# Phantom codes: Entangling logical qubits without physical operations

Jin Ming Koh,[1] Anqi Gong,[2] Andrei C. Diaconu,[1] Daniel Bochen Tan,[1] Alexandra A. Geim,[1] Michael J. Gullans,[3] Norman Y. Yao,[1] Mikhail D. Lukin,[1] and Shayan Majidy[1]

[1] *Department of Physics, Harvard University, Cambridge, MA 02138, USA*
[2] *Institute for Theoretical Physics, ETH Zürich, 8093 Zürich, Switzerland*
[3] *QuICS, University of Maryland, College Park, MD 20742, USA; and NIST, Gaithersburg, MD 20899, USA*

**Article:** arXiv.2601.20927

## Motivation

A central goal of current quantum error correction (QEC) research is to reduce overheads and enable efficient computation. Much recent work focuses on high-rate quantum low-density parity-check (qLDPC) codes that maximize the number of logical qubits per codeblock under sparsity constraints [1]. While this yields compact quantum memories, it does not directly constrain the *cost of computation*, which in practice is often dominated by addressable logical operations [2–6]. This motivates considering both storage and computation to effectively reduce overhead.

We pursue a complementary approach that places logical operations at the centre of the design. Starting from a target set of logical operations—here, logical entangling gates—we ask which codes support them with minimal overhead. In conventional codes, such gates require repeated measurement, dense two-qubit gate layers, or ancillary codeblocks [4–7]. Recent state-of-the-art experiments show that logical entangling gates occupy a substantial fraction of the total error budget [8, 9]. This suggests optimizing fault-tolerant computation at the level of the full stack.

Towards this goal, we examine the fundamental lower bound on the cost of a logical entangling gate. For certain codes, such gates can be implemented solely via permutations of physical qubits, which generate no new physical entanglement but instead redistribute existing correlations. When absorbed into circuit compilation rather than executed on hardware, these permutations incur zero overhead and achieve perfect fidelity (Fig. 1).

Motivated by this observation, we define a new class of QEC codes, termed *phantom codes*: stabilizer codes in which logical entangling gates between *any* ordered pair of logical qubits are realized *solely* by physical-qubit permutations, enabling all in-block logical entangling gates to disappear after classical compilation (Fig. 1). Among existing QEC codes, only two satisfy this definition: the $[\![12, 2, 4]\!]$ Carbon code [10] and the $[\![2^D, D, 2]\!]$ hypercube codes [11, 12]. Despite these examples, phantom codes remain largely unexplored: it is unknown whether additional families exist, what structural constraints they obey, or whether their advantages persist at scale under realistic noise.

## Phantom codes

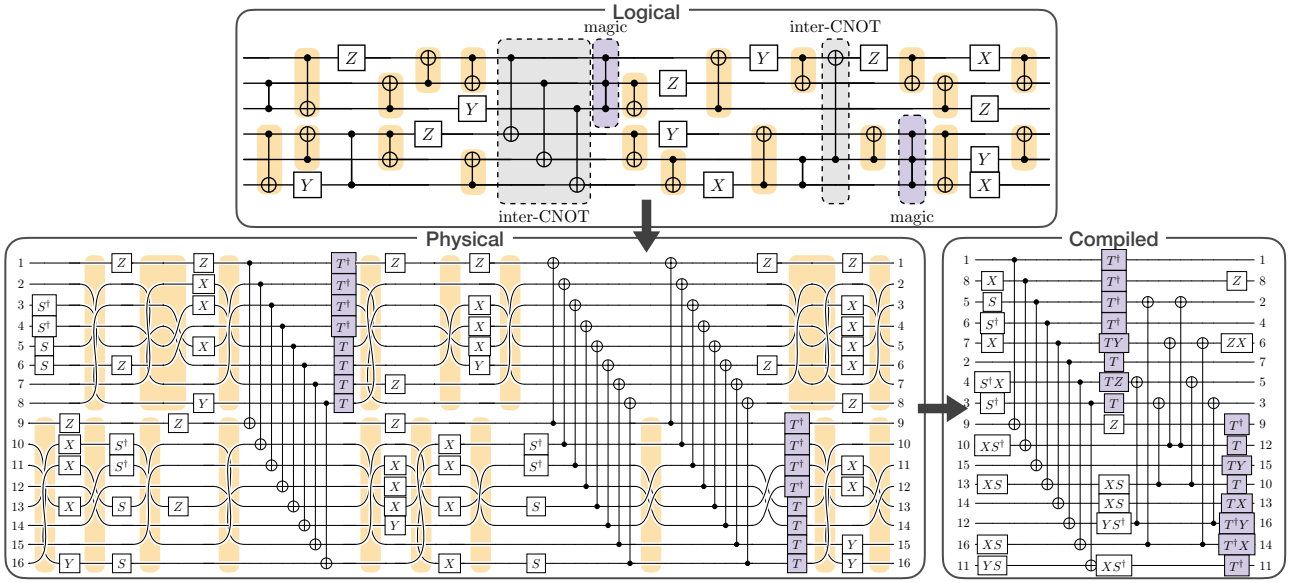



Figure 1. **Illustration of phantom codes.** Qubit permutations can be pulled through arbitrary circuits without operator spreading and absorbed into relabelling at the input or output.

We focus on CSS codes because they guarantee a transversal interblock $\overline{\text{CNOT}}$ required for connecting codeblocks. Consequently, the only in-block logical entangling gate realizable via qubit permutation is $\overline{\text{CNOT}}$. Thus, an $[\![n, k, d]\!]$ CSS code is phantom if the $\overline{\text{CNOT}}_{ab}$ gate for every pair of distinct logical qubits $(a, b) \in [k]^2$ can be implemented via qubit permutations, for some choice of logical basis. Phantom codes differ from three related ideas:

1. *Codes with some permutation symmetries.* Although many stabilizer codes admit permutation symmetries, phantom codes require individually addressable $\overline{\text{CNOT}}$s. This excludes codes such as $[\![2^{2r}, \binom{2r}{r}, 2^r]\!]$ quantum Reed–Muller codes ($r > 1$) and bivariate bicycle codes [4, 13], which permit only particular products of $\overline{\text{CNOT}}$s.

2. *Logical Pauli-frame tracking.* While Clifford logical circuits allow $\overline{\text{CNOT}}$s to be classically tracked (by the Gottesman–Knill theorem), phantom codes eliminate all in-block $\overline{\text{CNOT}}$s even when entangling operations are interleaved with non-Clifford (magic) gates (Fig. 1), yielding zero-cost all-to-all connectivity within a codeblock.

3. *Automorphism gates.* Phantom codes exclude constructions combining permutations with local physical Clifford gates [14, 15]. Zero-overhead, perfect-fidelity entangling gates arise only from pure permutations, which uniquely commute through arbitrary circuits without operator spread.

**Main result 1: Code discovery & construction**
We identify $>10^5$ phantom codes and infinite families via a combination of numerical and analytical constructions:

1. *Exhaustive enumeration.* We systematically identify phantom codes by exhaustively enumerating all inequivalent CSS codes of block length $n \leq 14$ and certifying phantomness via Boolean constraint satisfaction (SAT). Extending prior techniques for $n \leq 9$ [16], we generate codes by iteratively enlarging stabilizer groups and classify them up to qubit permutations and global Hadamards. CSS-specific simplifications and canonical graph labelling [17] sharply reduce both the search space and equivalence-testing cost. This yields a complete catalogue of $2.71 \times 10^{10}$ inequivalent CSS codes, the largest stabilizer-code dataset assembled to date.

2. *SAT-based search.* To extend the search beyond $n = 14$ and determine minimal block lengths for given $(k, d)$, we repurpose the phantomness SAT check as an automated code-discovery tool. By treating stabilizer generators as free variables, the solver constructs codes satisfying both permutation-only logical-gate constraints and distance-$d$ requirements. This yields phantom codes up to $n = 21$ that have the best-known parameters or gate sets.

3. *Quantum Reed–Muller codes.* To construct phantom codes at larger $k$, we build on quantum Reed–Muller (qRM) codes. While standard qRM codes support products of permutation-implemented $\overline{\text{CNOT}}$s, they are not phantom. We obtain phantom qRM codes by promoting selected logical operators to stabilizers, yielding parameters $[\![2^m, m - l + 1, \min(2^{m-l}, 2^l)]\!]$. In addition to phantomness, these codes efficiently support fault-tolerant state preparation, the full logical Clifford group, and a transversal magic-gate scheme via temporary projection into hypercube codes.

4. *Binarization and concatenation scheme.* We introduce a scheme based on high-distance self-dual GF(4) qudit codes. Each GF(4) symbol is first binarized into a qubit pair and then concatenated with the $[\![4, 2, 2]\!]$ phantom code, which produces a phantom code. The resulting qubit codes inherit large distances from the parent qudit codes as well as other transversal and fold-type logical gates.

Beyond permutation-implemented $\overline{\text{CNOT}}$ gates, phantom codes admit additional logical Clifford and non-Clifford operations that we identify via three mechanisms: (i) automorphism Cliffords from stabilizer symmetries, realized as qubit permutations and single-qubit physical Clifford gates; (ii) logical diagonal gates from non-uniform single-qubit $Z$-basis rotations; and (iii) fold-type diagonal gates identified by embedding the code into a larger one, inducing logical operations via patterned one- and two-qubit diagonal interactions that can sometimes preserve distance.

**Main result 2: End-to-end benchmarking**
Second, through end-to-end noisy simulations with state preparation, full QEC cycles, and realistic physical error rates, we demonstrate scalable advantages of phantom codes over the surface code across multiple tasks. We observe a one-to-two order-of-magnitude reduction in logical infidelity at comparable qubit overhead for logical GHZ-state preparation and Trotterized many-body simulation tasks, given a modest preselection acceptance rate (24% to 49%).

- *Benchmarking context.* As a strong reference, we use a fully optimized surface-code architecture with correlated decoding and algorithmic fault tolerance, which accounts for inter-block error propagation and reduces the number of syndrome-extraction rounds per transversal $\overline{\text{CNOT}}$ from $\mathcal{O}(d)$ to $\mathcal{O}(1)$.

  Operating the non-LDPC phantom code requires specialized protocols: Steane-style error correction; a short-depth fault-tolerant state-preparation scheme exploiting error detection and preselection; and spatiotemporal sliding-window correlated decoders that track error propagation through transversal gates and Steane gadgets while keeping decoding cost linear in circuit size. We use a gauge-fixing for the phantom qRM code so that $X$- and $Z$-stabilizer ranks are balanced, and a circuit-level noise model calibrated to recent neutral-atom experiments.

- *GHZ state preparation.* We benchmark phantom codes in a multi-codeblock setting by preparing logical GHZ states across $K = 4$–$64$ logical qubits, comparing against surface codes matched by physical-qubit footprint or code distance. As $K$ increases, the ratio of in-block permutation to interblock transversal $\overline{\text{CNOT}}$s drops from $3:0$ to $3:61$. This benchmark requires active Steane QEC and mixed-basis ($|\overline{+00}\rangle$) state-preparation, which we achieve fault-tolerantly through a low-depth teleportation-based protocol, on the phantom code.

  Despite an increasing number of interblock operations, the $[\![64, 4, 8]\!]$ phantom code retains a strong advantage: at $K = 64$ it yields a $\sim$56$\times$ reduction in logical infidelity versus a footprint-matched $d = 6$ surface code, and a $\sim$4$\times$ improvement over $d = 8$ while using $\sim$1.8$\times$ fewer physical qubits. This advantages persists at lower error rates.

- *Trotterized many-body simulation.* We simulate Trotterized time evolution under eight-body Ising interactions and a transverse field, with Trotter blocks decomposed into logarithmic-depth $\overline{\text{CNOT}}$ ladders interleaved with single-logical-qubit rotations. For $K = 8$–$64$ logical qubits and eight Trotter steps, the largest instance exceeds 2400 logical gates with two-qubit logical depth 96.

  With a physical footprint comparable to the surface code, the $[\![64, 4, 8]\!]$ phantom code consistently outperforms surface codes at both near-term and projected error rates, achieving a $\sim$94$\times$ ($\sim$10$\times$) reduction in logical infidelity relative to $d = 6$ ($d = 8$) surface codes under sliding-window MLE decoding. This advantage persists with faster but less accurate decoding or relaxed preselection, supporting large-scale many-body quantum simulation.

**Implications**

This work carries several implications for fault-tolerance quantum computing.

- *Experimental opportunities with phantom codes.* Phantom codes reduce logical error rates and overhead for circuits with dense local entanglement, bringing fermionic simulation and correlated-phase state preparation closer to practical realization. Combined with recent reductions in digital fermionic simulation overhead [18, 19], these gains substantially improve near-term feasibility.

  Beyond quantum simulation, phantom codes enable highly entangled logical states relevant to measurement-based quantum computation [20] and quantum communication [21]; demonstrating such states with near-zero physical entangling-gate cost would provide a clear experimental validation of the phantom-code paradigm.

  These benefits are accessible on platforms with long-range connectivity, notably neutral atoms and trapped ions. Neutral-atom systems support highly parallel transversal logical entangling gates, whereas trapped-ion platforms do not, making the impact of phantom codes especially pronounced for trapped-ion processors.

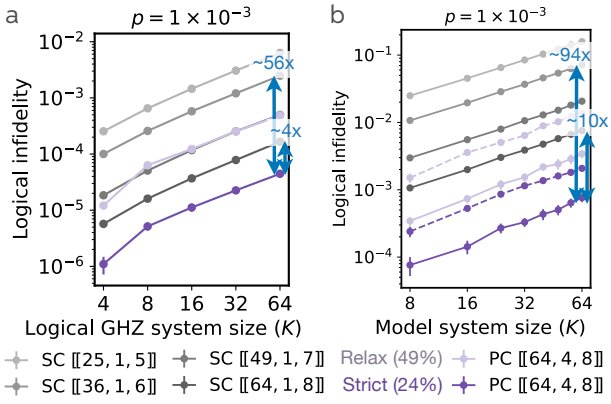



Figure 2. **Benchmarking phantom against surface codes.** **(a)** Logical GHZ preparation across multiple codeblocks. **(b)** Trotterized many-body time evolution over eight Trotter steps. In (a)–(b), dark (light) purple indicate strict (relaxed) phantom-code preselection; in (b), solid (dashed) lines denote sliding-window MLE (list) correlated decoding. Surface code baselines are $d = 5$–$8$.

- *A design continuum for codes.* Practical fault tolerance requires QEC codes that balance storage efficiency with efficient logical computation. High-rate qLDPC codes optimize qubit overhead, state preparation, and decoding, while phantom codes occupy the opposite extreme by eliminating the overhead of in-block logical entangling gates, yet remaining practically competitive despite lower encoding rates. This contrast suggests a broader design continuum between large-automorphism, low-rate codes and high-rate qLDPC constructions that offer limited logical operations, with resource-optimal operating points potentially lying between these extremes.

  The resources developed here enable systematic exploration of this design space. These include a complete database of CSS codes up to $n = 14$, a SAT-based automated code discovery method respecting specified logical gate structure, and a pipeline for identifying logical operations on a code. By relaxing the phantom constraint, the same framework probes trade-offs among encoding rate, code distance, stabilizer weight, and gate sets, enabling the identification of codes tailored to specific hardware connectivities or alternative objectives such as the cost of logical magic.

- *Practical tools for non-LDPC codes.* Our benchmarks motivate re-evaluating which QEC codes are viable for specific applications. With appropriate decoding and state-preparation protocols, non-LDPC codes can outperform LDPC codes on tailored tasks despite the latter's structural advantages. The tools we develop—covering decoding and fault-tolerant state preparation—apply broadly to non-LDPC codes beyond phantom constructions.

## REFERENCES


[1] N. P. Breuckmann and J. N. Eberhardt, Quantum low-density parity-check codes, PRX Quantum **2**, 040101 (2021).
[2] L. Z. Cohen, I. H. Kim, S. D. Bartlett, and B. J. Brown, Low-overhead fault-tolerant quantum computing using long-range connectivity, Science Advances **8**, eabn1717 (2022).
[3] T. J. Yoder, E. Schoute, P. Rall, E. Pritchett, J. M. Gambetta, A. W. Cross, M. Carroll, and M. E. Beverland, Tour de gross: A modular quantum computer based on bivariate bicycle codes (2025), arXiv:2506.03094 [quant-ph].
[4] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, High-threshold and low-overhead fault-tolerant quantum memory, Nature **627**, 778 (2024).
[5] Q. Xu, H. Zhou, G. Zheng, D. Bluvstein, J. P. B. Ataides, M. D. Lukin, and L. Jiang, Fast and parallelizable logical computation with homological product codes, Phys. Rev. X **15**, 021065 (2025).
[6] Q. Xu, J. P. Bonilla Ataides, C. A. Pattison, N. Raveendran, D. Bluvstein, J. Wurtz, B. Vasić, M. D. Lukin, L. Jiang, and H. Zhou, Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays, Nature Physics **20**, 1084 (2024).
[7] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Surface codes: Towards practical large-scale quantum computation, Phys. Rev. A **86**, 032324 (2012).
[8] D. Bluvstein, A. A. Geim, S. H. Li, S. J. Evered, J. P. Bonilla Ataides, G. Baranes, A. Gu, T. Manovitz, M. Xu, M. Kalinowski, S. Majidy, C. Kokail, N. Maskara, E. C. Trapp, L. M. Stewart, S. Hollerith, H. Zhou, M. J. Gullans, S. F. Yelin, M. Greiner, V. Vuletić, M. Cain, and M. D. Lukin, A fault-tolerant neutral-atom architecture for universal quantum computation, Nature **649**, 39– (2025).
[9] N. Lacroix, A. Bourassa, F. J. Heras, L. M. Zhang, J. Bausch, A. W. Senior, T. Edlich, N. Shutty, V. Sivak, A. Bengtsson, *et al.*, Scaling and logic in the color code on a superconducting quantum processor, Nature **645**, 614 (2025).
[10] A. Paetznick, M. P. da Silva, C. Ryan-Anderson, J. M. Bello-Rivas, J. P. C. III, A. Chernoguzov, J. M. Dreiling, C. Foltz, F. Frachon, J. P. Gaebler, T. M. Gatterman, L. Grans-Samuelsson, D. Gresh, D. Hayes, N. Hewitt, C. Holliman, C. V. Horst, J. Johansen, D. Lucchetti, Y. Matsuoka, M. Mills, S. A. Moses, B. Neyenhuis, A. Paz, J. Pino, P. Siegfried, A. Sundaram, D. Tom, S. J. Wernli, M. Zanner, R. P. Stutz, and K. M. Svore, Demonstration of logical qubits and repeated error correction with better-than-physical error rates (2024), arXiv:2404.02280 [quant-ph].
[11] M. Vasmer and A. Kubica, Morphing quantum codes, PRX Quantum **3**, 030319 (2022).
[12] D. Hangleiter, M. Kalinowski, D. Bluvstein, M. Cain, N. Maskara, X. Gao, A. Kubica, M. D. Lukin, and M. J. Gullans, Fault-tolerant compiling of classically hard instantaneous quantum polynomial circuits on hypercubes, PRX Quantum **6**, 020338 (2025).
[13] A. Gong and J. M. Renes, Computation with quantum reed–muller codes and their mapping onto 2D atom arrays (2024), arXiv:2410.23263 [quant-ph].
[14] H. Sayginel, S. Koutsioumpas, M. Webster, A. Rajput, and D. E. Browne, Fault-tolerant logical clifford gates from code automorphisms, PRX Quantum **6**, 030343 (2025).
[15] A. J. Malcolm, A. N. Glaudell, P. Fuentes, D. Chandra, A. Schotte, C. DeLisle, R. Haenel, A. Ebrahimi, J. Roffe, A. O. Quintavalle, S. J. Beale, N. R. Lee-Hone, and S. Simmons, Computing efficiently in QLDPC codes (2025), arXiv:2502.07150 [quant-ph].
[16] A. Cross and D. Vandeth, Small binary stabilizer subsystem codes (2025), arXiv:2501.17447 [quant-ph].
[17] B. D. McKay and A. Piperno, Practical graph isomorphism, ii, Journal of symbolic computation **60**, 94 (2014).
[18] N. Maskara, M. Kalinowski, D. Gonzalez-Cuadra, and M. D. Lukin, Fast simulation of fermions with reconfigurable qubits (2025), arXiv:2509.08898 [quant-ph].
[19] N. Constantinides, J. Yu, D. Devulapalli, A. Fahimniya, L. Schaeffer, A. M. Childs, M. J. Gullans, A. Schuckert, and A. V. Gorshkov, Low-depth fermion routing without ancillas (2025), arXiv:2510.05099 [quant-ph].
[20] H. J. Briegel, D. E. Browne, W. Dür, R. Raussendorf, and M. Van den Nest, Measurement-based quantum computation, Nature Physics **5**, 19 (2009).
[21] S. Majidy, D. Hangleiter, and M. J. Gullans, Scalable and fault-tolerant preparation of encoded $k$-uniform states, Phys. Rev. A **112**, 042409 (2025).

