoded with minimum-weight matching, similar to that in the surface code; Z-syndromes however cannot be decoded in this manner and require other means. Topologically, a graph can be interpreted as a simplicial complex with vertices as 0-simplices and edges as 1-simplices. As such, the HGP product complex is imbued with transversal CZ gates stemming from a cohomology operation known as a cup product [GL24, BDET24]. We defer the interested reader to the paper by Breuckmann *et al.* [BDET24], and we will simply recite their relevant results for our purposes. The first step to defining the "copy-cup" CZ gate is to assign directionality to the edges, i.e. an

[7] e.g. an extractor [HCWY25]

[8] Recall that the affine class of gates on k (qu)bits is generated by all CNOTs and forms the group $\mathrm{GL}_k(\mathbb{F}_2)$.

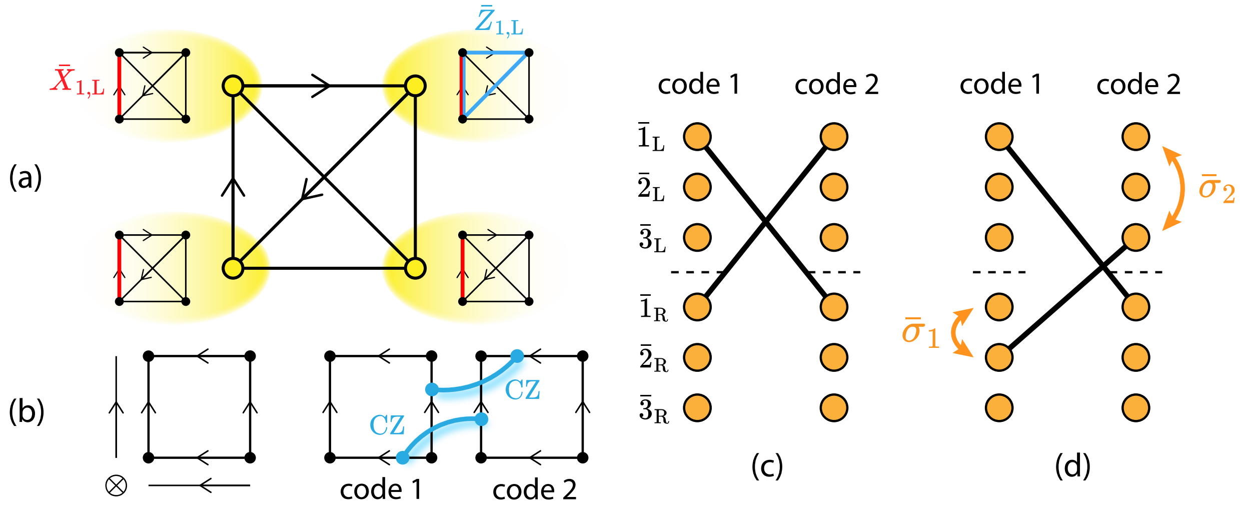

Figure 9: **(a)** One can view the K_4^2 graph as putting "inner" K_4 graphs, corresponding to one input K_4, on the vertices of an "outer" K_4, the other input, and connecting different inner graphs with transversal edges according to the edges of the outer graph. Logical operators for the first left logical qubit are drawn. An orientation satisfying the Leibniz condition along the support of (classical) codeword 1 is drawn on both inner and outer graphs. **(c)** Sketch of the copy-cup CZ gate on the data qubits (edges) resulting from a nonzero cup product. **(c)** The logical action of the copy-cup CZ gate is depicted. Orange circles depict logical qubits, and thick lines between logical qubits denote logical CZ gates. **(d)** With access to automorphism gadgets, the logical CZ gates from (c) can now address different logical qubits.