# Entangling logical qubits without physical operations

Jin Ming Koh,[1] Anqi Gong,[2] Andrei C. Diaconu,[1] Daniel Bochen Tan,[1] Alexandra A. Geim,[1] Michael J. Gullans,[3,4] Norman Y. Yao,[1] Mikhail D. Lukin,[1] and Shayan Majidy[1,*]

[1] *Department of Physics, Harvard University, Cambridge, Massachusetts 02138, USA*
[2] *Institute for Theoretical Physics, ETH Zürich, 8093 Zürich, Switzerland*
[3] *Joint Center for Quantum Information and Computer Science, University of Maryland, College Park, Maryland 20742, USA*
[4] *National Institute of Standards and Technology, Gaithersburg, MD 20899, USA*

Fault-tolerant logical entangling gates are essential for scalable quantum computing, but are limited by the error rates and overheads of physical two-qubit gates and measurements. To address this limitation we introduce *phantom codes*—quantum error-correcting codes that realize entangling gates between all logical qubits in a codeblock purely through relabelling of physical qubits during compilation, yielding perfect fidelity with no spatial or temporal overhead. We present a systematic study of such codes. First, we identify phantom codes using complementary numerical and analytical approaches. We exhaustively enumerate all $2.71 \times 10^{10}$ inequivalent CSS codes up to $n = 14$ and identify additional instances up to $n = 21$ via SAT-based methods. We then construct higher-distance phantom-code families using quantum Reed–Muller codes and the binarization of qudit codes. Across all identified codes, we characterize other supported fault-tolerant logical Clifford and non-Clifford operations. Second, through end-to-end noisy simulations with state preparation, full QEC cycles, and realistic physical error rates, we demonstrate scalable advantages of phantom codes over the surface code across multiple tasks. We observe one–to–two–order-of-magnitude reduction in logical infidelity at comparable qubit overhead for GHZ-state preparation and Trotterized many-body simulation tasks, given a modest preselection acceptance rate. Our work establishes phantom codes as a viable architectural route to fault-tolerant quantum computation with scalable benefits for workloads with dense local entangling structure, and introduces general tools for systematically exploring the broader landscape of quantum error-correcting codes.

## I. INTRODUCTION

To realize their potential, quantum computers must run deep circuits reliably in the presence of noise [1]. Achieving this at large scales requires the use of quantum error correction (QEC) [2]. The last several years have seen remarkable progress in QEC across a broad landscape of platforms ranging from neutral-atom and superconducting qubits to trapped-ions and bosonic systems [3–9]. Further progress hinges on reducing QEC overheads and enabling efficient computation with suppressed logical error rates.

Recent efforts to reduce QEC overhead focus on encoding as many logical qubits as possible within a codeblock while enforcing sparsity structure, namely in high-rate quantum low-density parity-check (qLDPC) codes [10]. While this design choice yields compact quantum memories, they do not directly constrain the cost of computation. In practice, addressable logical operations dominate space–time overheads and error budgets in such codes [11–15]. These observations suggest that reducing overhead requires jointly optimizing storage and computation, rather than treating logical gates as a downstream constraint. In this work, we pursue a complementary approach that places logical operations at the center of the design. We begin with a set of logical operations—in this case, targeted logical entangling gates—and ask which codes support them with minimal overhead. In conventional codes, such gates typically require repeated measurement rounds, dense layers of two-qubit gates, or ancillary codeblocks [13–16]. As observed in recent state-of-the-art experiments, logical entangling gates account for a substantial fraction of the total error budget [3, 4]. This raises the question of how fault-tolerant quantum computation can be optimized as a whole, encompassing both logical operations and QEC components.

As a step towards this optimization, we explore the fundamental lower bound on the cost of a logical entangling gate. Interestingly, for certain codes, logical entangling gates can be realized solely via the permutation of physical qubits. This permutation generates no new physical entanglement, but instead redistributes existing correlations among subsystems. Crucially, such an approach can incur zero overhead and exhibits perfect fidelity if the permutations are directly absorbed into circuit compilation rather than executed on hardware (Fig. 1).

Motivated by this observation, we designate a new class of QEC codes termed *phantom codes*: stabilizer codes in which logical entangling gates between every ordered pair of logical qubits can be implemented solely via physical qubit permutations. In such codes, all in-block logical entangling gates vanish from the physical circuit after classical circuit compilation (Fig. 2)—hence the name "phantom". Of the existing QEC codes, only two satisfy this definition: the $[\![12, 2, 4]\!]$ Carbon code [7] and $[\![2^D, D, 2]\!]$ hypercube codes [17, 18]. The former enabled

* smajidy@fas.harvard.edu

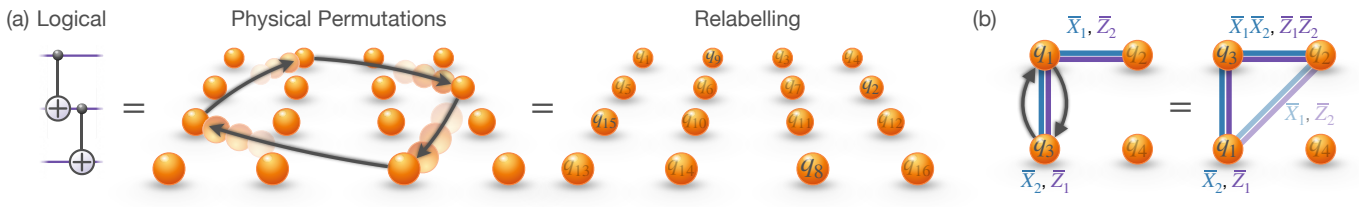



Figure 1. **Illustration of logical entanglement via relabelling. (a)** In phantom codes, logical entangling gates are realized by physical qubit permutations that are absorbed at the compilation stage as a relabelling, without performing any physical qubit permutation. **(b)** The $[\![4, 2, 2]\!]$ code is the smallest phantom code. It has logical operators $\overline{X}_1 = XXII$, $\overline{X}_2 = XIXI$, $\overline{Z}_1 = ZIZI$, $\overline{Z}_2 = ZZII$. Permuting qubits 1 and 3 (1 and 2) implements $\overline{\text{CNOT}}_{12}$ ($\overline{\text{CNOT}}_{21}$, not shown).

repeated rounds of QEC in trapped-ion experiments [7], while the latter underpinned recent neutral-atom experiments achieving logical-over-physical performance gains in quantum circuit sampling [5]. Despite intriguing properties and recent experimental success, phantom codes remain largely unexplored. For example, it is unknown whether additional classes of phantom codes exist, what structural constraints they obey, and whether their architectural advantages persist for scalable applications under realistic noise conditions.

We present a systematic and extensive investigation of phantom codes, exploring both their key features and limitations. Our main results are two fold. First, via a combination of numerics and analytic construction, we substantially expand the phantom-code landscape, from a single known error-correcting code and a single error-detecting code family, to over a hundred thousand new phantom code instances and multiple error-correcting families. Second, we demonstrate the practical advantages of phantom codes for applications in the presence of realistic noise. In particular, we benchmark a representative $[\![64, 4, 8]\!]$ phantom code against surface-code baselines through end-to-end noisy simulations including both state preparation and full QEC cycles. Using realistic physical error rates, we investigate the logical fidelity of entangled state preparation (i.e. of a GHZ state) and Trotterized many-body simulation; in both scenarios, we demonstrate one–to–two–order-of-magnitude improvements in the logical fidelity using a nearly identical number of physical qubits (and a preselection acceptance rate $\sim$24%), over a range of 8 to 64 logical qubits. Moreover, in addition to the logical entangling gates generated via recompilation, for all of the phantom codes discovered, we also identify fault-tolerant logical Clifford and non-Clifford gates. Finally, we develop practical tools for implementing non-LDPC phantom codes that also apply to other non-LDPC codes, including improved decoding and fault-tolerant state preparation.

## II. PHANTOM CODES

We begin by formally defining phantom codes and establishing their key structural properties. We then summarize our main results, explaining the implications of phantom codes for fault-tolerant design and of our methods for systematic QEC code discovery.

### A. Definitions and key properties

We focus on CSS stabilizer codes because scalable fault-tolerant architectures require a reliable interblock entangling mechanism, and the CSS structure guarantees access to a transversal interblock $\overline{\text{CNOT}}$ gate. On CSS codes, we assume $X$- ($Z$-) logical operators consist only of $X$- ($Z$-) physical operators. In this setting, the only nontrivial in-block logical entangling gate realizable via qubit permutation alone is a $\overline{\text{CNOT}}$.

**Definition 1** (CSS phantom codes)**.** *An $[\![n, k, d]\!]$ CSS code is phantom if the $\overline{\text{CNOT}}_{ab}$ gate for every ordered pair of logical qubits $(a, b) \in [k]^2$ can be implemented via qubit permutations, for some choice of logical basis.*

Definition 1 introduces a new class of codes. To help contextualize this notion, we clarify how phantom codes differ from three related ideas. First, although many stabilizer codes admit permutation symmetries, phantom codes satisfy a substantially stronger requirement. For example, the class excludes codes such as the $[\![2^{2r}, \binom{2r}{r}, 2^r]\!]$ qRM codes for $r > 1$ and bivariate bicycle codes [13, 19], which support only *products* of logical $\overline{\text{CNOT}}$ gates via permutations. Because these products cannot be composed to produce individually addressable $\overline{\text{CNOT}}$s, such codes do not qualify as phantom codes. Phantom codes are also distinct from permutation-invariant (PI) codes, whose codewords are invariant under all qubit permutations [20, 21].[1]

Second, the operation of phantom codes cannot simply be understood as a form of logical Pauli frame tracking. While logical Clifford circuits permit all $\overline{\text{CNOT}}$s to be absorbed into classical tracking of the Pauli frame, phantom codes allow entangling operations to be interleaved

[1] Most PI codes encode a single logical qubit ($k = 1$) and are non-additive; no known PI code satisfies the phantom criterion, and none of the phantom codes identified here are PI.

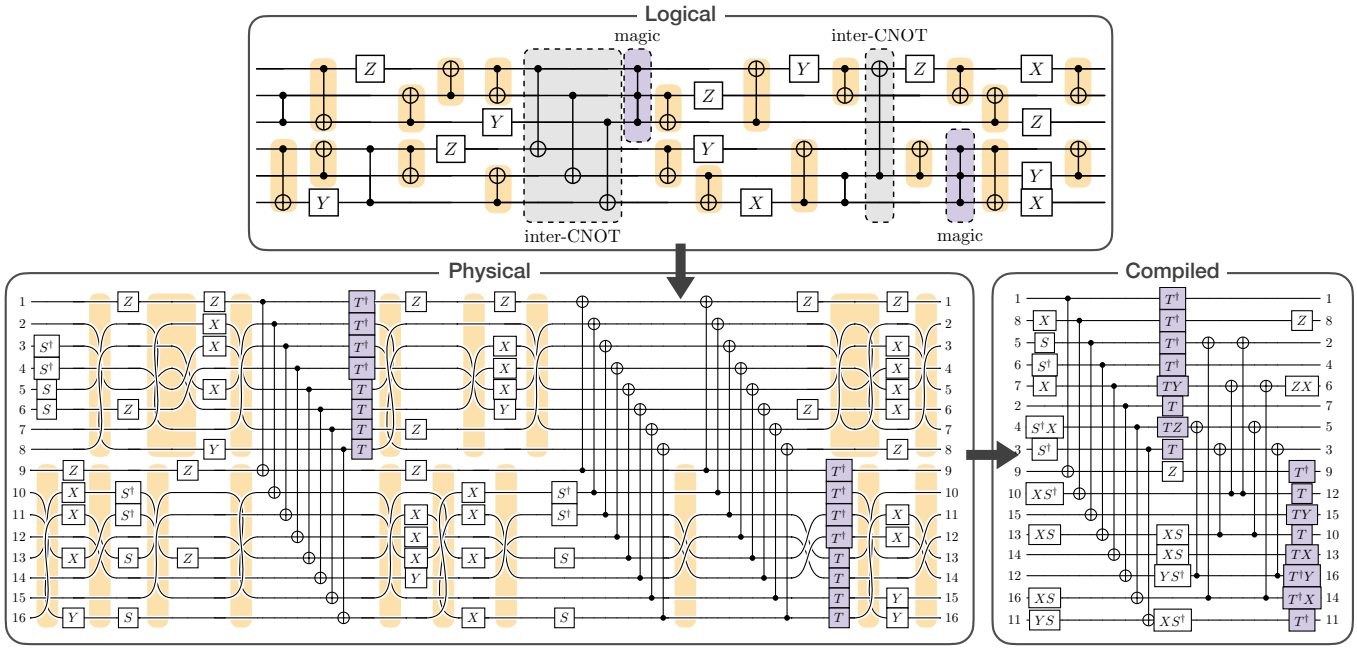



Figure 2. **Phantom codes in circuits with interblock and magic gates.** The benefit of phantom codes is most evident when $\overline{\text{CNOT}}$s appear alongside interblock and magic gates. Qubit permutations can be pulled through these operation without introducing operator spread, yielding substantial reductions in physical circuit size. In this example, the physical permutation required to implement the 20 in-block $\overline{\text{CNOT}}$s (highlighted in orange) vanish from the final compiled circuit.

with non-Clifford (magic) gates (Fig. 2) while still eliminating all in-block $\overline{\text{CNOT}}$s from the compiled circuit[2], thereby yielding zero-cost, all-to-all entangling connectivity within each codeblock.

Third, phantom codes exclude constructions that rely on permutations in combination with local physical Clifford gates [22, 23]. The reason is that zero-overhead, perfect-fidelity entangling gates are unique to *pure* permutations, which are the only operations that commute through arbitrary circuits without operator spread. Single-qubit Cliffords break this property.

Phantom codes exhibit several structural properties; we highlight three here and discuss others in App. A. First, phantom codes enable arbitrary $\overline{\text{CNOT}}$ circuits across *multiple codeblocks* to be efficiently executed. This is achieved by combining the complete, basis-resolved set of zero-depth in-block logical $\overline{\text{CNOT}}_{ab}$ gates with transversal interblock $\overline{\text{CNOT}}$s. This capability is formalized in the following theorem (see App. A 6 for the proof and Fig. 2 for an illustration).

**Theorem 1** (Efficient arbitrary $\overline{\text{CNOT}}$ circuits across CSS phantom codeblocks)**.** *Any logical circuit of* $\overline{\text{CNOT}}$ *gates acting on* $2^a$ *codeblocks, where* $a \in \mathbb{N}$*, of a CSS phantom code can be implemented in physical depth at most* $4(2^a - 1)$*, up to a residual permutation of logical qubits. This depth reduces to at most* $2(2^a - 1)$ *while maintaining the ordering of logical qubits when the* $\overline{\text{CNOT}}$ *gates are unidirectional.*

**Remark 1.** *The residual permutations of logical qubits in Theorem 1 incur zero cost when compiled away. In cases where they are necessary to be performed physically, they require depth at most* $8k+8$ *for any number of codeblocks of an* $[\![n, k, d]\!]$ *CSS phantom code.*

Second, the phantom property of CSS phantom codes is independent of the logical basis. This simplifies analysis and accelerates numerical searches.

Third, the weight distribution of physical Pauli operators within a logical Pauli equivalence class (e.g. $\overline{X}_1$, $\overline{Z}_2$, etc.), generated by stabilizer multiplication, are identical across all $\overline{X}$ equivalence classes and likewise for all $\overline{Z}$ classes. This observation, for example, leads to the following parameter bound, which is proved in App. A 5.

**Theorem 2** (Hamming bound for CSS phantom codes)**.** *An* $[\![n, k, d]\!]$ *CSS phantom code with* $d = d_\mu$ *for the sector* $\mu \in \{X, Z\}$ *satisfies*

$$\eta(2^k - 1) \leq B(n, d), \tag{1}$$

*where* $\eta \geq 1$ *is the number of weight-d* $\mu$*-type logical operators of the same logical equivalence class of the code, and* $B(n, d) \leq \binom{n}{d}$ *is the maximum number of weight-d binary strings of length* $n$ *such that the addition of any two strings is of weight at least* $d$*.*

[2] Even for Clifford circuits, absorbing $\overline{\text{CNOT}}$s by logical Pauli frame tracking induces logical operator spread, turning single-block measurements into joint measurements that require higher-overhead techniques (e.g. lattice surgery).

Theorem 2 constrains the possible $(n, k, d)$ parameters of CSS phantom codes. Certain phantom code families saturate this bound (see Sec. III E). We are not, however, aware of polynomial-time methods to compute $\eta$ and $B(n, d)$ in general, and using the $B(n, d) \leq \binom{n}{d}$ upper bound results in a loose Hamming bound for $d > 2$.

### B. Summary of results

To orient the reader, we briefly summarize the results of each section. We identify and construct phantom codes via four complementary approaches:

- *Exhaustive enumeration* (Sec. III A). We enumerate all $2.71 \times 10^{10}$ CSS codes with block length $n \leq 14$ and identify every phantom code within this range.
- *SAT-based search* (Sec. III B). To reach $n > 14$, we recast the search as a Boolean satisfiability problem, identify phantom codes up to $n = 21$, and establish minimal-$n$ realizations across a range of $k$ and $d$.
- *Quantum Reed–Muller (qRM) codes* (Sec. III C). To access higher $k$, we construct infinite families of phantom qRM codes, which generalize hypercube codes to higher distances, up to $d \leq \sqrt{n}$.
- *Binarization and concatenation scheme* (Sec. III D). To reach $d > \sqrt{n}$, we introduce a qudit-to-qubit construction which yields higher-distance codes.

For each of the identified phantom codes, we identify additional Clifford and non-Clifford logical operations in Sec. IV. These operations include gates implemented via local Cliffords and qubit permutations, fold-type gates involving in-block two-qubit interactions, and magic gates arising from diagonal single-qubit rotations.

Finally, in Sec. V, we present end-to-end benchmarks comparing a representative phantom code with surface-code baselines under realistic noise ($p = 10^{-3}$). We focus on two representative primitives:

- *GHZ state preparation* (Fig. 5b). For 4–64 logical qubits, the phantom code yields an approximately 56× reduction in logical infidelity at a 24% preselection acceptance rate, relative to surface codes with comparable physical-qubit counts.
- *Trotterized many-body simulation* (Fig. 6). For Hamiltonians with 8-body terms (string operators that naturally arise in several simulation contexts [24–26]) and 8–64 logical qubits, we observe an approximately 94× reduction in logical infidelity using nearly identical physical resources and the same 24% acceptance rate.

Sec. V also introduces practical tools for implementing non-LDPC phantom codes, including fault-tolerant state-preparation strategies and decoders that track spatiotemporal error correlations; these tools are directly applicable to other non-LDPC codes.

### C. Implications

The codes, benchmarks, methods and resources developed in this work carry several implications for fault-tolerant quantum computing.

First, phantom codes reduce logical error rates and overhead for circuits with dense local entanglement, bringing applications like fermionic simulation and correlated-phase preparation closer to practical realization. Together with recent works that lower the overhead of digital fermionic simulation [24, 27], these code- and algorithm-level advances markedly improve near-term experimental feasibility. This potential is accessible on hardware with long-range connectivity, including neutral-atom and trapped-ion platforms. Neutral-atom systems enable efficient parallel transversal logical entangling gates, whereas trapped-ion platforms lack a comparably parallel mechanism. Consequently, while phantom codes benefit both architectures, their impact is expected to be especially pronounced for trapped-ion processors.

Second, practical fault tolerance demands QEC codes that are co-designed for efficient storage and efficient logical computation. High rate and LDPC structure define one axis of optimization, supporting efficient state preparation, decoding, and low qubit overhead; an equally important axis is the availability of efficient logical gates, which distinguishes quantum memories from quantum computers. In this work, we target the zero-overhead extreme: although it appears to exclude high-rate qLDPC codes, the resulting codes are competitive in practical performance. This motivates developing QEC codes that optimize along both axes of this code design space.

The resources developed and compiled here enable systematic exploration of this design space. These include a complete database of $n \leq 14$ CSS codes, SAT-based searches for specified logical gate structure, and an automated pipeline for extracting logical operations. Within this space, phantom codes occupy an extreme point corresponding to vanishing logical entangling cost. By relaxing the zero-cost constraint, the same framework probes nearby trade-offs among logical overhead, code distance, stabilizer weight, and logical gate sets, enabling the identification of codes tailored to other hardware connectivities and objectives, such as reducing the cost of logical magic.

Finally, our benchmarking results motivate revisiting which QEC codes are viable for specific applications. With appropriate decoding and state-preparation protocols, our work suggests that non-LDPC codes can outperform LDPC codes on tailored tasks despite the latter's generic structural advantages.

| Code | Clifford | Fold | $d=2$ Magic | Found |
|---|---|---|---|---|
| $[\![4,2,2]\!]$ | CZ, $H^{\otimes 2}$ | $SS$ | – | [17] |
| $[\![8,3,2]\!]$ | $\mathrm{CZ}_{ij}$ | $S_iS_j$ | $\mathrm{C^2Z}$ | [17] |
| $[\![16,4,2]\!]$ | $\mathrm{CZ}_{ij}$ | $S_iS_j$ | $\mathrm{C^3Z}$ | [17] |
| $[\![12,2,4]\!]$ | $H^{\otimes 2}$ | $SS$, CZ | – | [6] |
| $[\![7,3,2]\!]$ | $\mathrm{CZ}_{ij}$ | $S_iS_j$ | – | Enumeration |
| $[\![12,2,2]\!]$ | CZ, $S_i$ | – | CS | Enumeration |
| $[\![14,3,3]\!]$ | – | $S_iS_j$, $\mathrm{CZ}_{ij}$ | – | Enumeration |
| $[\![18,2,5]\!]$ | $H^{\otimes 2}$ | $SS$, CZ | – | SAT search |
| $[\![20,2,6]\!]$ | CZ, $H^{\otimes 2}$ | $SS$ | – | SAT search |
| $[\![21,2,3]\!]$ | CZ, $S_i$ | – | – | SAT search |
| $[\![16,3,4]\!]$ | – | $S_iS_j$, $\mathrm{CZ}_{ij}$ | $\mathrm{C^2Z}$ | qRM |
| $[\![32,4,4]\!]$ | – | $S_iS_j$, $\mathrm{CZ}_{ij}$ | $\mathrm{C^3Z}$ | qRM |
| $[\![64,5,4]\!]$ | – | $S_iS_j$, $\mathrm{CZ}_{ij}$ | $\mathrm{C^4Z}$ | qRM |
| $[\![64,4,8]\!]$ | – | $S_iS_j$, $\mathrm{CZ}_{ij}$ | $\mathrm{C^3Z}$ | qRM |
| $[\![44,2,10]\!]$ | $H^{\otimes 2}$ | $SS$, CZ | – | B&C |
| $[\![52,2,10]\!]$ | CZ, $H^{\otimes 2}$ | ? | – | B&C |
| $[\![68,2,14]\!]$ | CZ, $H^{\otimes 2}$ | ? | – | B&C |
| $[\![76,2,14]\!]$ | $H^{\otimes 2}$ | $SS$, CZ | – | B&C |
| $[\![116,2,22]\!]$ | CZ, $H^{\otimes 2}$ | ? | – | B&C |

Table I. **Parameters and logical gate sets in representative phantom codes.** Listed are phantom codes that outperform prior examples and all phantom codes found in this work in parameters, gate sets, or both. Columns list the code parameters, Clifford gates available via single-qubit Cliffords and qubit permutations (i.e. code automorphisms), fold-diagonal Clifford gates, and magic gates. "–" denotes absence; "?" denotes unknown due to intractability. Fold-diagonal gates are non-exhaustive and omitted when already implementable by automorphisms. All magic gates are effectively distance-two. Any codes listed capable of implementing a $\mathrm{C^pZ}$ gate can also implement $\mathrm{C^qZ}$ gates for all $q < p$. Codes are obtained via exhaustive enumeration, SAT-based code discovery, quantum Reed–Muller (qRM) constructions, and binarized-qudit with concatenation (B&C) constructions. The Hadamard-dual of these codes obtained via code deformation by $H^{\otimes n}$ are also phantom, supporting the corresponding logical gate sets conjugated by $H^{\otimes k}$.

## III. CODE DISCOVERY & CONSTRUCTION

We identify phantom codes via a combination of numerical search and analytic construction. The main ideas are discussed here and full technical details are given in the Appendices. Table I summarizes representative phantom codes that outperform all previously known and newly generated codes in parameters, gate sets, or both.

Two major components of our numerical workflow employ SAT solvers [28]. Accordingly, App. B provides a brief primer. In short, SAT solving formulates a problem as Boolean constraints and searches for a satisfying assignment or certifies that none exists.

### A. Exhaustive code enumeration

Our first strategy is to construct a complete catalogue of inequivalent CSS codes to exhaustively identify all phantom codes up to some $n$. The dominant costs in this endeavor are (i) enumerating all admissible stabilizer groups defining candidate codes and (ii) identifying the equivalence class of each candidate code. The computational cost of both tasks scales superexponentially with $n$. We define equivalence up to qubit permutations and global Hadamards ($H^{\otimes n}$), as phantomness is preserved under these transformations.

Our approach builds on the method of Ref. [29], which enumerated stabilizer codes up to $n = 9$. Starting from the trivial $[\![n, n, 1]\!]$ code, we iteratively reduce $k$ by appending linearly independent, commuting Pauli operators to the stabilizer group, with each admissible extension defining a new code. At each $k$, codes are grouped into equivalence classes based on the Tanner graphs that represent their stabilizer groups, and a single representative code is retained.

To reach larger $n$, we introduce CSS-specific simplifications that reduce both the branching factor of the search and the cost of equivalence testing. First, letting $r_{\mathrm{x}}$ and $r_{\mathrm{z}}$ respectively denote the ranks of the $X$- and $Z$-type stabilizer groups, code equivalence up to Hadamard duality allows us to restrict to $r_{\mathrm{x}} \leq r_{\mathrm{z}}$, eliminating the remaining symmetric portion of the enumeration space. Second, CSS codes admit tripartite Tanner graphs [30], which simplifies canonical labelling and reduces memory requirement—an essential reduction given datasets spanning tens of TB. Computing and then deduplicating canonical labels [31] of Tanner graphs reduces equivalence testing from $\mathcal{O}(I^2)$ pairwise isomorphism checks to $\mathcal{O}(I)$ canonical-labelling operations, where the number of candidate codes $I$ in an iteration exceeds $10^{10}$ at the scale we explored.

Iterating this procedure yields all $2.71 \times 10^{10}$ inequivalent CSS codes of block length $n \leq 14$. By comparison, there are only 62 156 codes for $n \leq 9$, roughly a millionth of this dataset. To our knowledge, this is the largest complete stabilizer code catalogue assembled to date.

We identify all phantom codes in this catalogue using Boolean constraint satisfaction. Given $X$- and $Z$-type stabilizer generator matrices $H_{\mathrm{x}} \in \mathbb{F}_2^{r_{\mathrm{x}} \times n}$ and $H_{\mathrm{z}} \in \mathbb{F}_2^{r_{\mathrm{z}} \times n}$ defining a CSS code, we compute $X$- and $Z$-type logical operators $L_{\mathrm{x}}, L_{\mathrm{z}} \in \mathbb{F}_2^{k \times n}$ by Gaussian elimination. We then introduce a permutation matrix $P^{ab}$ representing the qubit permutation implementing $\overline{\mathrm{CNOT}}_{ab}$ for each pair of distinct logical qubits $(a, b) \in [k]^2$ as a free variable. Constraints enforcing the preservation of the codespace and correctness of the induced logical actions are compactly included in the SAT problem instance, which we solve using the state-of-the-art solver `kissat` [32]. A satisfying assignment signifies that the code is phantom; unsatisfiability certifies that no implementation of the gates exist.

Applying this procedure, we identify $1.39 \times 10^5$ CSS phantom codes up to $n = 14$—about one in every two hundred thousand CSS codes. Representative examples appear in Table I. Details of the enumeration algorithm, SAT formulation, and a breakdown of the results are

| $k$ | 2 | | | | | 3 | | | 4 |
|---|---|---|---|---|---|---|---|---|---|
| $d$ | 2 | 3 | 4 | 5 | 6 | 2 | 3 | 4 | 2 |
| Phantom | 4 | 11 | 12 | 18 | 20 | 7 | 14 | 15 | 15 |
| General | 4 | 10 | 12 | 18 | 20 | 6 | 11 | 14 | 6 |

Table II. **Minimal lengths of CSS phantom and general CSS codes.** Entries list the smallest block length $n$ for each logical-qubit count $k$ and distance $d$, for which a code exists.

given in App. C, along with an extension to non-CSS stabilizer codes.

### B. SAT-based code discovery

The exhaustive enumeration yielded a complete catalogue of CSS codes for $n \leq 14$, including all phantom codes. For $n > 14$, the number of CSS codes becomes prohibitive, so to identify phantom codes at larger $n$ and determine minimal block lengths at larger $k$ and $d$, we adopt a catalogue-free method.

Essentially, we convert the SAT check for phantomness into an automated code-discovery tool. Instead of fixing $H_x$ and $H_z$ as input, we treat them as free variables, allowing the SAT solver to construct codes that satisfy the required permutation gate-set constraints. We impose a standard form on $H_x, H_z$ so that a logical basis $L_x, L_z$ is explicit. Distance-$d$ constraints are enforced by demanding all Pauli errors of weight $< d$ (outside the stabilizers) to produce a nonzero syndrome. The full formulation, including non-CSS extensions, is provided in App. D.

Our SAT search uncovers CSS phantom codes up to $n = 21$, including several with best-known parameters or gate sets (see Table I). Unsatisfiability certificates allow us to determine minimal block lengths $n$ for a given $k$ and $d$ (see Table II). Compared with all CSS codes, phantom codes are near-identical in minimal block lengths for $k = 2$, exhibit small deviations for $k = 3$, and a larger deviation for the single $k = 4$ case, reflecting the increasing severity of the gate-set constraint on phantom codes.

### C. Phantom quantum Reed–Muller codes

Our numerical methods are tractable only for $k \leq 4$. To obtain phantom codes at larger $k$ and scalable code families beyond the reach of enumeration or SAT code discovery, we turn to an analytic construction based on quantum Reed–Muller (qRM) codes. qRM codes are CSS codes derived from pairs of classical Reed–Muller codes, with several families such as the $[\![2^D, D, 2]\!]$ hypercube codes as special cases. Our full derivation is detailed in App. E and uses the polynomial formalism [19, 33], which we review in App. E 1. Here we outline the construction and summarize the key properties.

Our construction begins with the qRM family $[\![2^m, \binom{m}{l}, \min(2^{m-l}, 2^l)]\!]$, where $m$ sets the number of variables and $l$ the monomial degree defining the $X$-type logical generators [6, 19]. These codes possess large automorphism groups and support products of $\overline{\text{CNOT}}$ gates implemented via permutations, which cannot be composed to produce individually addressable $\overline{\text{CNOT}}$s. Therefore these qRM codes are not generally phantom.

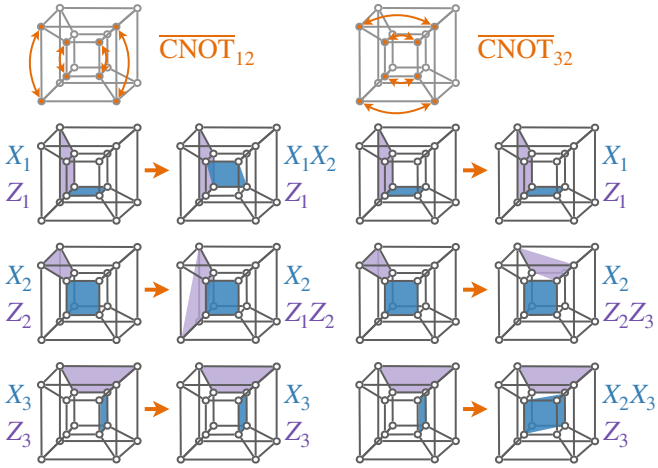



Figure 3. **The smallest error-correcting phantom qRM code.** Logical $\overline{\text{CNOT}}$ gates for the $[\![16, 3, 4]\!]$ phantom qRM code. The code is obtained by promoting three same-type logical operators ($X$ or $Z$) of the parent $[\![16, 6, 4]\!]$ code to stabilizers. The resulting reduction in the number of logical qubits enables arbitrary $\overline{\text{CNOT}}$ gates via qubit relabelling; examples of $\overline{\text{CNOT}}_{12}$ and $\overline{\text{CNOT}}_{32}$ are shown.

To obtain phantom codes, we fix selected logical qubits to $|\overline{0}\rangle$ or $|\overline{+}\rangle$, promoting the logical operators to $Z$- or $X$-type stabilizers. The resulting codes have parameters $[\![2^m, m - l + 1, \min(2^{m-l}, 2^l)]\!]$[3]. This choice is guided by the code's polynomial representation and admits a degree of flexibility to balance $X$- and $Z$-type stabilizer ranks, corresponding to different gauge-fixings of the parent qRM code. When $m = 2l$ and the $X$- and $Z$-distances are equal, such balancing can further improve logical error rates, a feature leveraged in our benchmarking (Sec. V).

A simple mnemonic for the phantom qRM code parameters is: starting from the hypercube parameters $[\![2^D, D, 2]\!]$, each reduction of $k$ by one doubles the distance $d$ at fixed $n$, up to $d = \sqrt{n}$. Representative examples appear in Table I, and Fig. 3 illustrates two permutation $\overline{\text{CNOT}}$s for the smallest ($d = 4$) error-correcting qRM code.

Beyond their phantom property, these codes offer several additional architectural advantages. They are compatible with preselection-based fault-tolerant state preparation [19], enabling high-fidelity codeblock initial-

[3] Concatenating a phantom code with a $k = 1$ inner quantum code preserves phantomness. While the code parameters can resemble concatenating a hypercube code with a repetition code [18], our qRM construction yields more balanced codes (see App. E 2).

ization despite their high-weight stabilizers. They support the full logical Clifford group via fold-$\overline{S}_i\overline{S}_j$ gates and teleported Hadamards from an all-$|\overline{+}\rangle$ state. Finally, they admit a $d = 2$ transversal magic-gate scheme: by temporarily projecting into the $[\![2^{m-l+1}, m-l+1, 2]\!]$ hypercube codes, applying their transversal non-Clifford gate, and returning to the phantom qRM codespace (see App. E).

### D. Binarization and concatenation scheme

While the phantom qRM family supports arbitrary $k$, its distance is bounded by $d \leq \sqrt{n}$. To surpass this bound at $k = 2$, we introduce an analytic construction based on high-distance self-dual GF(4) qudit codes encoding one logical qudit [34]. The underlying GF(4) codes achieve extremal distances at small block lengths.

To convert such a qudit code into a phantom code, we first binarize it by representing each GF(4) symbol by a pair of qubits [2]. We then concatenate each qubit pair with the $[\![4, 2, 2]\!]$ phantom code (Fig. 4). Both steps of this binarize-and-concatenate (B&C) scheme are essential: binarization alone does not yield a phantom code, as the $[\![4, 2, 2]\!]$ layer supplies a subset of required substructure for phantomness.

These codes inherit key properties from the starting qudit codes. Self-duality of the qudit code guarantees a nontrivial logical Hadamard on the resulting qubit code; Hermiticity additionally yields a $\overline{\text{CZ}}$ gate. As binarization does not decrease distance, their large qudit distances translate into high qubit distances. Additionally, concatenation with $[\![4, 2, 2]\!]$ introduces many low-weight stabilizers, though completing the stabilizer group still requires some generators of weight at least the code distance. More generally, any $k = 2$ CSS phantom code can be used for concatenation, but the $[\![4, 2, 2]\!]$ is favourable in encoding rate. Proofs and technical details of a phantom code family arising from quadratic-residue GF(4) codes are provided in App. F.

### E. Other code constructions

Finally, we discuss additional strategies for constructing phantom codes, highlighting a subset here and deferring full details to App. G.

- *Simple concatenation.* Concatenating a phantom outer code $[\![n_{\text{out}}, k, d_{\text{out}}]\!]$ with a high-distance inner code $[\![n_{\text{in}}, 1, d_{\text{in}}]\!]$ (e.g., a surface code) yields a phantom code with parameters $[\![n_{\text{in}}n_{\text{out}}, k, d_{\text{in}}d_{\text{out}}]\!]$. Logical $\overline{\text{CNOT}}$s are performed by relabellings at the outer level.

- *Hypergraph product (HGP) codes.* The HGP [30] increases both $k$ and $d$ while preserving permutation symmetries of the input classical codes [35]. Phantom codes can be constructed from, e.g., the HGP of a classical simplex and a repetition code (App. G 1). The smallest members of this family, $[\![7, 2, 2]\!]$ and $[\![49, 3, 4]\!]$, have lower rates than the phantom qRM codes.

- *Punctured hypercube codes.* Removing one qubit from the $[\![2^D, D, 2]\!]$ hypercube codes ($D > 2$) and truncating the affected stabilizers yields phantom codes with parameters $[\![2^D - 1, D, 2]\!]$. Both punctured and unpunctured versions saturate the Hamming bound for CSS phantom codes (Theorem 2), though puncturing can reduce the gate set. For example, the $[\![15, 4, 2]\!]$ code admits $\overline{\text{CCZ}}$ but not $\overline{\text{CCCZ}}$ (see App. G).

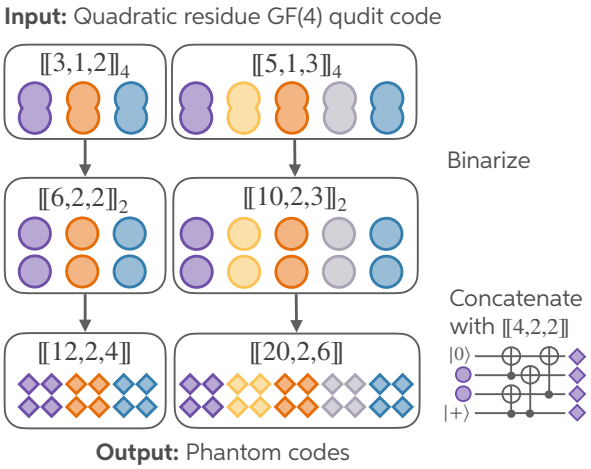



Figure 4. **Schematic construction of phantom codes from quadratic-residue qudit codes.** Phantom codes can be constructed by starting from a quadratic-residue $[\![n, k, d]\!]_4$ code over GF(4) and binarizing it to obtain a qubit code with parameters $[\![2n, 2k, d]\!]_2$. This intermediate code is not itself phantom, but it serves as the outer code in a concatenation with the $[\![4, 2, 2]\!]$ qubit code, and the resulting concatenated code is phantom. A detailed derivation is given in App. F.

## IV. LOGICAL GATES BEYOND PHANTOM

Practical QEC codes must support versatile fault-tolerant logical gates. We therefore identify additional Clifford and non-Clifford logical operations supported by phantom codes beyond their defining permutation $\overline{\text{CNOT}}$s. We begin with a fundamental constraint:

**Theorem 3** (No strictly transversal logical gates non–commuting with permutation logical gates)**.** *A stabilizer code supporting a logical gate $\overline{U}$ via qubit permutations can admit no strictly transversal logical gate $\overline{V}$ acting on any number of codeblocks $b$ where $\left[U^{\otimes b}, V\right] \neq 0$.*

On $b$ codeblocks of an $[\![n, k, d]\!]$ code, the strictly transversal gates of Theorem 3 have the form $\overline{V} = \prod_{i=1}^{n} W_{ii\cdots i}$, with $W_{ii\cdots i}$ acting on the $i^{\text{th}}$ qubit(s) of the codeblock(s). For phantom codes, this result precludes, for example, strictly transversal $\overline{H}$, $\overline{S}$, $\overline{\text{CZ}}$, and

magic gates. Additional logical gates beyond the permutation $\overline{\text{CNOT}}$s can arise from non-uniform single-qubit gates, qubit permutations, and in-block multi-qubit interactions. The three strategies described below address these mechanisms.