orientation, of the input K_4 graphs; we allow for edges to be "free" and have no direction. For a vertex $v \in V$, let $\delta(v) = \delta_{\mathrm{in}}(v) \cup \delta_{\mathrm{out}}(v) \cup \delta_{\mathrm{free}}(v) \in E$ denote the edges incident to v. The orientation needs to satisfy the "Leibniz condition": $|\delta_{\mathrm{in}}(v)| + |\delta_{\mathrm{out}}(v)| = 0 \pmod 2$ for all $v \in V$, in other words the parity of incoming and outgoing edges must be the same at every vertex. Upon choosing appropriate orientations for both our input K_4 graphs, we also induce an orientation on their Cartesian product K_4^2: whenever we copy an edge, we simply also copy its corresponding orientation; see Figure 9a for a sketch. The cup product is nonzero between any two directed edges on K_4^2 that satisfy

1. The two directed edges do not originate from the same input graph. Roughly, one edge stems from one input graph and the other edge from the other input graph.

2. The endpoint vertex of the first edge is the starting point of the second edge. In other words, the edges are consecutively oriented.

The copy-cup CZ gate between two code blocks is then constructed as follows. Whenever two edges satisfy the above conditions for a nonzero cup product, we apply a CZ gate between the first edge (according to the consecutive orientation) on the first code block and the second edge on the second code block; see Figure 9b for a sketch.

When we choose the orientations on the input K_4 graphs to follow our classical codewords (Figures 8 and 9a), the copy-cup CZ gate results in logical $\overline{\mathrm{CZ}}$ gates between logical qubits of different sectors; see Figure 9c. For example in Figure 9a, if we choose both input orientations to follow the support of logical bit 1 in the classical K_4 cycle code, then the copy-cup CZ gate performs logical $\overline{\mathrm{CZ}}$ between logical qubits $\bar{1}_\mathrm{L}$ and $\bar{1}_\mathrm{R}$ of two code blocks. Choosing the input orientations to follow different classical codewords results in $\overline{\mathrm{CZ}}$s that involve other logical qubits. Recall that we can perform arbitrary intrasector logical permutations using the automorphism gadgets lifted from the K_4 cycle code. Using these logical permutations, we can have the copy-cup CZ gates address different logical qubits, like in Figure 9d. This degree of addressability is greater than what we would simply get from the freedom of choosing the initial orientations.

7.3 $[\![288, 9, (16, 3)]\!]$ homological product code with $\mathcal{A} \simeq \mathrm{S}_4 \times \mathrm{S}_4 \times \mathrm{S}_4$ and transversal CCZ gates

We only briefly discuss our last example, since it is a simple extension of the previous one in Section 7.2. We take a (quantum × classical) homological product (18) of the $[\![48, 6, 3]\!]$ HGP code from the previous section with the transpose K_4 cycle code to obtain a $[\![288, 9, (d_X = 16, d_Z = 3)]\!]$ homological product code. Similar to the previous example, the structure of this code can be defined with respect to a graph, this time the triple Cartesian product of three K_4 graphs. X-checks are defined on the vertices, data qubits on the edges, and Z-checks on the faces (again spanned by edges originating from distinct K_4 copies). X-checks have weight 9, and each qubit participates in 2 of them. Z-checks have weight 4, and each qubit participates in 6 of them. The 9 logical qubits are partitioned into three sectors: left, middle and right, with 3 logical qubits belonging to each sector. We also have three types of automorphism gadgets (32): one from each input K_4 cycle code. Similar to the example in the previous section, the automorphism gadgets include arbitrary logical permutations within each sector.

The cup product formalism [BDET24] also applies to this code, and with an appropriate choice of orientations on the three input K_4 graphs, we can obtain transversal CCZ gates on our homological product code. Similar to Figure 9b, the three input orientations will induce an orientation on the product graph. For the copy-cup CCZ gate between three code blocks, we apply a CCZ gate between every three consecutive edges that originate from different input K_4 graphs: the first edge on the first code block, the second edge on the second code block, and the third edge on the third code block. On the logical level, the copy-cup CCZ gate will perform logical $\overline{\mathrm{CCZ}}$ gates on logical qubits residing in different sectors and in different code blocks: i.e. logical qubit $\bar{1}_{\mathrm{L}}$ on the first code block, $\bar{1}_{\mathrm{M}}$ on the second code block, and $\bar{1}_{\mathrm{R}}$ on the third code block, as well as their interblock permuted triples. Leveraging logical permutations from the automorphism gadgets, we can increase the total number of logical qubit triples that the copy-cup $\overline{\mathrm{CCZ}}$s act on and hence increase the addressability of the copy-cup gates.