We first identify the logical gates arising from physical qubit permutations and local Cliffords, i.e. the *automorphism Cliffords*. Following Refs. [22, 36], we analyze the permutation automorphisms of the extended symplectic stabilizer generator matrix to determine the set of logical gates. Details are given in App. H 1; here we illustrate the approach using the $[\![4,2,2]\!]$ phantom code, which intersects several well-studied families [37–39]. Its stabilizers are generated by $X^{\otimes 4}$ and $Z^{\otimes 4}$, which in extended symplectic form are

$$\left[\begin{array}{cccc|cccc|cccc} 1&1&1&1&0&0&0&0&1&1&1&1\\ 0&0&0&0&1&1&1&1&1&1&1&1 \end{array}\right]. \tag{2}$$
$$\underbrace{\phantom{1111}}_{H^{(\mathrm{x})}} \quad \underbrace{\phantom{1111}}_{H^{(\mathrm{z})}} \quad \underbrace{\phantom{1111}}_{H^{(\mathrm{x})}\oplus H^{(\mathrm{z})}}$$

Column permutations that preserve the row span[4] represent physical SWAP, $H$, and $S$ combinations that act logically. For example, swapping columns 1–4 with 5–8 leaves the matrix invariant and corresponds to an $H^{\otimes 4}$, yielding the logical action $\overline{H}^{\otimes 2}\,\overline{\text{SWAP}}$. Thus, automorphism Cliffords follow directly from permutation symmetries of the extended stabilizer matrix.

Second, we identify logical diagonal magic gates obtained via non-uniform single-qubit physical $Z$ rotations [40, 41]. Fixing a level of the diagonal Clifford hierarchy (e.g. $T$ or $\sqrt{T}$), we solve for integer-power assignments of this gate on the physical qubits whose action preserves every logical computational-basis state up to a state-dependent phase. Each qubit may receive a distinct rotation angle. These yield logical diagonal gates at the chosen hierarchy level (see App. H 2).

Finally, we extend the diagonal-rotation method to identify fold-diagonal gates [22, 41]. The key idea is to embed a $[\![n,k,d]\!]$ code into a larger $[\![n+\binom{n}{2},k,d']\!]$ code and search for (non-uniform) single-qubit physical $Z$-basis rotations that act as logical operators on the enlarged code. Using the construction of App. H 2 c, any such operator induces a logical operation on the original code that is implemented by patterned one- and two-qubit diagonal interactions. Restricting to depth-one circuits yields fold gates, which in favourable cases preserve the $X$-sector code distance [23].

Together, these mechanisms generate a wide class of logical operations on phantom codes. For example, fold-$\overline{S}_i\overline{S}_j$ gates combined with automorphism operations implementing $\overline{H}^{\otimes k}$ generate the full logical Clifford group, as realized by the $[\![20,2,6]\!]$ phantom code.

[4] Technically, they must satisfy an additional condition (App. H 1).

## V. NUMERICAL BENCHMARKING

Lastly, we assess whether phantom codes can be practically useful in realistic fault-tolerant settings. Two structural properties make their practical usefulness non-obvious. First, the phantom codes identified in this study possess high-weight stabilizers and are non-LDPC, which generally precludes high-fidelity codeblock initialization and syndrome extraction using standard bare-ancilla techniques. Second, in generic logical circuits not all $\overline{\text{CNOT}}$s can be in-block.

To this end, we benchmark a concrete phantom code against a surface-code baseline using end-to-end noisy simulations. We focus on the $[\![64,4,8]\!]$ phantom qRM code, whose parameters are suitable for near-term experiments, and compare its performance to rotated surface codes of varying $d$ at both near-term ($10^{-3}$) and projected ($5\times10^{-4}$) physical error rates. [Results at a higher $3\times10^{-3}$ error rate are provided in App. I 7.] Our study is organized around three benchmarks of increasing complexity: (i) repeated in-block logical $\overline{\text{CNOT}}$ circuits on a single codeblock; (ii) logical GHZ state preparation across multiple codeblocks; and (iii) Trotterized many-body quantum simulation. These benchmarks probe regimes successively dominated by in-block entanglement and by interblock operations, with circuit widths far exceeding the per-codeblock logical dimension $k$.

### A. Operation of codes and error correction

We begin by specifying how the phantom and surface codes are operated, including state-preparation protocols, syndrome-extraction strategies, and decoding procedures, since this common framework underlies all our benchmarks.

Because all performance comparisons depend on the strength of the reference architecture, we use a fully optimized, state-of-the-art surface-code implementation as our baseline. The surface code is LDPC and has benefited from ∼25 years of development: it supports low-overhead state preparation, efficient hook-error-free bare-ancilla syndrome extraction, a near-optimal matching decoder, and transversal interblock $\overline{\text{CNOT}}$ gates.

Crucially, we use modern techniques of correlated decoding and algorithmic fault tolerance [42, 43] to ensure a strong surface-code baseline. These methods account for error propagation across codeblocks and allow the number of syndrome-extraction rounds per transversal $\overline{\text{CNOT}}$ to be reduced from $\mathcal{O}(d)$ to $\mathcal{O}(1)$ while maintaining fault tolerance. Concretely, we employ a recent matching-based correlated decoder [44] and, following prior results, use a single round of syndrome extraction per transversal $\overline{\text{CNOT}}$. Technical details of the surface-code implementation are given in App. I 2.

We next describe how we operate the $[\![64,4,8]\!]$ phantom code. Its non-LDPC structure necessitates syndrome-extraction, state-preparation, and decoding

procedures compatible with high-weight stabilizers:

- Because bare-ancilla syndrome extraction is not viable, we employ Steane-style quantum error correction, in which errors are coherently copied onto ancilla codeblocks that are then measured transversally.
- Bare-ancilla codeblock initialization is likewise precluded, so we develop a short-depth, fault-tolerant state preparation protocol extending Ref. [19]. Our scheme uses a compact four-to-one codeblock certification circuit, exploits qRM permutation automorphisms for fault tolerance, and operates as a state factory via error detection and preselection (see App. I 3 a). At two-qubit error rates of $10^{-3}$ and $5 \times 10^{-4}$, strict syndrome-free preselection yields acceptance rates of ∼24% and ∼49%, respectively; and allowing weight-one residual errors increases these to ∼40% and ∼64%. We focus on the stricter preselection protocol for comparison discussions in this section.
- To accurately decode large circuits, an efficient decoder that tracks spatiotemporal error correlations is needed. The $[\![64, 4, 8]\!]$ code is non-matchable, and existing approaches such as code-capacity list decoding [19, 45] do not extend directly to the correlated setting. We therefore develop spatiotemporal sliding-window correlated list and most-likely-error (MLE) decoders that account for error propagation through transversal $\overline{\text{CNOT}}$ gates and Steane error-correction gadgets. Although MLE decoding is not polynomial-time in general, our protocol operates only on constant-sized windows[5], yielding decoding costs that scale linearly with logical circuit size on a fixed code. Details are given in App. I 3 c.

Additionally, as described in Sec. III C, phantom qRM codes with identical $[\![n, k, d]\!]$ parameters can be obtained by different choices of gauge-fixed logical operators, which can favour the preparation fidelity of either $|\overline{0}\rangle^{\otimes k}$ or $|\overline{+}\rangle^{\otimes k}$. For the numerics presented here, we adopt a balanced construction in which neither state is disadvantaged (see Definition 8).

We employ a circuit-level noise model that includes single- and two-qubit gate errors, idle errors, and reset and measurement errors, with relative error rates calibrated to recent neutral-atom array experiments [3] (see also comparable error rates in Refs. [46–48]). Full technical details of our benchmarking appear in App. I.

### B. Single-codeblock repeated $\overline{\text{CNOT}}$ circuits

Our first benchmark isolates the most favourable regime for phantom codes, in which all $\overline{\text{CNOT}}$s are in-block. While in-block $\overline{\text{CNOT}}$s on phantom codes incur no cost after compilation, state preparation is generally more costly than on surface codes. This benchmark investigates when the reduction in gate cost outweighs the increased initialization overhead.

To probe this trade-off, we consider circuits consisting of fault-tolerant state preparation followed by repeated in-block $\overline{\text{CNOT}}$s acting on four logical qubits hosted on a single $[\![64, 4, 8]\!]$ codeblock or four surface codes (see Fig. 5a). We benchmark surface codes of distance $d = 5$–8. In all cases, results are averaged over $|\overline{0}\rangle^{\otimes 4}$ and $|\overline{+}\rangle^{\otimes 4}$ initial logical states to remove basis dependence.

We first assess logical state-preparation performance alone, corresponding to the $r = 0$ limit of Fig. 5a. The

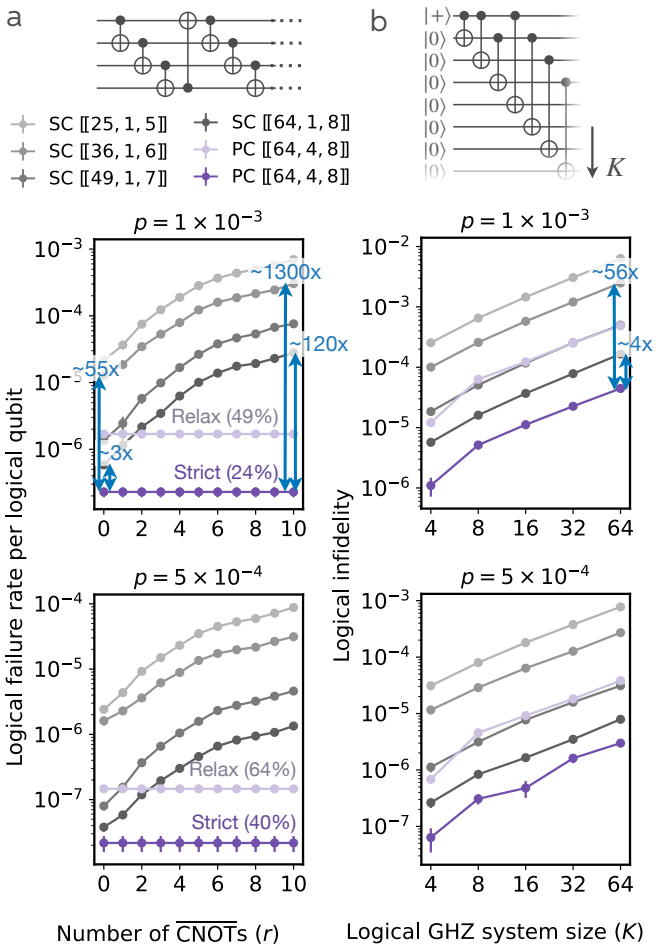



Figure 5. **Benchmarking phantom and surface codes in $\overline{\text{CNOT}}$ circuits and logical GHZ state preparation.** **(a)** Repeated in-block $\overline{\text{CNOT}}$ circuits on four logical qubits hosted on a single $[\![64, 4, 8]\!]$ phantom codeblock or four surface codeblocks ($d = 5$–8). The numerics include fault-tolerant state preparation and failure rates are averaged over $|\overline{0}\rangle^{\otimes 4}$ and $|\overline{+}\rangle^{\otimes 4}$ initial states. **(b)** Logical GHZ state preparation across multiple codeblocks, probing performance as the fraction of in-block permutation $\overline{\text{CNOT}}$s decreases. Dark and pale purple denote strict and relaxed preselection, respectively. Error bars are 98% confidence intervals.

[5] Constant-sized windows (around each transversal $\overline{\text{CNOT}}$ gate in our implementation) suffice because verified-ancilla Steane QEC limits correlations to a constant-sized spacetime neighbourhood.

phantom code uses 64 data qubits and $64 \times 3$ ancilla qubits for its state-preparation factory. Accounting for preselection overhead, it achieves a $\sim$55$\times$ reduction in logical failure rate over the $d = 6$ surface code with a comparable spatial footprint (144 data and 140 ancilla qubits). Even relative to the larger $d = 8$ surface code, requiring roughly twice as many physical qubits, the phantom code achieves a $\sim$3$\times$ advantage.

As in-block $\overline{\text{CNOT}}$s are added, the logical failure rate of the surface code grows linearly, whereas the phantom code's remains unchanged because its $\overline{\text{CNOT}}$s require no physical operations. At near-term error rates, the resulting improvement reaches $\sim$1300$\times$ ($\sim$120$\times$) over the $d = 6$ ($d = 8$) surface code on the deepest (depth-10) circuit studied. The logical gate time on the phantom code also remains effectively zero after state preparation, while it increases linearly with logical circuit depth on the surface code. The relative logical performance between phantom and surface codes remains similar at projected future physical error rates.

### C. Multiple-codeblock GHZ state preparation

Practical applications often require the use of multiple codeblocks, so our second benchmark studies logical performance as the fraction of in-block $\overline{\text{CNOT}}$s implemented via permutations is reduced.

To control the ratio of in-block permutation and interblock transversal $\overline{\text{CNOT}}$ gates in a simple and interpretable way, we benchmark logical GHZ state preparation across an increasing number of codeblocks. We consider logical system sizes $K = 4$–$64$ (see Fig. 5b). For $K = 4$, all three entangling $\overline{\text{CNOT}}$ gates are in-block and can be implemented as qubit permutations, giving a permutation-to-transversal gate ratio of $3 : 0$. As $K$ increases, entangling operations between codeblocks become necessary; for $K = 64$ this ratio decreases to $3{:}61$. We expect the phantom-code advantage to diminish as the contribution of transversal $\overline{\text{CNOT}}$s increases.

Compared to the single-codeblock benchmark of Sec. V B, GHZ state preparation introduces two additional factors for the phantom code. First, interblock $\overline{\text{CNOT}}$ gates necessitate active QEC, which we perform via Steane syndrome extraction as described in Sec. V A, incurring an overhead of two ancillary codeblocks per data codeblock. Second, the logical circuit requires codeblocks initialized in mixed computational- and Hadamard-basis logical states, namely the $|\overline{+000}\rangle$ state. Fault-tolerant preparation of mixed-basis logical states on $k > 1$ codes are generally costly. To implement this efficiently we (i) prepare two phantom codeblocks in $|\bar{0}\rangle^{\otimes k}$ and $|\overline{+}\rangle^{\otimes k}$, (ii) addressably teleport logical qubits between the codeblocks using at most two transversal $\overline{\text{CNOT}}$s, and (iii) measure one of the codeblocks transversally with preselection to suppress errors. We simulate the entire noisy process. Technical details are in App. I 5.

We compare the phantom code against surface codes matched either in spatial footprint or in code distance. For $K = 64$, a $d = 6$ surface code requires 64 codeblocks of 36 data plus 35 ancilla qubits each, for a total of 4544 qubits. Each $[\![64, 4, 8]\!]$ phantom codeblock requires two ancillary codeblocks for Steane QEC. To be conservative, we allocate one state-preparation factory per two data codeblocks, noting that this overhead could decrease on future hardware. At $K = 64$, this amounts to a footprint of 4608 qubits. At this scale, the phantom code achieves a $\sim$56$\times$ reduction in logical infidelity over the surface code. Compared to the $d = 8$ surface code, the phantom code retains $\sim$4$\times$ infidelity advantage while using $\sim$1.8$\times$ fewer physical qubits. Similar relative behaviour in logical fidelity persists at projected future physical error rates. Notably, the phantom-code advantage remains for all $K$, despite the increasing fraction of interblock $\overline{\text{CNOT}}$s.

### D. Trotterized many-body quantum simulation

We conclude our benchmarking study with an application-motivated test of large-scale quantum many-body simulation, asking whether phantom codes can provide a sustained advantage as system size increases and physical error rates improve. This benchmark is motivated by the natural suitability of phantom codes for implementing $\overline{\text{CNOT}}$ ladders, in which $\overline{\text{CNOT}}$s act sequentially on neighbouring logical qubits. Such structures arise in Hamiltonian simulation, where terms of the form $\exp\left(-i\theta Z^{\otimes L}\right)$ decompose into paired $\overline{\text{CNOT}}$ ladders surrounding a single-qubit rotation. When these ladders span multiple codeblocks, a logarithmic-depth hypercube layout of $\overline{\text{CNOT}}$ gates minimizes the number of required interblock transversal $\overline{\text{CNOT}}$s, which we exploit.

Concretely, we simulate Trotterized time-evolution generated by eight-body Ising Hamiltonian terms $\exp\left(-i\theta Z^{\otimes 8}\right)$ in a brickwork pattern, interleaved with single-qubit transverse-field terms $\exp(-i\phi X)$ (see Fig. 6). For the phantom code, each eight-body term use three in-block and four physical $\overline{\text{CNOT}}$s. We study systems with $K = 8$–$64$ logical qubits over eight Trotter steps. The largest circuit contains >2400 logical gates and have a two-qubit logical gate depth of 96 at $K = 64$.

At these widths, such quantum simulation circuits are generally classically intractable. For $K = 64$, the $[\![64, 4, 8]\!]$ phantom code again requires 4608 qubits, inclusive of ancillary codeblocks for Steane QEC and state-preparation factories. This is to be compared with 4544 (8128) qubits for the $d = 6$ ($d = 8$) surface code. To make simulations tractable, we fix $\theta = \phi = \pi/2$; in physical experiments, smaller angles would be used. Such rotations can be implemented using the STAR protocol [49], whose application to the $[\![64, 4, 8]\!]$ phantom code is discussed in App. E 6 (its use for surface codes was detailed in Ref. [49]).

We use two polynomial-time spatiotemporal sliding-window decoders for the phantom code: a most-likely-

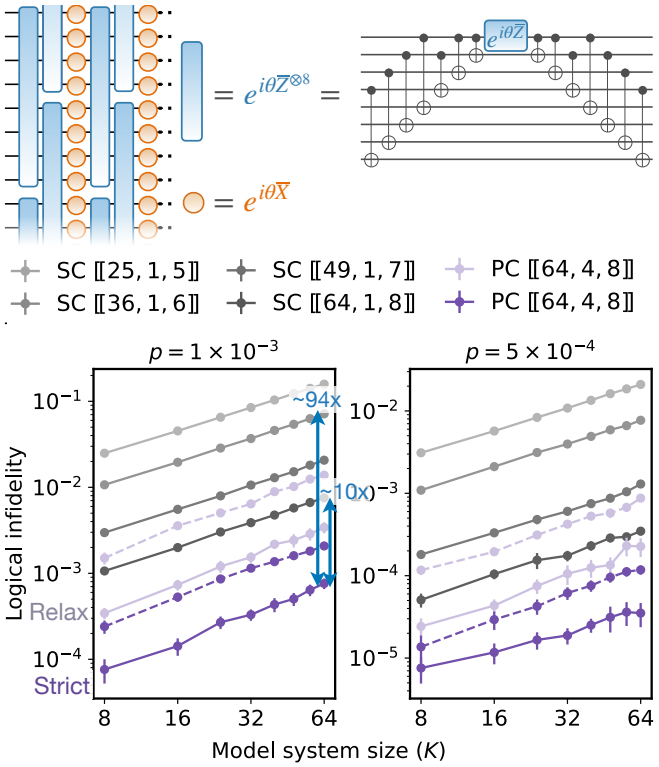



Figure 6. **Benchmarking phantom and surface codes in many-body quantum simulation.** Logical circuits perform Trotterized time-evolution of a Hamiltonian with eight-body Ising interaction terms and a transverse field, on up to 64 logical qubits over eight Trotter steps. The largest system size uses 16 codeblocks of the $[\![64, 4, 8]\!]$ phantom code, or 64 surface codeblocks of distance $d = 5$–$8$. Dark and pale purple denote strict and relaxed preselection for state preparation on the phantom code, respectively; solid and dashed lines denote sliding-window MLE and correlated list decoding. Error bars are 98% confidence intervals.

error (MLE) decoder, and a correlated list decoder that is ∼100× faster but less accurate (see App. I 3 c). The surface code continues to use the matching-based correlated decoder as described in Sec. V A and App. I 2. We use one round of syndrome extraction per transversal $\overline{\text{CNOT}}$ for both codes.

As shown in Fig. 6, the phantom code outperforms the surface code across near-term and projected future physical error rates, and this advantage is maintained as the system size increases. At near-term error rates, with sliding-window MLE decoding, the phantom codes achieves a ∼94× (∼10×) reduction in logical infidelity compared against the $d = 6$ ($d = 8$) surface code, with similar relative improvements at future error rates. Even with relaxed preselection, the phantom code outperforms the $d = 8$ surface code. Although the sliding-window correlated list decoder sacrifices decoding accuracy, the phantom code still exceeds the $d = 8$ ($d = 7$) surface code in logical fidelity under strict (relaxed) preselection.

## VI. DISCUSSION AND OUTLOOK

As emphasized, reducing QEC overhead requires jointly optimizing for storage and computation: what ultimately matters are the operations (including memory) that a QEC code enables. High rate and LDPC structure define one axis, enabling efficient state preparation, decoding, and low qubit overhead. An equally important axis is efficient logical gates, which distinguish quantum memories from quantum computers. Here, we identify a broad class of quantum error-correcting codes that realize logical entanglement without physical overhead or infidelity, and show that these codes can outperform leading fault-tolerant architectures for well-chosen tasks. These phantom codes bring workloads with dense local entangling structure closer to practical realization and enable new opportunities for logical algorithms in, for example, trapped-ion and neutral-atom platforms. The resources developed in this work broaden the scope of code discovery, enabling a systematic cataloguing of small-footprint CSS codes and direct solver-based search for codes with targeted properties.