Acknowledgments

We thank Victor Albert, Ali Fahimniya, Andrew Lucas and Shayan Majidy for helpful discussions. This material is based upon work supported in part by the Defense Advanced Research Projects Agency (DARPA) under Agreement HR00112490357, the NSF QLCI award OMA2120757 and the DoE ASCR Quantum Testbed Pathfinder program (awards No. DE-SC0019040 and No. DE-SC0024220). SJST acknowledges funding and support from Joint Center for Quantum Information and Computer Science (QuICS) Lanczos Graduate Fellowship, MathQuantum Graduate Fellowship, and the National University of Singapore (NUS) Development Grant.

References

[ABO08] Dorit Aharonov and Michael Ben-Or. Fault-tolerant quantum computation with constant error rate. *SIAM Journal on Computing*, 38(4):1207–1282, 2008.

[AGS15] Scott Aaronson, Daniel Grier, and Luke Schaeffer. The classification of reversible bit operations. *arXiv preprint arXiv:1504.05155*, 2015.

[ASY20] Emmanuel Abbe, Amir Shpilka, and Min Ye. Reed-muller codes: Theory and algorithms. *arXiv preprint arXiv:2002.03317*, 2020.

[Bab16] László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, page 684–697, New York, NY, USA, 2016. Association for Computing Machinery.

[BB24] Nikolas P. Breuckmann and Simon Burton. Fold-transversal clifford gates for quantum codes. *Quantum*, 8:1372, June 2024.

[BC06] Dave Bacon and Andrea Casaccino. Quantum error correcting subsystem codes from two classical linear codes. *arXiv preprint arXiv/quant-ph:0610088*, 2006.

[BCG$^+$24] Sergey Bravyi, Andrew W. Cross, Jay M. Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J. Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627(8005):778–782, March 2024.

[BDET24] Nikolas P. Breuckmann, Margarita Davydova, Jens N. Eberhardt, and Nathanan Tantivasadakarn. Cups and gates i: Cohomology invariants and logical quantum operations. *arXiv preprint arXiv:2410.16250*, 2024.

[BDSB24] Simon Burton, Elijah Durso-Sabina, and Natalie C. Brown. Genons, double covers and fault-tolerant clifford gates. *arXiv preprint arXiv:2406.09951*, 2024.

[BE21a] Nikolas P. Breuckmann and Jens N. Eberhardt. Balanced product quantum codes. *IEEE Transactions on Information Theory*, 67(10):6653–6674, October 2021.

[BE21b] Nikolas P. Breuckmann and Jens Niklas Eberhardt. Quantum low-density parity-check codes. *PRX Quantum*, 2(4), October 2021.

[BH14] Sergey Bravyi and Matthew B. Hastings. Homological product codes. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 273–282, New York, NY, USA, 2014. Association for Computing Machinery.

[BHM10] Sergey Bravyi, Matthew B. Hastings, and Spyridon Michalakis. Topological quantum order: Stability under local perturbations. *Journal of Mathematical Physics*, 51(9), September 2010.

[Big74] Norman Biggs. *Algebraic Graph Theory*. Cambridge Mathematical Library. Cambridge University Press, 2 edition, 1974.

[BMD06] H. Bombin and M. A. Martin-Delgado. Topological quantum distillation. *Physical Review Letters*, 97(18), October 2006.

[Bon84] Arrigo Bonisoli. Every equidistant linear code is a sequence of dual hamming codes. *Ars Combinatoria*, 18:181–186, 1984.

[BRC60a] R.C. Bose and D.K. Ray-Chaudhuri. Further results on error correcting binary group codes. *Information and Control*, 3(3):279–290, 1960.

[BRC60b] R.C. Bose and D.K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960.