Our results reveal that phantomness is a stringent structural condition that selects a highly symmetric corner of the space of codes and impose two practical consequences. First, all phantom codes discovered here are non-LDPC, necessitating Steane-style QEC, preselected logical-state factories, and the development of decoders presented in this work. Second, the phantom codes we identify encode $k = \mathcal{O}(\log n)$ logical qubits, limiting the fraction of phantom $\overline{\text{CNOT}}$s in large-scale algorithms. As a result, scalable advantages arise in workloads where entangling operations occur in dense local patches and logical blocks are connected via transversal $\overline{\text{CNOT}}$s.

These results open several new research directions, one of which is motivated by the observed limitations. The absence of LDPC phantom codes hint that permutation-implemented $\overline{\text{CNOT}}$s may be incompatible with bounded-weight or geometrically local stabilizer generators, motivating either new constructions or formal no-go theorems. Likewise, the empirical bound $k = \mathcal{O}(\log n)$ raise the question of whether large automorphism groups can coexist with higher encoding rates. Finally, the fact that all transversal magic gates we found are effectively distance-two may reflect a deeper structural obstruction. Resolving which features are fundamental would clarify the asymptotic limits of permutation-implemented $\overline{\text{CNOT}}$s.

A second direction concerns mapping the broader space of code design. Phantom codes minimize the temporal cost of logical entanglement, at the apparent expense of low encoding rates, whereas many prominent qLDPC codes exhibit the opposite trade-off. This contrast suggests an existence of a continuum between large-automorphism, low-rate codes and high-rate qLDPC constructions. To what extent this continuum exists, and how resource-optimal operating points are distributed along it, remains an open problem.

More broadly, the resources developed and assembled in this work—including a complete database of CSS codes up to $n = 14$, SAT-based search methods for identifying codes with specified logical gate structure, and an automated pipeline for extracting logical operations—provide the tools needed to explore this space systematically. These tools can be used, for example, to identify low-overhead codes tailored to specific hardware connectivities, such as on superconducting architectures, or to optimize alternative objectives, including reducing the cost of logical magic.

A third direction concerns software and decoding. The large automorphism groups of phantom codes suggest automorphism-aware compilers that treat permutations as free, pack $\overline{\text{CNOT}}$-dense subroutines into phantom codeblocks, and schedule interblock gates to minimize noise. Many workloads beyond the Trotterized circuits benchmarked contain a high density of logical entangling gates. Identifying other algorithms that benefit from phantom codes would help focus these compiler developments.

A final direction concerns experimental implementation. Small-angle rotations in our Trotterized circuits can be realized using the STAR protocol [49], opening the possibility of near-term fermionic simulations at realistic error rates. Beyond dynamics, phantom codes naturally support highly entangled logical states, which are central to measurement-based quantum computation [50] and quantum communication tasks [51]. Demonstrating such states with near-zero entangling-gate cost would validate the phantom-code paradigm.

## ACKNOWLEDGMENTS

We dedicate this work to the memory of Ray Laflamme (1960–2025), whose pioneering contributions and leadership have left a lasting legacy in quantum computing.

We would like to thank Gefen Baranes, Juan Pablo Bonilla Ataides, Madelyn Cain, Daniel Gottesmann, Milan Kornjača, Aleksander Kubica, Jonathan Lu, Nishad Maskara, Rohan Mehta, Chris Pattinson, John Preskill, Michael Vasmer, Adam Wills, Qian Xu, and Harry Zhou for useful discussions. We acknowledge financial support from the IARPA and the Army Research Office, under the Entangled Logical Qubits program (Cooperative Agreement Number W911NF-23-2-0219); the DARPA ONISQ MeasQuIT program (grant number HR0011-24-9-0359); the U.S. Department of Energy (DOE Quantum Systems Accelerator Center, contract number 7568717); the Center for Ultracold Atoms (an NSF Physics Frontier Center); the National Science Foundation (grant numbers CCF-2313084, QLCI grant OMA-2120757 and OMA-2016245, and NQVL: Pilot:DLPQC grant 2410716 and Design:ORAQL grant 2533041); Wellcome Leap Quantum for Bio program; and QuEra Computing. We acknowledge additional support from the A*STAR Graduate Academy (J.M.K.), a Harvard Quantum Initiative postdoctoral fellowship (D.B.T.), a Simons Investigator award (N.Y.Y.), and a Banting Postdoctoral Fellowship (S.M.).

## REFERENCES


[1] S. Majidy, C. Wilson, and R. Laflamme, *Building quantum computers: a practical introduction* (Cambridge University Press, 2024).

[2] D. Gottesman, Surviving as a quantum computer in a classical world, Textbook manuscript preprint **8**, 8 (2024).

[3] D. Bluvstein, A. A. Geim, S. H. Li, S. J. Evered, J. P. Bonilla Ataides, G. Baranes, A. Gu, T. Manovitz, M. Xu, M. Kalinowski, S. Majidy, C. Kokail, N. Maskara, E. C. Trapp, L. M. Stewart, S. Hollerith, H. Zhou, M. J. Gullans, S. F. Yelin, M. Greiner, V. Vuletić, M. Cain, and M. D. Lukin, A fault-tolerant neutral-atom architecture for universal quantum computation, Nature **649**, 39–(2025).

[4] N. Lacroix, A. Bourassa, F. J. Heras, L. M. Zhang, J. Bausch, A. W. Senior, T. Edlich, N. Shutty, V. Sivak, A. Bengtsson, *et al.*, Scaling and logic in the color code on a superconducting quantum processor, Nature **645**, 614 (2025).

[5] D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter, J. P. Bonilla Ataides, N. Maskara, I. Cong, X. Gao, P. Sales Rodriguez, T. Karolyshyn, G. Semeghini, M. J. Gullans, M. Greiner, V. Vuletić, and M. D. Lukin, Logical quantum processor based on reconfigurable atom arrays, Nature **626**, 58 (2024).

[6] B. W. Reichardt, D. Aasen, R. Chao, A. Chernoguzov, W. van Dam, J. P. Gaebler, D. Gresh, D. Lucchetti, M. Mills, S. A. Moses, B. Neyenhuis, A. Paetznick, A. Paz, P. E. Siegfried, M. P. da Silva, K. M. Svore, Z. Wang, and M. Zanner, Demonstration of quantum computation and error correction with a tesseract code (2024), arXiv:2409.04628 [quant-ph].

[7] A. Paetznick, M. P. da Silva, C. Ryan-Anderson, J. M. Bello-Rivas, J. P. C. III, A. Chernoguzov, J. M. Dreiling, C. Foltz, F. Frachon, J. P. Gaebler, T. M. Gatterman, L. Grans-Samuelsson, D. Gresh, D. Hayes, N. Hewitt, C. Holliman, C. V. Horst, J. Johansen, D. Lucchetti, Y. Matsuoka, M. Mills, S. A. Moses, B. Neyenhuis, A. Paz, J. Pino, P. Siegfried, A. Sundaram, D. Tom, S. J. Wernli, M. Zanner, R. P. Stutz, and K. M. Svore, Demonstration of logical qubits and repeated error correction with better-than-physical error rates (2024), arXiv:2404.02280 [quant-ph].

[8] Google Quantum AI and Collaborators, Quantum error correction below the surface code threshold, Nature **638**, 920 (2025).

[9] B. L. Brock, S. Singh, A. Eickbusch, V. V. Sivak, A. Z. Ding, L. Frunzio, S. M. Girvin, and M. H. Devoret, Quantum error correction of qudits beyond break-even, Nature **641**, 612 (2025).

[10] N. P. Breuckmann and J. N. Eberhardt, Quantum low-density parity-check codes, PRX Quantum **2**, 040101 (2021).

[11] L. Z. Cohen, I. H. Kim, S. D. Bartlett, and B. J. Brown, Low-overhead fault-tolerant quantum computing using long-range connectivity, Science Advances **8**, eabn1717 (2022).

[12] T. J. Yoder, E. Schoute, P. Rall, E. Pritchett, J. M. Gambetta, A. W. Cross, M. Carroll, and M. E. Beverland, Tour de gross: A modular quantum computer based on bivariate bicycle codes (2025), arXiv:2506.03094 [quant-ph].

[13] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, High-threshold and low-overhead fault-tolerant quantum memory, Nature **627**, 778 (2024).

[14] Q. Xu, H. Zhou, G. Zheng, D. Bluvstein, J. P. B. Ataides, M. D. Lukin, and L. Jiang, Fast and parallelizable logical computation with homological product codes, Phys. Rev. X **15**, 021065 (2025).

[15] Q. Xu, J. P. Bonilla Ataides, C. A. Pattison, N. Raveendran, D. Bluvstein, J. Wurtz, B. Vasić, M. D. Lukin, L. Jiang, and H. Zhou, Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays, Nature Physics **20**, 1084 (2024).

[16] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Surface codes: Towards practical large-scale quantum computation, Phys. Rev. A **86**, 032324 (2012).

[17] M. Vasmer and A. Kubica, Morphing quantum codes, PRX Quantum **3**, 030319 (2022).

[18] D. Hangleiter, M. Kalinowski, D. Bluvstein, M. Cain, N. Maskara, X. Gao, A. Kubica, M. D. Lukin, and M. J. Gullans, Fault-tolerant compiling of classically hard instantaneous quantum polynomial circuits on hypercubes, PRX Quantum **6**, 020338 (2025).

[19] A. Gong and J. M. Renes, Computation with quantum reed–muller codes and their mapping onto 2D atom arrays (2024), arXiv:2410.23263 [quant-ph].

[20] H. Pollatsek and M. B. Ruskai, Permutationally invariant codes for quantum error correction, Linear algebra and its applications **392**, 255 (2004).

[21] Y. Ouyang, Permutation-invariant quantum codes, Phys. Rev. A **90**, 062317 (2014).

[22] H. Sayginel, S. Koutsioumpas, M. Webster, A. Rajput, and D. E. Browne, Fault-tolerant logical clifford gates from code automorphisms, PRX Quantum **6**, 030343 (2025).

[23] A. J. Malcolm, A. N. Glaudell, P. Fuentes, D. Chandra, A. Schotte, C. DeLisle, R. Haenel, A. Ebrahimi, J. Roffe, A. O. Quintavalle, S. J. Beale, N. R. Lee-Hone, and S. Simmons, Computing efficiently in QLDPC codes (2025), arXiv:2502.07150 [quant-ph].

[24] N. Maskara, M. Kalinowski, D. Gonzalez-Cuadra, and M. D. Lukin, Fast simulation of fermions with reconfigurable qubits (2025), arXiv:2509.08898 [quant-ph].

[25] A. Wietek, Y.-Y. He, S. R. White, A. Georges, and E. M. Stoudenmire, Stripes, antiferromagnetism, and the pseudogap in the doped hubbard model at finite temperature, Phys. Rev. X **11**, 031007 (2021).

[26] R. Jafari, S. Mahdavifar, and A. Akbari, Geometrically frustrated anisotropic four-leg spin-1/2 nanotube, Journal of Physics: Condensed Matter **31**, 495601 (2019).

[27] N. Constantinides, J. Yu, D. Devulapalli, A. Fahimniya, L. Schaeffer, A. M. Childs, M. J. Gullans, A. Schuckert, and A. V. Gorshkov, Low-depth fermion routing without ancillas (2025), arXiv:2510.05099 [quant-ph].

[28] A. Biere, M. Heule, and H. van Maaren, *Handbook of satisfiability*, Vol. 185 (IOS press, 2009).

[29] A. Cross and D. Vandeth, Small binary stabilizer subsystem codes (2025), arXiv:2501.17447 [quant-ph].

[30] J.-P. Tillich and G. Zémor, Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength, IEEE Transactions on Information Theory **60**, 1193 (2013).

[31] B. D. McKay and A. Piperno, Practical graph isomorphism, ii, Journal of symbolic computation **60**, 94 (2014).

[32] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT competition 2024, Proc. of SAT Competition **B-2024-1**, 8 (2024).

[33] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*, Vol. 16 (Elsevier, 1977).

[34] F. J. MacWilliams, A. M. Odlyzko, N. J. Sloane, and H. N. Ward, Self-dual codes over GF(4), Journal of Combinatorial Theory, Series A **25**, 288 (1978).

[35] N. Berthusen, M. J. Gullans, Y. Hong, M. Mudassar, and S. J. S. Tan, Automorphism gadgets in homological product codes (2025), arXiv:2508.04794 [quant-ph].

[36] N. P. Breuckmann and S. Burton, Fold-transversal clifford gates for quantum codes, Quantum **8**, 1372 (2024).

[37] B. Criger and B. Terhal, Noise thresholds for the [[4, 2, 2]]-concatenated toric code, Quantum Information and Computation **16**, 1261 (2016).

[38] S. Majidy, A unification of the coding theory and oaqec perspectives on hybrid codes, International Journal of Theoretical Physics **62**, 177 (2023).

[39] A. Kubica, B. Yoshida, and F. Pastawski, Unfolding the color code, New Journal of Physics **17**, 083026 (2015).

[40] M. A. Webster, B. J. Brown, and S. D. Bartlett, The XP stabiliser formalism: a generalisation of the Pauli stabiliser formalism with arbitrary phases, Quantum **6**, 815 (2022).

[41] M. A. Webster, A. O. Quintavalle, and S. D. Bartlett, Transversal diagonal logical operators for stabiliser codes, New Journal of Physics **25**, 103018 (2023).

[42] M. Cain, C. Zhao, H. Zhou, N. Meister, J. P. B. Ataides, A. Jaffe, D. Bluvstein, and M. D. Lukin, Correlated decoding of logical algorithms with transversal gates, Phys. Rev. Lett. **133**, 240602 (2024).

[43] H. Zhou, C. Zhao, M. Cain, D. Bluvstein, N. Maskara, C. Duckering, H.-Y. Hu, S.-T. Wang, A. Kubica, and M. D. Lukin, Low-overhead transversal fault tolerance for universal quantum computation, Nature **646**, 303–(2025).

[44] M. Cain, D. Bluvstein, C. Zhao, S. Gu, N. Maskara, M. Kalinowski, A. A. Geim, A. Kubica, M. D. Lukin, and H. Zhou, Fast correlated decoding of transversal logical algorithms (2025), arXiv:2505.13587 [quant-ph].

[45] A. Gong and J. M. Renes, Improved logical error rate via list decoding of quantum polar codes, in *2024 IEEE International Symposium on Information Theory (ISIT)* (IEEE, 2024) pp. 2496–2501.

[46] H. J. Manetsch, G. Nomura, E. Bataille, X. Lv, K. H. Leung, and M. Endres, A tweezer array with 6,100 highly coherent atomic qubits, Nature **647**, 60 (2025).

[47] J. A. Muniz, D. Crow, H. Kim, J. M. Kindem, W. B. Cairncross, A. Ryou, T. C. Bohdanowicz, C.-A. Chen, Y. Ji, A. M. W. Jones, E. Megidish, C. Nishiguchi, M. Urbanek, L. Wadleigh, T. Wilkason, D. Aasen, K. Barnes, J. M. Bello-Rivas, I. Bloomfield, G. Booth, A. Brown, M. O. Brown, K. Cassella, G. Cowan, J. Epstein, M. Feldkamp, C. Griger, Y. Hassan, A. Heinz, E. Halperin, T. Hofler, F. Hummel, M. Jaffe, E. Kapit, K. Kotru, J. Lauigan, J. Marjanovic, M. Meredith, M. McDonald, R. Morshead, S. Narayanaswami, K. A. Pawlak, K. L. Pudenz, D. R. Pérez, P. Sabharwal, J. Simon, A. Smull, M. Sorensen, D. T. Stack, M. Stone, L. Taneja, R. J. M. van de Veerdonk, Z. Vendeiro, R. T. Weverka, K. White, T.-Y. Wu, X. Xie, E. Zalys-Geller, X. Zhang, J. King, B. J. Bloom, and M. A. Norcia, Repeated ancilla reuse for logical computation on a neutral atom quantum computer, Phys. Rev. X **15**, 041040 (2025).

[48] A. Senoo, A. Baumgärtner, J. W. Lis, G. M. Vaidya, Z. Zeng, G. Giudici, H. Pichler, and A. M. Kaufman, High-fidelity entanglement and coherent multiqubit mapping in an atom array (2025), arXiv:2506.13632 [quant-ph].

[49] Y. Akahoshi, K. Maruyama, H. Oshima, S. Sato, and K. Fujii, Partially fault-tolerant quantum computing architecture with error-corrected clifford gates and space-time efficient analog rotations, PRX Quantum **5**, 010337 (2024).

[50] H. J. Briegel, D. E. Browne, W. Dür, R. Raussendorf, and M. Van den Nest, Measurement-based quantum computation, Nature Physics **5**, 19 (2009).

[51] S. Majidy, D. Hangleiter, and M. J. Gullans, Scalable and fault-tolerant preparation of encoded $k$-uniform states, Phys. Rev. A **112**, 042409 (2025).

[52] S. Yu, Q. Chen, and C. H. Oh, Graphical quantum error-correcting codes (2007), arXiv:0709.1780 [quant-ph].

[53] L. d. Moura and N. Bjørner, Z3: An efficient SMT solver, in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Springer, 2008) pp. 337–340.

[54] A. Ekert and C. Macchiavello, Error correction in quantum communication (1996), arXiv:quant-ph/9602022 [quant-ph].

[55] M. Grassl and M. Roetteler, Leveraging automorphisms of quantum codes for fault-tolerant quantum computation, in *2013 IEEE International Symposium on Information Theory* (IEEE, 2013) pp. 534–538.

[56] A. Ignatiev, A. Morgado, and J. Marques-Silva, PySAT: A python toolkit for prototyping with SAT oracles, in *International Conference on Theory and Applications of Satisfiability Testing* (Springer, 2018) pp. 428–437.

[57] M. Soos, K. Nohl, and C. Castelluccia, Extending SAT solvers to cryptographic problems, in *International Conference on Theory and Applications of Satisfiability Testing* (Springer, 2009) pp. 244–257.

[58] L. Babai and E. M. Luks, Canonical labeling of graphs, in *Proceedings of the fifteenth annual ACM symposium on Theory of computing* (1983) pp. 171–183.

[59] T. Junttila and P. Kaski, Engineering an efficient canonical labeling tool for large and sparse graphs, in *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)* (SIAM, 2007) pp. 135–149.

[60] Blosc Development Team, A fast, compressed and persistent data store library (2009-2025).

[61] M. Tremblay, G. Duclos-Cianci, and S. Kourtis, Finite-rate sparse quantum codes aplenty, Quantum **7**, 985 (2023).

[62] D. Gottesman, *Stabilizer codes and quantum error correction* (California Institute of Technology, 1997).

[63] E. Arikan, A performance comparison of polar codes and reed-muller codes, IEEE Communications Letters **12**, 447 (2008).

[64] C. Gidney, Stim: a fast stabilizer circuit simulator, Quantum **5**, 497 (2021).

[65] C. Jones, Multilevel distillation of magic states for quantum computing, Phys. Rev. A **87**, 042305 (2013).