[Cam19] Earl T Campbell. A theory of single-shot error correction for adversarial noise. *Quantum Science and Technology*, 4(2):025006, February 2019.

[CFD+24] Nathan Constantinides, Ali Fahimniya, Dhruv Devulapalli, Dolev Bluvstein, Michael J. Gullans, J. V. Porto, Andrew M. Childs, and Alexey V. Gorshkov. Optimal routing protocols for reconfigurable atom arrays. *arXiv preprint arXiv:2411.05061*, 2024.

[CHRY24] Andrew Cross, Zhiyang He, Patrick Rall, and Theodore Yoder. Improved qldpc surgery: Logical measurements and bridging codes. *arXiv preprint arXiv:2407.18393*, 2024.

[CR18a] Rui Chao and Ben W. Reichardt. Fault-tolerant quantum computation with few qubits. *npj Quantum Information*, 4(1), September 2018.

[CR18b] Rui Chao and Ben W. Reichardt. Quantum error correction with only two extra qubits. *Physical Review Letters*, 121(5), August 2018.

[CRSS97] A.R. Calderbank, E.M. Rains, P.W. Shor, and N.J.A. Sloane. Quantum error correction via codes over gf(4). In *Proceedings of IEEE International Symposium on Information Theory*, pages 292–, 1997.

[CS96] A. R. Calderbank and Peter W. Shor. Good quantum error-correcting codes exist. *Phys. Rev. A*, 54:1098–1105, Aug 1996.

[CZZ+24] Madelyn Cain, Chen Zhao, Hengyun Zhou, Nadine Meister, J. Pablo Bonilla Ataides, Arthur Jaffe, Dolev Bluvstein, and Mikhail D. Lukin. Correlated decoding of logical algorithms with transversal gates. *Phys. Rev. Lett.*, 133:240602, Dec 2024.

[DEL+22] Irit Dinur, Shai Evra, Ron Livne, Alexander Lubotzky, and Shahar Mozes. Locally testable codes with constant rate, distance, and locality. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 357–374, New York, NY, USA, 2022. Association for Computing Machinery.

[EK09] Bryan Eastin and Emanuel Knill. Restrictions on transversal encoded quantum gate sets. *Physical Review Letters*, 102(11), March 2009.

[EKZ22] Shai Evra, Tali Kaufman, and Gilles Zémor. Decodable quantum ldpc codes beyond the n distance barrier using high-dimensional expanders. *SIAM Journal on Computing*, 53(6):FOCS20–276, 2022.

[FB88] R.M. Foster and I.Z. Bouwer. *The Foster Census: R.M. Foster's Census of Connected Symmetric Trivalent Graphs.* Charles Babbage Research Centre, 1988.

[FH14] Michael H. Freedman and Matthew B. Hastings. Quantum systems on non-k-hyperfinite complexes: a generalization of classical statistical mechanics on expander graphs. *Quantum Info. Comput.*, 14(1–2):144–180, January 2014.

[FH21] Michael Freedman and Matthew Hastings. Building manifolds from quantum codes. *Geometric and Functional Analysis*, 31(4):855–894, Aug 2021.

[FZLL25] Xiaozhen Fu, Han Zheng, Zimu Li, and Zi-Wen Liu. No-go theorems for logical gates on product quantum codes. *arXiv preprint arXiv:2507.16797*, 2025.

[GG24] Louis Golowich and Venkatesan Guruswami. Quantum ldpc codes of almost linear distance via homological products. *arXiv preprint arXiv:2411.03646*, 2024.

[GJ25] Jérôme Guyot and Samuel Jaques. On the addressability problem on CSS codes. *arXiv preprint arXiv:2502.13889*, 2025.

[GKZ24] David Gamarnik, Bobak T. Kiani, and Alexander Zlokapa. Slow mixing of quantum gibbs samplers. *arXiv preprint arXiv:2411.04300*, 2024.

[GL24] Louis Golowich and Ting-Chun Lin. Quantum ldpc codes with transversal non-clifford gates via products of algebraic codes. *arXiv preprint arXiv:2410.14662*, 2024.