[66] A. Barg, N. J. Coble, D. Hangleiter, and C. Kang, Geo-

metric structure and transversal logic of quantum Reed–Muller codes, IEEE Transactions on Information Theory **72**, 415 (2026).
[67] N. Rengaswamy, R. Calderbank, M. Newman, and H. D. Pfister, On optimality of CSS codes for transversal $t$, IEEE Journal on Selected Areas in Information Theory **1**, 499 (2020).
[68] H. Choi, F. T. Chong, D. Englund, and Y. Ding, Fault tolerant non-Clifford state preparation for arbitrary rotations (2023), arXiv:2303.17380 [quant-ph].
[69] A. A. Kovalev and L. P. Pryadko, Quantum Kronecker sum-product low-density parity-check codes with finite rate, Phys. Rev. A **88**, 012311 (2013).
[70] W. Bosma, J. Cannon, and C. Playoust, The Magma algebra system I: The user language, Journal of Symbolic Computation **24**, 235 (1997).
[71] D. Gottesman, Theory of fault-tolerant quantum computation, Phys. Rev. A **57**, 127 (1998).
[72] M. Amy, D. Maslov, and M. Mosca, Polynomial-time $t$-depth optimization of Clifford+$t$ circuits via matroid partitioning, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **33**, 1476 (2014).
[73] E. T. Campbell and M. Howard, Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost, Phys. Rev. A **95**, 022316 (2017).
[74] M. Bilokur, S. Gopalakrishnan, and S. Majidy, Thermodynamic limitations on fault-tolerant quantum computing (2024), arXiv:2411.12805 [quant-ph].
[75] S. Dasu, S. Burton, K. Mayer, D. Amaro, J. A. Gerber, K. Gilmore, D. Gresh, D. DelVento, A. C. Potter, and D. Hayes, Breaking even with magic: demonstration of a high-fidelity logical non-Clifford gate (2025), arXiv:2506.14688 [quant-ph].
[76] P.-J. H. Derks, A. Townsend-Teague, A. G. Burchards, and J. Eisert, Designing fault-tolerant circuits using detector error models, Quantum **9**, 1905 (2025).
[77] O. Higgott and C. Gidney, Sparse Blossom: correcting a million errors per core second with minimum-weight matching, Quantum **9**, 1600 (2025).
[78] A. Paetznick and B. W. Reichardt, Fault-tolerant ancilla preparation and noise threshold lower bounds for the 23-qubit golay code (2013), arXiv:1106.2190 [quant-ph].
[79] A. Cassagne, O. Hartmann, M. Léonardon, K. He, C. Leroux, R. Tajan, O. Aumage, D. Barthou, T. Tonnellier, V. Pignoly, B. Le Gal, and C. Jégo, Aff3ct: A Fast Forward Error Correction Toolbox!, Elsevier SoftwareX **10**, 100345 (2019).
[80] S. T. Flammia and Y.-K. Liu, Direct fidelity estimation from few pauli measurements, Phys. Rev. Lett. **106**, 230501 (2011).
[81] J. M. Koh, D. E. Koh, and J. Thompson, Readout error mitigation for mid-circuit measurements and feedforward, PRX Quantum **7**, 010317 (2026).
[82] E. Bäumer, V. Tripathi, D. S. Wang, P. Rall, E. H. Chen, S. Majumder, A. Seif, and Z. K. Minev, Efficient long-range entanglement using dynamic circuits, PRX Quantum **5**, 030339 (2024).

# APPENDICES

## CONTENTS

## Appendix A: Phantom code basics, conditions, and properties

### 1. Review of binary symplectic representation of Pauli operators and Clifford gates

*Pauli operators.* We make pervasive use of the symplectic representation in this work. In this representation, a Pauli operator on $n$ qubits is encoded as a binary vector $(\mathbf{x} \mid \mathbf{z}) \in \mathbb{F}_2^{2n}$, where $\mathbf{x}$ and $\mathbf{z}$ indicate the positions of $X$ and $Z$ components, respectively. For example, $XYIZ$ is represented as $(1100 \mid 0101)$, as $Y = iXZ$ contributes to both parts. Commutation relations are captured by the symplectic inner product—defining[6]

$$\Omega = \begin{bmatrix} 0 & \mathbb{I} \\ \mathbb{I} & 0 \end{bmatrix}, \tag{A1}$$

two Pauli operators represented by symplectic vectors $\mathbf{v}_1 = (\mathbf{x}_1 \mid \mathbf{z}_1)$ and $\mathbf{v}_2 = (\mathbf{x}_2 \mid \mathbf{z}_2)$ commute if and only if $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle_S := \mathbf{v}_1 \Omega \mathbf{v}_2^{\mathrm{T}} = \mathbf{x}_1 \cdot \mathbf{z}_2 + \mathbf{x}_2 \cdot \mathbf{z}_1 = 0$. Finally, relevant to our setting here, a qubit permutation $\pi \in \mathrm{S}_n$ represented by an $n \times n$ permutation matrix $P$ acts on the symplectic representation of Pauli operators in a block-wise manner, $(\mathbf{x} \mid \mathbf{z}) \mapsto (\mathbf{x} \mid \mathbf{z})(P \oplus P) = (\mathbf{x}P \mid \mathbf{z}P)$.

*Clifford gates and circuits.* Clifford circuits are linear maps between Pauli operators under conjugation. That is, $UPU^\dagger = P'$ for a Clifford unitary $U$ and Pauli operators $P$ and $P'$. In the symplectic representation, on a register of $k$ qubits, a Clifford circuit $U$ can be represented by a binary Pauli transfer matrix $F(U) \in \mathrm{Sp}(2k, \mathbb{F}_2)$. That $F(U)$ is a symplectic matrix means that $F(U)^{\mathsf{T}} \Omega F(U) = \Omega$. For example, on $k = 1$ qubit, the Hadamard gate has the representation

$$F(H) = \left[\begin{array}{c|c} F_{\mathrm{xx}}(H) & F_{\mathrm{xz}}(H) \\ \hline F_{\mathrm{zx}}(H) & F_{\mathrm{zz}}(H) \end{array}\right] = \left[\begin{array}{c|c} 0 & 1 \\ \hline 1 & 0 \end{array}\right], \tag{A2}$$

and on a register of $k = 3$ qubits, a CNOT gate controlled by the first qubit and targeting the second has the representation

$$F(\mathrm{CNOT}_{12}) = \left[\begin{array}{c|c} F_{\mathrm{xx}}(\mathrm{CNOT}_{12}) & F_{\mathrm{xz}}(\mathrm{CNOT}_{12}) \\ \hline F_{\mathrm{zx}}(\mathrm{CNOT}_{12}) & F_{\mathrm{zz}}(\mathrm{CNOT}_{12}) \end{array}\right] = \left[\begin{array}{ccc|ccc} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]. \tag{A3}$$

The rows of the $F$ matrix correspond to input Pauli operators and the columns to their transformed outputs under conjugation, with basis elements ordered as $X_1, \ldots, X_k, Z_1, \ldots, Z_k$. Here, the defining action of $H$ is $X \mapsto Z$ and $Z \mapsto X$, and that of $\mathrm{CNOT}_{12}$ is $X_1 \mapsto X_1 X_2$, $Z_2 \mapsto Z_1 Z_2$, with all other basis Pauli operators remaining unchanged. This same formalism applies on logical qubits, but with physical Pauli operators replaced by logical Pauli operators (i.e. $X_1, \ldots, X_k, Z_1, \ldots, Z_k$ replaced by $\overline{X}_1, \ldots, \overline{X}_k, \overline{Z}_1, \ldots, \overline{Z}_k$).

Composition of Clifford gates or circuits in the symplectic representation corresponds to multiplying their Pauli transfer matrices. The Pauli transfer matrix of a circuit $U$ comprising first the subcircuit $U_1$, then $U_2$, henceforth till $U_l$ is given by $F(U) = F(U_l) \cdots F(U_2) F(U_1)$.

*CNOT and bias-preserving circuits.* A gate or circuit that is bias-preserving is one that maps $X$- ($Z$-) type Pauli operators solely to $X$- ($Z$-) type Pauli operators under conjugation. Modulo Pauli operators, the CNOT gate is the only generator of the Clifford group that is bias-preserving; hence bias-preserving Clifford circuits are exactly those that are generated by CNOT gates (i.e. CNOT circuits).

The bias-preserving property means that the off-diagonal blocks of the symplectic representation of CNOT gates and circuits vanish: $F_{\mathrm{xz}}(\mathrm{CNOT}_{ij}) = F_{\mathrm{zx}}(\mathrm{CNOT}_{ij}) = 0$ for any $\mathrm{CNOT}_{ij}$ gate, and more generally $F_{\mathrm{xz}}(C) = F_{\mathrm{zx}}(C) = 0$ for any CNOT circuit $C$. Moreover, as a consequence of reversibility of the circuit, the diagonal submatrices must be invertible. The binary Pauli transfer matrix $F(C)$ for a CNOT circuit on $k$ qubits takes the form

$$F(C) = \mathrm{diag}\left(A, A^{-\mathsf{T}}\right), \tag{A4}$$

---

[6] Throughout our manuscript, the sizes of the identity matrix $\mathbb{I}$ and symplectic metric $\Omega$ are inferred from context where unambiguous, and arithmetic on binary vectors and matrices are modulo-2 unless stated otherwise.

where $A \in \mathrm{GL}(k, \mathbb{F}_2)$, and we use the shorthand for the transpose of the inverse $A^{-\mathsf{T}} = (A^{-1})^{\mathsf{T}} = (A^{\mathsf{T}})^{-1}$. Such a CNOT circuit transforms $X$- and $Z$-type Pauli operators under conjugation as $\mathbf{x}^{\mathsf{T}} \mapsto A\mathbf{x}^{\mathsf{T}}$ and $\mathbf{z}^{\mathsf{T}} \mapsto A^{-\mathsf{T}}\mathbf{z}^{\mathsf{T}}$ in the symplectic representation. Equivalently, the CNOT circuit effects the action $|a\rangle \mapsto |aA^{\mathsf{T}}\rangle$ for computational basis states $|a\rangle$ where $a \in \{0,1\}^k$. Because $F(C)$ has the block-diagonal form in Eq. (A4), it suffices to consider the top-left $A$ block when analyzing CNOT circuits $C$, a fact that simplifies proofs in subsequent sections.

### 2. Conditions for permutation logical Clifford gate sets and phantomness on stabilizer codes

An $[\![n, k, d]\!]$ stabilizer code is specified by a full-row-rank stabilizer generator matrix $H \in \mathbb{F}_2^{r\times 2n}$ whose rows comprise a generating set of stabilizers in symplectic representation. Selecting an orthonormal logical basis fixes $Q_{\mathrm{x}}, Q_{\mathrm{z}} \in \mathbb{F}_2^{k\times 2n}$, whose rows are the symplectic representations of the logical operators $\{\overline{X}_i\}_{i=1}^k$ and $\{\overline{Z}_i\}_{i=1}^k$, such that $Q_{\mathrm{x}}\Omega Q_{\mathrm{z}}^{\mathsf{T}} = \mathbb{I}$. We can now express the conditions that the code has to satisfy to possess a given logical Clifford gate set implemented by a certain set of qubit permutations:

**Proposition 1** (Permutation logical Clifford gate set on a stabilizer code)**.** *Given an $[\![n, k, d]\!]$ stabilizer code with stabilizer generators $H \in \mathbb{F}_2^{r\times 2n}$ and an orthonormal logical basis $Q_{\mathrm{x}}, Q_{\mathrm{z}} \in \mathbb{F}_2^{k\times 2n}$ in symplectic form ($Q_{\mathrm{x}}\Omega Q_{\mathrm{z}}^{\mathsf{T}} = \mathbb{I}$), stack $Q := \binom{Q_{\mathrm{x}}}{Q_{\mathrm{z}}}$, and denote by $P^{(i)} \in \mathbb{F}_2^{n\times n}$ the permutation matrix representing $\pi^{(i)} \in \mathrm{S}_n$. Then $\{\pi^{(i)}\}_{i=1}^p$ implements the logical Clifford gate set $\{\overline{U}^{(i)}\}_{i=1}^p$ on this code in this logical basis iff:*

$$Q\big(P^{(i)} \oplus P^{(i)}\big)\Omega Q^{\mathsf{T}} = F\big(U^{(i)}\big)\Omega, \tag{A5a}$$

$$H\big(P^{(i)} \oplus P^{(i)}\big)\Omega H^{\mathsf{T}} = 0, \tag{A5b}$$

$$H\big(P^{(i)} \oplus P^{(i)}\big)\Omega Q^{\mathsf{T}} = 0, \tag{A5c}$$

$$Q\big(P^{(i)} \oplus P^{(i)}\big)\Omega H^{\mathsf{T}} = 0, \tag{A5d}$$

*for every $i \in [p]$.*

Above, Eq. (A5a) ensures that each permutation transforms logical operators correctly, modulo stabilizers, and hence implements the desired logical action; while Eqs. (A5b) to (A5d) demand that commutation relations among stabilizer and logical operators are maintained, so that the stabilizer group and hence codespace is preserved.

By definition, phantom codes support a complete set of individually addressable $\overline{\mathrm{CNOT}}$ gates performed by qubit permutations, thus leading to Proposition 2.

**Proposition 2** (Phantomness of a stabilizer code)**.** *An $[\![n, k, d]\!]$ stabilizer code is phantom iff it satisfies Proposition 1 for the gate set $\{\overline{\mathrm{CNOT}}_{ab} : (a, b) \in [k]^2, a \neq b\}$ for some choice of logical basis $Q_{\mathrm{x}}, Q_{\mathrm{z}}$.*

There is an additional consideration when using Proposition 1 to determine whether a stabilizer code supports a specific permutation logical gate set, or Proposition 2 to determine whether a code is phantom: the logical bases that enable the desired gate set to be implemented via permutations are generically unknown a priori. While a choice of $Q_{\mathrm{x}}, Q_{\mathrm{z}}$ can always be written given $H$ via basis completion (i.e. Gaussian elimination), this arbitrary choice may not support the specific desired gate set, and one must allow changes in logical basis to determine whether the code can support the gate set. To do so, we note that every pair of orthonormal logical bases $Q := \binom{Q_{\mathrm{x}}}{Q_{\mathrm{z}}}$ and $L := \binom{L_{\mathrm{x}}}{L_{\mathrm{z}}}$ of a code is related by a logical Clifford circuit, and accordingly, $Q = RL$ where $R \in \mathrm{Sp}(2k, \mathbb{F}_2)$—see App. A 1 for a review of the symplectic representation. Such a rotation of logical bases preserves the anticommutation relations, $Q_{\mathrm{x}}\Omega Q_{\mathrm{z}}^{\mathsf{T}} = \mathbb{I} \iff L_{\mathrm{x}}\Omega L_{\mathrm{z}}^{\mathsf{T}} = \mathbb{I}$. Therefore, in using Proposition 1 or Proposition 2 to determine the supported gate set or phantomness of a stabilizer code, a fixed logical basis $L$ can first be computed from $H$ and $Q$ set as $Q = RL$, where $R$ representing the Clifford transform is allowed to be freely chosen.

### 3. Conditions for permutation $\overline{\mathrm{CNOT}}$ gate sets and phantomness on CSS codes

We now specialize Propositions 1 and 2 to CSS codes. CSS codes are defined by stabilizer generators that each comprise purely $X$ or $Z$ Pauli operators, and their logical bases can similarly be chosen such that $\overline{X}_i$ ($\overline{Z}_i$) logicals comprise only $X$ ($Z$) operators, which we refer to as being in CSS form. Thus, a more compact "half-symplectic" binary representation is natural, where the zero $X$- or $Z$-portion of the symplectic representation is removed. An $[\![n, k, d]\!]$ CSS code is then specified by full-row-rank stabilizer generator matrices $H_{\mathrm{x}} \in \mathbb{F}_2^{r_{\mathrm{x}}\times n}$ and $H_{\mathrm{z}} \in \mathbb{F}_2^{r_{\mathrm{z}}\times n}$, whose rows encode generating set of $X$- and $Z$-type stabilizers, respectively, and likewise selecting an orthonormal CSS logical basis fixes $Q_{\mathrm{x}}, Q_{\mathrm{z}} \in \mathbb{F}_2^{k\times n}$ such that $Q_{\mathrm{x}}Q_{\mathrm{z}}^{\mathsf{T}} = \mathbb{I}$.

First, we establish results that restrict the possible scope of permutation logical gates on CSS codes, as formalized in Proposition 3 and Remark 2.

**Proposition 3** (Qubit permutations can achieve only $\overline{\text{CNOT}}$ circuits with CSS logical basis)**.** *The only logical action implementable by qubit permutations on a stabilizer code are $\overline{\text{CNOT}}$ circuits when using a CSS logical basis, modulo logical Pauli gates.*

*Proof.* A CSS logical basis defines logical operators that are pure $X$-or $Z$-type. Qubit permutations are bias-preserving and cannot induce Pauli basis changes—they map $X$- ($Z$)-type Pauli operators exactly onto $X$- ($Z$)-type Pauli operators, and so cannot change the $X$- or $Z$-type of logical operators on the code. The only bias-preserving Clifford circuits are CNOT circuits (see App. A 1), modulo single-qubit Pauli gates. Additionally, logical action at the third or higher level of the Clifford hierarchy is not possible as qubit permutations on stabilizer codes contain no magic. □

**Remark 2.** *Proposition 3 does not hold for stabilizer, even CSS, codes when a non-CSS logical basis is used.*

*Proof.* We give a counter-example. Consider a CSS logical basis of an arbitrary $k = 2$ code, $\{\overline{X}_1, \overline{X}_2, \overline{Z}_1, \overline{Z}_2\}$, and suppose $\overline{\text{CNOT}}_{12}$ is available through a qubit permutation $\pi$ in this basis. Now consider the non-CSS logical basis

$$\overline{X}'_1 = \overline{X}_1\overline{Z}_1, \qquad \overline{X}'_2 = \overline{X}_2\overline{Z}_2, \qquad \overline{Z}'_1 = \overline{X}_1\overline{X}_2\overline{Z}_2, \qquad \overline{Z}'_2 = \overline{X}_1\overline{X}_2\overline{Z}_1.$$

In this logical basis, $\pi$ has action $\overline{X}'_1 = \overline{X}_1\overline{Z}_1 \mapsto \overline{X}_1\overline{X}_2\overline{Z}_1 = \overline{Z}'_2$, which cannot be generated by $\overline{\text{CNOT}}$ gates. □

Remark 2 suggests that, to assess logical operations beyond $\overline{\text{CNOT}}$ circuits via qubit permutations on a CSS code, one can employ a non-CSS logical basis on the code. However, the availability of transversal (i.e. depth-one) $\overline{\text{CNOT}}$ gates between codeblocks generally require CSS logical bases to be chosen. Therefore, demanding the general ability to efficiently entangle distinct codeblocks through transversal $\overline{\text{CNOT}}$ gates places us within the regime of Proposition 3, which constrains available permutation logical actions to $\overline{\text{CNOT}}$ circuits. With this limitation, Proposition 1 simplifies to Proposition 4, and Proposition 6 follows.

**Proposition 4** (Permutation $\overline{\text{CNOT}}$-circuit gate sets on a CSS code)**.** *Given an $[\![n, k, d]\!]$ CSS code with stabilizer generators $H_{\text{x}} \in \mathbb{F}_2^{r_{\text{x}}\times n}$, $H_{\text{z}} \in \mathbb{F}_2^{r_{\text{z}}\times n}$ and an orthonormal CSS logical basis $Q_{\text{x}}, Q_{\text{z}} \in \mathbb{F}_2^{k\times n}$ in binary form ($Q_{\text{x}}Q_{\text{z}}^{\mathsf{T}} = \mathbb{I}$), denote by $P^{(i)} \in \mathbb{F}_2^{n\times n}$ the permutation matrix representing $\pi^{(i)} \in \text{S}_n$. Then $\{\pi^{(i)}\}_{i=1}^p$ implements the logical gate set $\{\overline{U}^{(i)}\}_{i=1}^p$ on this code in this logical basis iff:*

$$Q_{\text{x}}P^{a_ib_i}Q_{\text{z}}^{\mathsf{T}} = F_{\text{xx}}\big(U^{(i)}\big), \tag{A6a}$$
$$Q_{\text{z}}P^{a_ib_i}Q_{\text{x}}^{\mathsf{T}} = F_{\text{zz}}\big(U^{(i)}\big), \tag{A6b}$$
$$H_{\text{x}}P^{a_ib_i}H_{\text{z}}^{\mathsf{T}} = H_{\text{z}}P^{a_ib_i}H_{\text{x}}^{\mathsf{T}} = 0, \tag{A6c}$$
$$H_{\text{x}}P^{a_ib_i}Q_{\text{z}}^{\mathsf{T}} = H_{\text{z}}P^{a_ib_i}Q_{\text{x}}^{\mathsf{T}} = 0, \tag{A6d}$$
$$Q_{\text{x}}P^{a_ib_i}H_{\text{z}}^{\mathsf{T}} = Q_{\text{z}}P^{a_ib_i}H_{\text{x}}^{\mathsf{T}} = 0, \tag{A6e}$$

*for every $i \in [p]$, where each $U^{(i)} \in \langle \text{CNOT}_{jk} : j, k \in [k]^2, j \neq k\rangle$.*

The issue of an unknown suitable choice of logical basis $Q_{\text{x}}, Q_{\text{z}}$ again arises when using Proposition 4 to determine whether a CSS code supports a specific gate set. We address this by noting that every pair of orthonormal CSS logical bases $Q_{\text{x}}, Q_{\text{z}}$ and $L_{\text{x}}, L_{\text{z}}$ is related by a $\overline{\text{CNOT}}$ circuit—namely, $Q_{\text{x}} = RL_{\text{x}}$ and $Q_{\text{z}} = R^{-\mathsf{T}}L_{\text{z}}$ where $R \in \text{GL}(k, \mathbb{F}_2)$. The $\overline{\text{CNOT}}$ circuit $C$ performing the logical basis change is then described by $F(C) = \text{diag}(R, R^{-\mathsf{T}})$; see App. A 1 for a review of the symplectic representation. Such a rotation of logical basis preserves the anticommutation relations, $Q_{\text{x}}Q_{\text{z}}^{\mathsf{T}} = \mathbb{I} \iff L_{\text{x}}L_{\text{z}}^{\mathsf{T}} = \mathbb{I}$. Therefore, in using Proposition 4 to determine the supported permutation gate set of a CSS code, a fixed logical basis $L_{\text{x}}, L_{\text{z}}$ can first be computed from $H_{\text{x}}, H_{\text{z}}$, and $Q_{\text{x}}, Q_{\text{z}}$ then set as the rotated versions of $L_{\text{x}}, L_{\text{z}}$, where $R$ are allowed to be freely chosen.