[Got97] Daniel Gottesman. Stabilizer codes and quantum error correction. *arXiv preprint arXiv:quant-ph/9705052*, 1997.

[Got14] Daniel Gottesman. Fault-tolerant quantum computation with constant overhead. *Quantum Info. Comput.*, 14(15–16):1338–1372, November 2014.

[GR13] Markus Grassl and Martin Roetteler. Leveraging automorphisms of quantum codes for fault-tolerant quantum computation. In *2013 IEEE International Symposium on Information Theory*, page 534–538. IEEE, 2013.

[GR24] Anqi Gong and Joseph M. Renes. Computation with quantum reed-muller codes and their mapping onto 2d atom arrays. *arXiv preprint arXiv:2410.23263*, 2024.

[Hao21] Hanson Hao. Investigations on automorphism groups of quantum stabilizer codes. *arXiv preprint arXiv:2109.12735*, 2021.

[HB68] S. Hakimi and J. Bredeson. Graph theoretic error-correcting codes. *IEEE Transactions on Information Theory*, 14(4):584–591, 1968.

[HBD17] Harald Andrés Helfgott, Jitendra Bajpai, and Daniele Dona. Graph isomorphisms in quasi-polynomial time. *arXiv preprint arXiv:1710.04574*, 2017.

[HCWY25] Zhiyang He, Alexander Cowtan, Dominic J. Williamson, and Theodore J. Yoder. Extractors: Qldpc architectures for efficient pauli-based computation. *arXiv preprint arXiv:2503.10390*, 2025.

[HDSHL24] Yifan Hong, Elijah Durso-Sabina, David Hayes, and Andrew Lucas. Entangling four logical qubits beyond break-even in a nonlocal code. *Phys. Rev. Lett.*, 133:180601, 2024.

[HFWH13] Charles D. Hill, Austin G. Fowler, David S. Wang, and Lloyd C. L. Hollenberg. Fault-tolerant quantum error correction code conversion. *Quantum Info. Comput.*, 13(5–6):439–451, May 2013.

[HGL25] Yifan Hong, Jinkang Guo, and Andrew Lucas. Quantum memory at nonzero temperature in a thermodynamically trivial system. *Nature Communications*, 16(1), January 2025.

[HKZ24] Po-Shen Hsin, Ryohei Kobayashi, and Guanyu Zhu. Classifying logical gates in quantum codes via cohomology operations and symmetry. *arXiv preprint arXiv:2411.15848*, 2024.

[HMKL24] Yifan Hong, Matteo Marinelli, Adam M. Kaufman, and Andrew Lucas. Long-range-enhanced surface codes. *Physical Review A*, 110(2), 2024.

[Hoc59] Alexis Hocquenghem. Codes correcteurs d'erreurs. *Chiffres (Paris)*, 2:147–156, 1959.

[Joy05] David Joyner. Guava: an error-correcting codes package. *SIGSAM Bull.*, 39(2):65–68, June 2005.

[Kas61] T. Kasami. A topological approach to construction of group codes. *J. Inst. Elec. Commun. Engrs.*, 44:1316–1321, 1961.

[Kit03] A.Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, January 2003.

[Kit06] Alexei Kitaev. Anyons in an exactly solved model and beyond. *Annals of Physics*, 321(1):2–111, January 2006.

[KLZ98] Emanuel Knill, Raymond Laflamme, and Wojciech H. Zurek. Resilient quantum computation. *Science*, 279(5349):342–345, 1998.

[KSWB25] Stergios Koutsioumpas, Hasan Sayginel, Mark Webster, and Dan E Browne. Automorphism ensemble decoding of quantum ldpc codes. *arXiv preprint arXiv:2503.01738*, 2025.

[LG17] Joshua Lockhart and Carlos E González Guillén. Quantum state isomorphism. *arXiv preprint arXiv:1709.09622*, 2017.

[LGAB24] Ali Lavasani, Michael J. Gullans, Victor V. Albert, and Maissam Barkeshli. On stability of k-local quantum phases of matter. *arXiv preprint arXiv:2405.19412*, 2024.

[Lit19] Daniel Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum*, 3:128, March 2019.

[LP24] Hsiang-Ku Lin and Leonid P. Pryadko. Quantum two-block group algebra codes. *Phys. Rev. A*, 109:022407, 2024.

[LPS88] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, September 1988.

[Mac64] Jessie Macwilliams. Permutation decoding of systematic codes. *The Bell System Technical Journal*, 43(1):485–505, 1964.

[MC25] Argyris Giannisis Manes and Jahan Claes. Distance-preserving stabilizer measurements in hypergraph product codes. *Quantum*, 9:1618, 2025.

[MGF$^+$25] Alexander J. Malcolm, Andrew N. Glaudell, Patricio Fuentes, Daryus Chandra, Alexis Schotte, Colby DeLisle, Rafael Haenel, Amir Ebrahimi, Joschka Roffe, Armanda O. Quintavalle, Stefanie J. Beale, Nicholas R. Lee-Hone, and Stephanie Simmons. Computing efficiently in qLDPC codes. *arXiv preprint arXiv:2502.07150*, 2025.

[MS77] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-correcting Codes*. Mathematical Library. North-Holland Publishing Company, 1977.

[Mul54] D. E. Muller. Application of boolean algebra to switching circuit design and to error detection. *Transactions of the I.R.E. Professional Group on Electronic Computers*, EC-3(3):6–12, 1954.

[NP24] Quynh T. Nguyen and Christopher A. Pattison. Quantum fault tolerance with constant-space and logarithmic-time overheads. *arXiv preprint arXiv:2411.03632*, 2024.

[PCV25] Michele Pacenti, Dimitris Chytas, and Bane Vasic. Construction and decoding of quantum margulis codes. *arXiv preprint arXiv:2503.03936*, 2025.

[PK21a] Pavel Panteleev and Gleb Kalachev. Degenerate quantum LDPC codes with good finite length performance. *Quantum*, 5:585, 2021.

[PK21b] Pavel Panteleev and Gleb Kalachev. Quantum LDPC codes with almost linear minimum distance. *IEEE Transactions on Information Theory*, 68(1):213–229, 2021.

[PK22] Pavel Panteleev and Gleb Kalachev. Asymptotically good quantum and locally testable classical LDPC codes. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 375–388, New York, NY, USA, 2022. Association for Computing Machinery.

[PR97] E. Petrank and R.M. Roth. Is code equivalence easy to decide? *IEEE Transactions on Information Theory*, 43(5):1602–1604, 1997.

[PRBK24] Benedikt Placke, Tibor Rakovszky, Nikolas P. Breuckmann, and Vedika Khemani. Topological quantum spin glass order and its realization in qldpc codes. *arXiv preprint arXiv:2412.13248*, 2024.

[QC22] Armanda O. Quintavalle and Earl T. Campbell. Reshape: A decoder for hypergraph product codes. *IEEE Transactions on Information Theory*, 68(10):6569–6584, 2022.

[QWV23] Armanda O. Quintavalle, Paul Webster, and Michael Vasmer. Partitioning qubits in hypergraph product codes to implement logical gates. *Quantum*, 7:1153, 2023.

[RAC$^+$24] Ben W. Reichardt, David Aasen, Rui Chao, Alex Chernoguzov, Wim van Dam, John P. Gaebler, Dan Gresh, Dominic Lucchetti, Michael Mills, Steven A. Moses, Brian Neyenhuis, Adam Paetznick, Andres Paz, Peter E. Siegfried, Marcus P. da Silva, Krysta M. Svore, Zhenghan Wang, and Matt Zanner. Demonstration of quantum computation and error correction with a tesseract code. *arXiv preprint arXiv:2409.04628*, 2024.

[RK23] Tibor Rakovszky and Vedika Khemani. The physics of (good) ldpc codes i. gauging and dualities. *arXiv preprint arXiv:2310.16032*, 2023.

[RK24] Tibor Rakovszky and Vedika Khemani. The physics of (good) ldpc codes ii. product constructions. *arXiv preprint arXiv:2402.16831*, 2024.