Interestingly, while the specific permutation gate set of a CSS code is logical basis-dependent, the phantomness of the code is basis-independent. We formalize this in Proposition 5.

**Proposition 5** (Phantomness is independent of CSS logical basis)**.** *If a stabilizer code is phantom in some CSS logical basis, it is phantom in every CSS logical basis.*

*Proof.* Any two CSS logical bases are related by a logical $\overline{\text{CNOT}}$ circuit, and a phantom code can implement any $\overline{\text{CNOT}}$ circuit through qubit permutations. Suppose the code is phantom in some CSS logical basis $L_{\text{x}}, L_{\text{z}}$. Then, in some other CSS logical basis $L'_{\text{x}}, L'_{\text{z}}$, to perform an arbitrary $\overline{\text{CNOT}}$, one does the following: (1) transform the logical basis to $L_{\text{x}}, L_{\text{z}}$ through qubit permutations; (2) implement the $\overline{\text{CNOT}}$ as known; (3) transform the logical basis back to $L'_{\text{x}}, L'_{\text{z}}$ through qubit permutations. The entire procedure comprises only qubit permutations, so the code is phantom also in logical basis $L'_{\text{x}}, L'_{\text{z}}$. □

This basis independence allows a simplification of Proposition 2 to Proposition 6. Here, as explained in the main text and motivated above, we restrict our attention to CSS logical bases only on CSS codes, so as to guarantee the availability of transversal $\overline{\text{CNOT}}$ gates between codeblocks.

**Proposition 6** (Phantomness of a CSS code). *An $[\![n,k,d]\!]$ CSS code is phantom iff it satisfies Proposition 4 for the gate set $\{\overline{\text{CNOT}}_{ab} : (a,b) \in [k]^2, a \neq b\}$ for an arbitrarily chosen CSS logical basis $Q_{\text{x}}, Q_{\text{z}}$.*

Lastly, we remark that a smaller gate set can be used in the conditions of Propositions 2 and 6 when assessing whether a code is phantom. Doing so is advantageous for computational efficiency, for example in our code enumeration and SAT-based code discovery efforts described later in Apps. C and D.

**Remark 3** (Minimal-size gate set for phantomness of stabilizer codes). *In Propositions 2 and 6, the smaller-sized gate set $\{\overline{\text{CNOT}}_{12}, \overline{\text{SWAP}}_{12}, \overline{\text{SWAP}}_{23}, \ldots, \overline{\text{SWAP}}_{(k-1)(k)}\}$ can equivalently be used.*

*Proof.* Either gate set generates the other. □

### 4. Code equivalence under qubit permutation and global Hadamards

As mentioned in the main text and to facilitate further technical discussion, it is useful to define a suitable notion of equivalence of codes. While prior literature considered equivalence of codes up to qubit permutations and local (single-qubit) Cliffords [29, 52], the phantom property of codes is not preserved under local Clifford deformations. It is most suitable instead to restrict the freedom of local Cliffords to global Hadamards, leading to Definition 2.

**Definition 2** (ΠH equivalence of codes). *Two $n$-qubit codes are ΠH-equivalent if one can be mapped onto the other through a qubit permutation and optionally a global Hadamard ($H^{\otimes n}$).*

This notion of equivalence preserves the $[\![n,k,d]\!]$ parameters and crucially the phantom property of codes.

**Proposition 7** (Invariance of phantomness under ΠH equivalence). *Any code that is ΠH-equivalent to a phantom code is also phantom.*

### 5. Uniform weight distribution of logical operators on phantom codes and the Hamming bound

As the weight, and in fact at a finer-grained level the $X$-, $Y$- and $Z$-operator weights, of a Pauli operator is invariant under qubit permutations, yet qubit permutations induce logical $\overline{\text{CNOT}}$ actions on phantom codes, phantom codes must possess an underlying structure of weight uniformity on their logical operators. This observation ultimately enables the derivation of a Hamming bound on the parameters of phantom codes. We formalize these concepts and this line of argument in this section.

**Definition 3** (Logical Pauli equivalence classes). *For a logical Pauli operator $\overline{P}$ on a stabilizer code, we define:*

- *$\mathcal{P}_{\overline{P}}$ to be the set of all physical Pauli operators implementing the logical Pauli $\overline{P}$, i.e., the set obtained by multiplying a fixed representative of $\overline{P}$ by all elements of the stabilizer group of the code.*
- *$\mathcal{P}_{\overline{P},w}$ to be the subset of $\mathcal{P}_{\overline{P}}$ consisting of physical Pauli operators of weight $w$.*
- *$\mathcal{P}_{\overline{P},\mathbf{w}}$ to be the subset of $\mathcal{P}_{\overline{P}}$ consisting of physical Pauli operators with weight vector $\mathbf{w} := (w_{\text{x}}, w_{\text{z}}, w_{\text{y}})$, where $w_{\text{x}}$, $w_{\text{z}}$, and $w_{\text{y}}$ count the numbers of $X$, $Z$, and $Y$ operators, respectively, in the physical Pauli implementation.*

Note that the $\mathcal{P}_{\overline{P},w}$ subsets are disjoint for different $w$—$\mathcal{P}_{\overline{P},w} \cap \mathcal{P}_{\overline{P},w'} =$ when $w \neq w'$—and likewise $\mathcal{P}_{\overline{P},\mathbf{w}} \cap \mathcal{P}_{\overline{P},\mathbf{w}'} =$ when $\mathbf{w} \neq \mathbf{w}'$. As the weight of a Pauli operator is the sum of its part-wise weights, $|\mathbf{w}| = w_{\text{x}} + w_{\text{z}} + w_{\text{y}}$, we have also that $\mathcal{P}_{\overline{P},w} = \bigcup_{\mathbf{w}:|\mathbf{w}|=w} \mathcal{P}_{\overline{P},\mathbf{w}}$, where all subsets in the union are disjoint. That the distance of the code is $d$ implies that there can be no logical operators of weight less than $d$, therefore $\mathcal{P}_{\overline{P},w} = \emptyset$ for all $w < d$, and likewise $\mathcal{P}_{\overline{P},\mathbf{w}} = \emptyset$ for all $\mathbf{w}$ with $|\mathbf{w}| < d$.

As a concrete example, we list comprehensively the (nontrivial) logical Pauli equivalence classes and their weights for the $[\![4,2,2]\!]$ phantom code in Table III. We read, for instance, that $|\mathcal{P}_{\overline{X}_1}| = 4$ is naturally the same size as the stabilizer group of the code; $|\mathcal{P}_{\overline{X}_1,2}| = 2$ and $|\mathcal{P}_{\overline{X}_1,4}| = 2$. As the code is CSS and we have used a CSS logical basis, there is only a single possible weight vector for each weight: $|\mathcal{P}_{\overline{X}_1,(2,0,0)}| = |\mathcal{P}_{\overline{X}_1,(0,2,2)}| = 2$.

| Logical Pauli equivalence class | Weight $\lvert\mathbf{w}\rvert$ | $\mathbf{w} = (w_\mathrm{x}, w_\mathrm{z}, w_\mathrm{y})$ | Class members (physical Pauli operators) |
|---|---|---|---|
| $\overline{X_1}$ | 2 | $(2,0,0)$ | $X_1X_4, X_2X_3$ |
| | 4 | $(0,2,2)$ | $Y_1Z_2Z_3Y_4, Z_1Y_2Y_3Z_4$ |
| $\overline{X_2}$ | 2 | $(2,0,0)$ | $X_1X_2, X_3X_4$ |
| | 4 | $(0,2,2)$ | $Y_1Y_2Z_3Z_4, Z_1Z_2Y_3Y_4$ |
| $\overline{X_1X_2}$ | 2 | $(2,0,0)$ | $X_1X_3, X_2X_4$ |
| | 4 | $(0,2,2)$ | $Y_1Z_2Y_3Z_4, Z_1Y_2Z_3Y_4$ |
| $\overline{Z_1}$ | 2 | $(0,2,0)$ | $Z_1Z_2, Z_3Z_4$ |
| | 4 | $(2,0,2)$ | $X_1X_2Y_3Y_4, Y_1Y_2X_3X_4$ |
| $\overline{Z_2}$ | 2 | $(0,2,0)$ | $Z_1Z_4, Z_2Z_3$ |
| | 4 | $(2,0,2)$ | $X_1Y_2Y_3X_4, Y_1X_2X_3Y_4$ |
| $\overline{Z_1Z_2}$ | 2 | $(0,2,0)$ | $Z_1Z_3, Z_2Z_4$ |
| | 4 | $(2,0,2)$ | $X_1Y_2X_3Y_4, Y_1X_2Y_3X_4$ |
| $\overline{X_1Z_2}$ | 2 | $(0,0,2)$ | $Y_1Y_4, Y_2Y_3$ |
| | 4 | $(2,2,0)$ | $X_1Z_2Z_3X_4, Z_1X_2X_3Z_4$ |
| $\overline{Z_1X_2}$ | 2 | $(0,0,2)$ | $Y_1Y_2, Y_3Y_4$ |
| | 4 | $(2,2,0)$ | $X_1X_2Z_3Z_4, Z_1Z_2X_3X_4$ |
| $\overline{Y_1Y_2}$ | 2 | $(0,0,2)$ | $Y_1Y_3, Y_2Y_4$ |
| | 4 | $(2,2,0)$ | $X_1Z_2X_3Z_4, Z_1X_2Z_3X_4$ |
| $\overline{Y_1}$ | 3 | $(1,1,1)$ | $X_1Z_3Y_4, X_2Y_3Z_4, Y_1Z_2X_4, Z_1Y_2X_3$ |
| $\overline{Y_2}$ | 3 | $(1,1,1)$ | $X_1Y_2Z_3, Y_1X_2Z_4, Z_1X_3Y_4, Z_2Y_3X_4$ |
| $\overline{X_1Y_2}$ | 3 | $(1,1,1)$ | $X_1Z_2Y_3, Y_1X_3Z_4, Y_2Z_3X_4, Z_1X_2Y_4$ |
| $\overline{Y_1X_2}$ | 3 | $(1,1,1)$ | $X_1Y_3Z_4, X_2Z_3Y_4, Y_1Z_2X_3, Z_1Y_2X_4$ |
| $\overline{Y_1Z_2}$ | 3 | $(1,1,1)$ | $X_1Z_2Y_4, Y_1Z_3X_4, Y_2X_3Z_4, Z_1X_2Y_3$ |
| $\overline{Z_1Y_2}$ | 3 | $(1,1,1)$ | $X_1Y_2Z_4, Y_1X_2Z_3, Z_1Y_3X_4, Z_2X_3Y_4$ |

Table III. **Logical Pauli equivalence classes of the $[\![\mathbf{4,2,2}]\!]$ code and their weights.** The last column lists explicitly all members of each class $\mathcal{P}_{\overline{P},\mathbf{w}}$; the size of the class $|\mathcal{P}_{\overline{P},\mathbf{w}}|$ is the length of the list.

**Lemma 1** (Uniform Pauli weight distribution of logical operators on phantom stabilizer codes)**.** *For an $[\![n,k,d]\!]$ phantom stabilizer code and any fixed weight vector $\mathbf{w}$, the size of the equivalence class $|\mathcal{P}_{\overline{P},\mathbf{w}}|$, in other words the number of physical Pauli operators implementing the logical Pauli operator $\overline{P}$, is the same for all nontrivial $X$-type logical operators $\overline{P}$. (There are $2^k-1$ such operators: $\overline{X}_1, \overline{X}_2, \ldots, \overline{X}_1\overline{X}_2, \ldots, \overline{X}_1\overline{X}_2\cdots\overline{X}_k$.) The same statement holds for $Z$-type logical operators.*

*Proof.* Consider any two $X$-type logical operators $\overline{P}$ and $\overline{P}'$. These must be related by $\overline{\mathrm{CNOT}}$ transformations: there exist $\overline{\mathrm{CNOT}}$ circuits $C$ and $C'$ transforming $\overline{P}$ to $\overline{P}'$ and $\overline{P}'$ to $\overline{P}$, respectively (in fact $C'$ is the circuit $C$ reversed). As the code is phantom, $C$ and $C'$ can be implemented by qubit permutations. Any qubit permutation preserves the weight vector of any physical Pauli operator. Therefore, each physical Pauli operator in the equivalence class $\mathcal{P}_{\overline{P},\mathbf{w}}$ is mapped to a physical Pauli operator in $\mathcal{P}_{\overline{P}',\mathbf{w}}$, and likewise from $\mathcal{P}_{\overline{P}',\mathbf{w}}$ to $\mathcal{P}_{\overline{P},\mathbf{w}}$. This defines a bijective map between $\mathcal{P}_{\overline{P},\mathbf{w}}$ and $\mathcal{P}_{\overline{P}',\mathbf{w}}$, so the two sets have the same cardinality. An analogous argument applies to $Z$-type logical operators. □

A consequence of this structure of weight uniformity is a Hamming bound for CSS phantom codes, which limits the possible $[\![n,k,d]\!]$ parameters of a CSS phantom code. This result has been presented as Theorem 2 in the main text and is reproduced below for convenient reference. Here, we provide a proof for the bound.

**Theorem 2** (Hamming bound for CSS phantom codes)**.** *An $[\![n,k,d]\!]$ CSS phantom code with $d = d_\mu$ for the sector $\mu \in \{X,Z\}$ satisfies*

$$\eta(2^k-1) \le B(n,d), \tag{1}$$

*where $\eta \ge 1$ is the number of weight-$d$ $\mu$-type logical operators of the same logical equivalence class of the code, and $B(n,d) \le \binom{n}{d}$ is the maximum number of weight-$d$ binary strings of length $n$ such that the addition of any two strings is of weight at least $d$.*

| $d$ | $n=4$ | $n=5$ | $n=6$ | $n=7$ | $n=8$ | $n=9$ | $n=10$ | $n=11$ | $n=12$ | $n=13$ | $n=14$ | $n=15$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 6 (6) | 10 (10) | 15 (15) | 21 (21) | 28 (28) | 36 (36) | 45 (45) | 55 (55) | 66 (66) | 78 (78) | 91 (91) | 105 (105) |
| 3 | 1 (2) | 2 (3) | 4 (5) | 7 (7) | $\geq 8$ (9) | 12 (12) | $\geq 13$ (15) | $\geq 17$ (18) | $\geq 20$ (22) | 26 (26) | $\geq 27$ (30) | 35 (35) |
| 4 | 1 (1) | 1 (2) | 3 (5) | 7 (8) | 14 (14) | $\geq 16$ (21) | 30 (30) | $\geq 31$ (41) | $\geq 41$ (55) | $\geq 53$ (71) | $\geq 68$ (91) | $\geq 85$ (113) |
| 5 | 0 (0) | 1 (1) | 1 (2) | 1 (3) | 2 (5) | 3 (8) | $\geq 6$ (12) | $\geq 8$ (16) | $\geq 11$ (22) | $\geq 14$ (28) | $\geq 27$ (36) | $\geq 22$ (45) |

Table IV. **Computed exact values and bounds on $B(n,d)$.** Used in the Hamming bound, $B(n,d)$ is the maximum number of length-$n$ weight-$d$ bitstrings with pairwise weight $\geq d$. Entries denoted with $\geq$ are best-known lower bounds obtained by the `Z3` SMT solver [53] within a 12-hour time limit; the other numbers are proven optimal. Upper bounds of $\lfloor \binom{n}{t}/\binom{d}{t} \rfloor$, $t = \lfloor d/2 \rfloor + 1$, are in parentheses.

*Proof.* Consider the case $\mu = X$, that is, the $X$-distance is limiting. By Lemma 1, all $2^k - 1$ nontrivial $X$-type logical operators $\overline{P}$ have equivalence classes of the same size at weight $d$, namely $|\mathcal{P}_{\overline{P},d}| = \eta$. Thus, the left-hand side of the Hamming bound, $\eta(2^k - 1)$, counts the total number of weight-$d$ Pauli-$X$ physical operators that implement nontrivial logical $X$-type operators. Note that, because logical states on a code are orthogonal, these physical operators are all distinct.

On the other hand, the right-hand side $B(n,d)$ is an upper bound on this quantity, since each such physical Pauli-$X$ operator corresponds to a binary string of length $n$ and weight $d$, and because the code has distance $d$, the product of any two such physical operators must have weight at least $d$. This is precisely the constraint in the definition of $B(n,d)$. The case of $\mu = Z$ is analogous. □

We make a few further remarks on the Hamming bound. First, in the presentation of Theorem 2, we implicitly assume that $\eta$ is known. If only $n, k, d$ are specified and $\eta$ is unknown, $\eta$ may be replaced by any lower bound on the size of the equivalence class of physical weight-$d$ Pauli operators implementing a type-$\mu$ logical Pauli.

Secondly, the right-hand side of Theorem 2 can in principle be tightened by computing sharper upper bounds on the number of type-$\mu$ physical Pauli operators that implement logical operators. Using the `Z3` SMT solver [53], we computed exact values of $B(n,d)$ for small $n$ and $d$, which are reported in Table IV. For $d = 2$, one has an exact closed-form formula $F(n,2) = n(n-1)/2$, since any two distinct weight-2 bitstrings can overlap in at most one position. The parentheses in Table IV enclose upper bounds $B(n,d) \leq \lfloor \binom{n}{t}/\binom{d}{t} \rfloor$ where $t = \lfloor d/2 \rfloor + 1$. To see this bound, identify the support of each weight-$d$ string with a block $S \subseteq [n]$ of size $d$. For two blocks $S$ and $T$, the condition $|S \oplus T| \leq d$ implies $|S \cap T| \leq t - 1$, equivalently that no $t$-subset of $[n]$ is contained in more than one block. Each block contains $\binom{d}{t}$ distinct $t$-subsets, while each $t$-subset can appear in at most one block. Counting $t$-subsets therefore yields the stated upper bound.

Lastly, Theorem 2 is reminiscent of the Hamming bound for classical linear codes. We emphasize that, for a classical linear code, the codewords are only required to have weight at least $d$, whereas in the definition of $B(n,d)$, all strings are constrained to have weight exactly $d$. We comment that there is also a quantum Hamming bound for non-degenerate stabilizer codes [54], but some phantom codes are degenerate (e.g. the $[\![20,2,6]\!]$ code), so that bound does not directly apply.

### 6. Efficient addressable $\overline{\text{CNOT}}$ gates and arbitrary $\overline{\text{CNOT}}$ circuits between CSS phantom codeblocks

Phantom codes support super-efficient arbitrary in-block $\overline{\text{CNOT}}$ circuits implemented by qubit permutations, which do not have to be physically performed as circuit compilation can commute the permutations through the rest of the circuit. We additionally show that individually addressable interblock $\overline{\text{CNOT}}$ gates, and arbitrary $\overline{\text{CNOT}}$ circuits spanning multiple codeblocks, enjoy benefits from the phantomness of CSS codes and can too be performed efficiently, as has been formalized in Theorem 1 and Remark 1 in the main text. Here we give the proofs of these results.

The central idea is to interleave in-block permutation $\overline{\text{CNOT}}$ gates between interblock transversal $\overline{\text{CNOT}}$ gates [55] to obtain arbitrary interblock $\overline{\text{CNOT}}$ connectivity. This decomposition is applied recursively from large to small scale on the desired logical circuit, with suitable parallelization of the required transversal $\overline{\text{CNOT}}$ gates to control depth. As our codes are phantom and the qubit permutations implementing in-block $\overline{\text{CNOT}}$ gates need not be physically performed, the only contribution to physical depth comes from the transversal $\overline{\text{CNOT}}$s, which are depth one each.