[RKL$^+$24] Wojciech De Roeck, Vedika Khemani, Yaodong Li, Nicholas O'Dea, and Tibor Rakovszky. Ldpc stabilizer codes as gapped quantum phases: stability under graph-local perturbations. *arXiv preprint arXiv:2411.02384*, 2024.

[SJOY25] Esha Swaroop, Tomas Jochym-O'Connor, and Theodore J. Yoder. Universal adapters between quantum ldpc codes. *arXiv preprint arXiv:2410.03628*, 2025.

[SKW$^+$25] Hasan Sayginel, Stergios Koutsioumpas, Mark Webster, Abhishek Rajput, and Dan E Browne. Fault-tolerant logical clifford gates from code automorphisms. *arXiv preprint arXiv:2409.18175*, 2025.

[Ste96] A. M. Steane. Error correcting codes in quantum theory. *Phys. Rev. Lett.*, 77:793–797, Jul 1996.

[TKY24] Shiro Tamiya, Masato Koashi, and Hayata Yamasaki. Polylog-time- and constant-space-overhead fault-tolerant quantum computation with quantum low-density parity-check codes. *arXiv preprint arXiv:2411.03683*, 2024.

[TS24] Shi Jie Samuel Tan and Lev Stambler. Effective distance of higher dimensional HGPs and weight-reduced quantum LDPC codes. *arXiv preprint arXiv:2409.02193*, 2024.

[Tut47] W. T. Tutte. A family of cubical graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 43(4):459–474, 1947.

[Tut66] William Thomas Tutte. Connectivity in graphs. In *Connectivity in Graphs*. University of Toronto press, 1966.

[TZ14] Jean-Pierre Tillich and Gilles Zemor. Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60(2):1193–1202, 2014.

[VB19] Michael Vasmer and Dan E. Browne. Three-dimensional surface codes: Transversal gates and fault-tolerant architectures. *Phys. Rev. A*, 100:012312, Jul 2019.

[WVSB22] Paul Webster, Michael Vasmer, Thomas R. Scruby, and Stephen D. Bartlett. Universal fault-tolerant quantum computing with stabilizer codes. *Phys. Rev. Res.*, 4:013092, Feb 2022.

[XBAP$^+$24] Qian Xu, J. Pablo Bonilla Ataides, Christopher A. Pattison, Nithin Raveendran, Dolev Bluvstein, Jonathan Wurtz, Bane Vasić, Mikhail D. Lukin, Liang Jiang, and Hengyun Zhou. Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays. *Nature Physics*, 20(7):1084–1090, 2024.

[XZZ$^+$24] Qian Xu, Hengyun Zhou, Guo Zheng, Dolev Bluvstein, J. Pablo Bonilla Ataides, Mikhail D. Lukin, and Liang Jiang. Fast and parallelizable logical computation with homological product codes. *arXiv preprint arXiv:2407.18490*, 2024.

[YL24] Chao Yin and Andrew Lucas. Low-density parity-check codes as stable phases of quantum matter. *arXiv preprint arXiv:2411.01002*, 2024.

[YTC16] Theodore J. Yoder, Ryuji Takagi, and Isaac L. Chuang. Universal fault-tolerant gates on concatenated stabilizer codes. *Phys. Rev. X*, 6:031039, Sep 2016.

[ZP19] Weilei Zeng and Leonid P. Pryadko. Higher-dimensional quantum hypergraph-product codes with finite rates. *Phys. Rev. Lett.*, 122:230501, Jun 2019.

[ZP20] Weilei Zeng and Leonid P. Pryadko. Minimal distances for certain quantum product codes and tensor products of chain complexes. *Physical Review A*, 102(6), December 2020.

[ZSP$^+$24] Guanyu Zhu, Shehryar Sikander, Elia Portnoy, Andrew W. Cross, and Benjamin J. Brown. Non-clifford and parallelizable fault-tolerant logical gates on constant and almost-constant rate homological quantum ldpc codes via higher symmetries. *arXiv preprint arXiv:2310.16982*, 2024.