**Lemma 2** (Efficient arbitrary $\overline{\text{CNOT}}$ circuits between two CSS phantom codeblocks)**.** *Arbitrary $\overline{\text{CNOT}}$ circuits between two codeblocks of a CSS phantom code can be performed in physical depth at most four, up to a residual permutation of logical qubits. This depth reduces to at most two whilst preserving the ordering of logical qubits when the $\overline{\text{CNOT}}$ gates in the circuit are unidirectional.*

*Proof.* We consider two codeblocks of an $[\![n,k,d]\!]$ CSS phantom code, for a total of $2k$ logical qubits. Recall that, in the symplectic representation, a CNOT circuit $C$ on $2k$ qubits is described by a symplectic matrix $F(C) \in \text{Sp}(4k, \mathbb{F}_2)$

with a block-diagonal structure $F(C) = \mathrm{diag}(X, X^{-\mathsf{T}})$ for $X \in \mathrm{GL}(2k, \mathbb{F}_2)$—see App. A 1 for a review. Henceforth we deal only the top-left block $X$, which contains all degrees of freedom characterizing the circuit. We consider cases of increasing complexity:

- $X$ is upper-block-triangular,

$$X = \left(\begin{array}{c|c} \mathbb{I} & U \\ \hline 0 & \mathbb{I} \end{array}\right), \tag{A7}$$

where each block is $k \times k$. First, if $U$ is invertible, then the circuit can be implemented using a single transversal $\overline{\mathrm{CNOT}}$ gate between the codeblocks, sandwiched by in-block $\overline{\mathrm{CNOT}}$ gates on the control codeblock:

$$X = \left(\begin{array}{c|c} \mathbb{I} & U \\ \hline 0 & \mathbb{I} \end{array}\right) = \underbrace{\left(\begin{array}{c|c} U & 0 \\ \hline 0 & \mathbb{I} \end{array}\right)}_{\text{in-block}} \underbrace{\left(\begin{array}{c|c} \mathbb{I} & \mathbb{I} \\ \hline 0 & \mathbb{I} \end{array}\right)}_{\text{transversal}} \underbrace{\left(\begin{array}{c|c} U^{-1} & 0 \\ \hline 0 & \mathbb{I} \end{array}\right)}_{\text{in-block}}. \tag{A8}$$

Otherwise, if $U$ is not invertible, by Ref. [23, Lemma IX.7], $U$ can be written as the sum of two $\mathrm{GL}(k, \mathbb{F}_2)$ matrices $U_1, U_2$. The block-triangular structure of these matrices enables us to rewrite the sum as a product, and we use the circuit decomposition of Eq. (A8) for each of the terms,

$$\begin{aligned} X = \left(\begin{array}{c|c} \mathbb{I} & U_1 + U_2 \\ \hline 0 & \mathbb{I} \end{array}\right) &= \left(\begin{array}{c|c} \mathbb{I} & U_1 \\ \hline 0 & \mathbb{I} \end{array}\right) \left(\begin{array}{c|c} \mathbb{I} & U_2 \\ \hline 0 & \mathbb{I} \end{array}\right) \\ &= \underbrace{\left(\begin{array}{c|c} U_1 & 0 \\ \hline 0 & \mathbb{I} \end{array}\right)}_{\text{in-block}} \underbrace{\left(\begin{array}{c|c} \mathbb{I} & \mathbb{I} \\ \hline 0 & \mathbb{I} \end{array}\right)}_{\text{transversal}} \underbrace{\left(\begin{array}{c|c} U_1^{-1} & 0 \\ \hline 0 & \mathbb{I} \end{array}\right)}_{\text{in-block}} \underbrace{\left(\begin{array}{c|c} U_2 & 0 \\ \hline 0 & \mathbb{I} \end{array}\right)}_{\text{in-block}} \underbrace{\left(\begin{array}{c|c} \mathbb{I} & \mathbb{I} \\ \hline 0 & \mathbb{I} \end{array}\right)}_{\text{transversal}} \underbrace{\left(\begin{array}{c|c} U_2^{-1} & 0 \\ \hline 0 & \mathbb{I} \end{array}\right)}_{\text{in-block}}. \end{aligned} \tag{A9}$$

Therefore, the circuit can be implemented using two transversal $\overline{\mathrm{CNOT}}$ gates between the codeblocks, interleaved with in-block $\overline{\mathrm{CNOT}}$ gates on the control codeblock.

- $X$ is lower-block-triangular. The argument is analogous as for the upper-block-triangular case. Circuits with unidirectional $\overline{\mathrm{CNOT}}$ gates correspond precisely to either a lower- or upper-block-triangular $X$, hence proving the second part of Lemma 2.

- $X$ is not block-triangular. In this case, we invoke the block-PLDU decomposition[7]:

$$X = P \left(\begin{array}{c|c} \mathbb{I} & 0 \\ \hline L & \mathbb{I} \end{array}\right) \underbrace{\left(\begin{array}{c|c} C_1 & 0 \\ \hline 0 & C_2 \end{array}\right)}_{\text{in-block}} \left(\begin{array}{c|c} \mathbb{I} & U \\ \hline 0 & \mathbb{I} \end{array}\right), \tag{A10}$$

where $P$ is a permutation matrix, $L, U \in \mathbb{F}_2^{k \times k}$, and $C_1, C_2 \in \mathrm{GL}(k, \mathbb{F}_2)$. The middle $\mathrm{diag}(C_1, C_2)$ term corresponds to in-block $\overline{\mathrm{CNOT}}$ gates on the two codeblocks, while the lower- and upper-block-triangular terms can be treated as described above. Thus, at most four transversal $\overline{\mathrm{CNOT}}$ gates between the codeblocks are needed, up to a permutation of logical qubits $P$.

□

Now we examine $\overline{\mathrm{CNOT}}$ circuits across a larger number of codeblocks, which is the general setting considered in Theorem 1 of the main text. We reproduce the statement here for reference and provide a proof.

**Theorem 1** (Efficient arbitrary $\overline{\mathrm{CNOT}}$ circuits across CSS phantom codeblocks)**.** *Any logical circuit of $\overline{\mathrm{CNOT}}$ gates acting on $2^a$ codeblocks, where $a \in \mathbb{N}$, of a CSS phantom code can be implemented in physical depth at most $4(2^a - 1)$, up to a residual permutation of logical qubits. This depth reduces to at most $2(2^a - 1)$ while maintaining the ordering of logical qubits when the $\overline{\mathrm{CNOT}}$ gates are unidirectional.*

---

[7] For invertible $X$, we can first apply a permutation matrix $P$, such that $P^{-1}X$ has an invertible upper-left quadrant $A$, then use the block-LDU decomposition $\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) = \left(\begin{array}{c|c} I & 0 \\ \hline CA^{-1} & I \end{array}\right) \left(\begin{array}{c|c} A & 0 \\ \hline 0 & D - CA^{-1}B \end{array}\right) \left(\begin{array}{c|c} I & A^{-1}B \\ \hline 0 & I \end{array}\right)$.

*Proof.* There are $2^a$ codeblocks of an $[\![n,k,d]\!]$ CSS phantom code, amounting to $2^a k$ logical qubits in total. Like before, let $X \in \mathrm{GL}(2^a k, \mathbb{F}_2)$ be the top-left block of the symplectic matrix representing the CNOT circuit. We invoke the block-PLDU decomposition on $X$ as in Eq. (A10), and further decompose $C_1, C_2 \in \mathrm{GL}(2^{a-1}k, \mathbb{F}_2)$ into PLDU forms, which involve permutation matrices $P_1, P_2$. We can then pull $P_1, P_2$ to the left,

$$X = P \left(\begin{array}{c|c} \mathbb{I} & 0 \\ \hline L & \mathbb{I} \end{array}\right) \left(\begin{array}{c|c} P_1 & 0 \\ \hline 0 & P_2 \end{array}\right) \cdots = P \left(\begin{array}{c|c} P_1 & 0 \\ \hline 0 & P_2 \end{array}\right) \left(\begin{array}{c|c} \mathbb{I} & 0 \\ \hline P_2^{-1} L P_1 & \mathbb{I} \end{array}\right) \cdots . \tag{A11}$$

Merging $P$ and $\mathrm{diag}(P_1, P_2)$ leads to another permutation matrix. Repeating this process further, we obtain the following decomposition:

$$X = P \left(\begin{array}{c|c} \mathbb{I} & 0 \\ \hline L_0 & \mathbb{I} \end{array}\right) \left(\begin{array}{cc|c} \mathbb{I} & 0 & 0 \\ L_{1,1} & \mathbb{I} & \\ \hline 0 & & \begin{array}{cc} \mathbb{I} & 0 \\ L_{1,2} & \mathbb{I} \end{array} \end{array}\right) \cdots \underbrace{\begin{pmatrix} C_1 & & & \\ & C_2 & & \\ & & \ddots & \\ & & & C_{2^a} \end{pmatrix}}_{\text{in-block}} \cdots \left(\begin{array}{cc|c} \mathbb{I} & U_{1,1} & 0 \\ 0 & \mathbb{I} & \\ \hline 0 & & \begin{array}{cc} \mathbb{I} & U_{1,2} \\ 0 & \mathbb{I} \end{array} \end{array}\right) \left(\begin{array}{c|c} \mathbb{I} & U_0 \\ \hline 0 & \mathbb{I} \end{array}\right), \tag{A12}$$

where $L_{i,j}, U_{i,j}$ for $j \in [2^i]$ are binary square matrices of size $2^{a-i-1}k \times 2^{a-i-1}k$, and $C_i \in \mathrm{GL}(k, \mathbb{F}_2)$ for $i \in [2^a]$. The $C_i$ term in the middle corresponds to in-block $\overline{\mathrm{CNOT}}$ gates on the codeblocks independently. It then remains to implement the lower- and upper-block-triangular block matrices in this decomposition. To reduce depth, we seek to parallelize the required interblock $\overline{\mathrm{CNOT}}$ interactions as far as possible. This can be done through cyclic scheduling [23], which amounts to decomposing the block matrix into a sum of matchings on the complete directed graph of codeblocks. All $\overline{\mathrm{CNOT}}$ interactions in a matching are on disjoint pairs of codeblocks, so each matching is fully parallelizable; the overall depth of the implementation is then determined by the number of matchings (i.e. rounds) required. As a concrete example, a cyclic scheduling for an upper-block-triangular term where $V$ is $4k \times 4k$ is

$$\left(\begin{array}{c|c} \mathbb{I} & V \\ \hline 0 & \mathbb{I} \end{array}\right) = \left(\begin{array}{c|c} \mathbb{I} & V^{(1)} + V^{(2)} + V^{(3)} + V^{(4)} \\ \hline 0 & \mathbb{I} \end{array}\right) = \left(\begin{array}{c|c} \mathbb{I} & V^{(1)} \\ \hline 0 & \mathbb{I} \end{array}\right) \left(\begin{array}{c|c} \mathbb{I} & V^{(2)} \\ \hline 0 & \mathbb{I} \end{array}\right) \left(\begin{array}{c|c} \mathbb{I} & V^{(3)} \\ \hline 0 & \mathbb{I} \end{array}\right) \left(\begin{array}{c|c} \mathbb{I} & V^{(4)} \\ \hline 0 & \mathbb{I} \end{array}\right), \tag{A13}$$

where

$$V = \begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} \\ A_{3,1} & A_{3,2} & A_{3,3} & A_{3,4} \\ A_{4,1} & A_{4,2} & A_{4,3} & A_{4,4} \end{pmatrix} = \underbrace{\begin{pmatrix} A_{1,1} & 0 & 0 & 0 \\ 0 & A_{2,2} & 0 & 0 \\ 0 & 0 & A_{3,3} & 0 \\ 0 & 0 & 0 & A_{4,4} \end{pmatrix}}_{V^{(1)}} + \underbrace{\begin{pmatrix} 0 & A_{1,2} & 0 & 0 \\ A_{2,1} & 0 & 0 & 0 \\ 0 & 0 & 0 & A_{3,4} \\ 0 & 0 & A_{4,3} & 0 \end{pmatrix}}_{V^{(2)}} + \underbrace{\begin{pmatrix} 0 & 0 & A_{1,3} & 0 \\ 0 & 0 & 0 & A_{2,4} \\ A_{3,1} & 0 & 0 & 0 \\ 0 & A_{4,2} & 0 & 0 \end{pmatrix}}_{V^{(3)}} + \underbrace{\begin{pmatrix} 0 & 0 & 0 & A_{1,4} \\ 0 & 0 & A_{2,3} & 0 \\ 0 & A_{3,2} & 0 & 0 \\ A_{4,1} & 0 & 0 & 0 \end{pmatrix}}_{V^{(4)}}. \tag{A14}$$

Each $V^{(j)}$ round is implementable with fully parallelized unidirectional $\overline{\mathrm{CNOT}}$ circuits between disjoint pairs of codeblocks. In general, a $V$ that is $2^b k \times 2^b k$ in size requires $2^b$ rounds in its cyclic scheduling.

In the decomposition of Eq. (A12), $U_0$ is $2^{a-1}k \times 2^{a-1}k$ in size and therefore its cyclic scheduling takes $2^{a-1}$ rounds. Each round contains only lower- or upper-block-triangular submatrices, corresponding to unidirectional $\overline{\mathrm{CNOT}}$ circuits between disjoint pairs of codeblocks, and so takes depth at most two to implement by Lemma 2 with parallelization. For the next term, the submatrices $U_{1,1}, U_{1,2}$ are $2^{a-2}k \times 2^{a-2}k$ in size each and their cyclic schedules can be performed in parallel; only $2^{a-2}$ rounds are needed in their schedules. Continuing this reasoning to all the upper-block-triangular matrices in Eq. (A12), we find that the total number of rounds needed is at most $2^{a-1} + 2^{a-2} + \cdots + 1 = 2^a - 1$, for a total depth of at most $2(2^a - 1)$. The same is true for the lower-block-triangular matrices. Therefore, the overall depth required is at most $4(2^a - 1)$ up to a residual permutation of logical qubits $P$.

When the desired $\overline{\mathrm{CNOT}}$ circuit contains only unidirectional $\overline{\mathrm{CNOT}}$ gates, $X$ is itself either lower- or upper-block-triangular. Then the decomposition of Eq. (A12) contains only either the lower- or upper-block-triangular half, and there is no permutation $P$ present. Therefore, the overall depth required is at most $2(2^a - 1)$ whilst maintaining ordering of the logical qubits. □

While the worst-case depth up to residual permutation $P$ is $k$-independent, in cases where the permutation must be performed physically, $P$ could take a worst-case depth proportional to $k$ to implement. This has been stated in Remark 1 of the main text, reproduced below for reference.

**Remark 1.** *The residual permutations of logical qubits in Theorem 1 incur zero cost when compiled away. In cases where they are necessary to be performed physically, they require depth at most $8k + 8$ for any number of codeblocks of an $[\![n,k,d]\!]$ CSS phantom code.*

*Proof.* We first decompose the permutation of logical qubits into two layers of involutions [23, Thm. XI.4]. An involution layer can be understood as a product of $\overline{\text{SWAP}}$s on disjoint supports that can be performed in parallel. The $\overline{\text{SWAP}}$s that take place within the same codeblock can be compiled away for phantom codes (i.e. by writing $\overline{\text{SWAP}}_{ij} = \overline{\text{CNOT}}_{ij}\overline{\text{CNOT}}_{ji}\overline{\text{CNOT}}_{ij}$ and the qubit permutations implementing the in-block $\overline{\text{CNOT}}$ gates are pulled through the physical circuit), so we henceforth deal only with interblock $\overline{\text{SWAP}}$s.

For each involution layer, we construct a graph where each codeblock serves as a vertex. We add an edge between two vertices if there exists interblock $\overline{\text{SWAP}}$ gates between the two corresponding codeblocks[8]. This graph has degree at most $k$, hence an edge-colouring of the graph requires at most $k+1$ colours by Vizing's theorem. An edge between two vertices represent a $\overline{\text{SWAP}}$ circuit between two codeblocks; edges labelled by the same colour can be performed in parallel. For each colour, we use Lemma 3 to exactly implement the $\overline{\text{SWAP}}$ circuits in depth four.

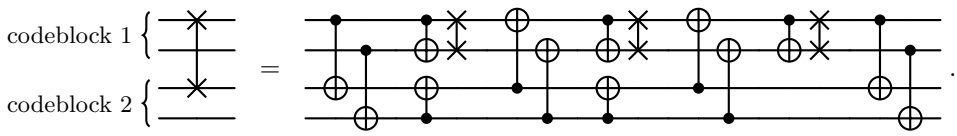
Therefore, the total depth required for an arbitrary permutation is $2 \times (k+1) \times 4 = 8k+8$; this number is independent of the number of codeblocks. □

**Lemma 3.** *Arbitrary $\overline{\text{SWAP}}$ circuits between two codeblocks of a CSS phantom code can be performed in physical depth at most four.*

*Proof.* We start with a case wherein a $\overline{\text{SWAP}}$ is to be implemented between the first logical qubits of the two codeblocks. A naive construction, for example, decomposes the $\overline{\text{SWAP}}$ into three $\overline{\text{CNOT}}$s and implements each using two transversal $\overline{\text{CNOT}}$s, for a total depth of six. Here we show that a depth-four implementation is possible. We rely on the following decomposition

$$\left(\begin{array}{cc|cc} 0&0&1&0\\ 0&1&0&0\\ \hline 1&0&0&0\\ 0&0&0&1 \end{array}\right) = \underbrace{\left(\begin{array}{cc|cc} 1&&1&\\ &1&&1\\ \hline &&1&\\ &&&1 \end{array}\right)}_{\text{transversal}}\underbrace{\left(\begin{array}{cc|cc} 1&1&&\\ 1&0&&\\ \hline &&1&0\\ &&1&1 \end{array}\right)}_{\text{in-block}}\underbrace{\left(\begin{array}{cc|cc} 1&&&\\ &1&&\\ \hline &&1&\\ &&&1 \end{array}\right)}_{\text{transversal}}\underbrace{\left(\begin{array}{cc|cc} 1&1&&\\ 1&0&&\\ \hline &&1&0\\ &&1&1 \end{array}\right)}_{\text{in-block}}\underbrace{\left(\begin{array}{cc|cc} 1&&&\\ &1&&\\ \hline &&1&\\ &&&1 \end{array}\right)}_{\text{transversal}}\underbrace{\left(\begin{array}{cc|cc} 1&1&&\\ 1&0&&\\ \hline &&1&0\\ &&0&1 \end{array}\right)}_{\text{in-block}}\underbrace{\left(\begin{array}{cc|cc} 1&&1&\\ &1&&1\\ \hline &&1&\\ &&&1 \end{array}\right)}_{\text{transversal}}, \tag{A15}$$

which corresponds to the circuit




To avoid clutter we have expressed the above for a $k=2$ code, but the decomposition trivially generalizes for arbitrary $k$: matrices $1,3,5,7$ remain as transversal $\overline{\text{CNOT}}$s, and the in-block permutation $\overline{\text{CNOT}}$ (matrices $2,4,6$) receive additional identity block matrices for the remaining $k-2$ logical qubits.

Next, we note that swapping all $k$ logical qubits between the two codeblocks amounts to completely swapping the two codeblocks, which can be performed by relabelling the codeblocks, plus the additional freedom of depth-zero in-block $\overline{\text{SWAP}}$ operations. Therefore, up to codeblock relabelling, we only need to swap at most $\lfloor k/2 \rfloor$ logicals between the two codeblocks. Let $\ell \leq \lfloor k/2 \rfloor$ be the number of $\overline{\text{SWAP}}$s we need to implement.

Since in-block $\overline{\text{SWAP}}$s are free, without loss of generality, we can assume the action to be implemented is to swap the logical qubit $2i-1$ on the first codeblock with that on the second, for all $1 \leq i \leq \ell$. The swapping of the logical qubit $2i-1$ uses the logical qubit $2i$ as in-block ancillary aid, as appears in matrices $2,4,6$ of Eq. (A15); the logical qubits $2\ell+1,\ldots,k$ are left invariant. This finishes our constructive proof for a depth-four implementation. □

### 7. Additional permutation logical gate and phantom code properties

Here we discuss various additional properties of permutation logical gates and phantom codes.

---

[8] This is different from the multigraph construction in Ref. [23, Thm. XI.4] where an edge is added for *every* interblock $\overline{\text{SWAP}}$ gate. A multigraph of degree $k$ could take up to $3k/2$ colours for edge colouring using Shannon's theorem. The reason that Ref. [23] uses a multigraph is that a single $\overline{\text{CNOT}}$ or $\overline{\text{SWAP}}$ gate between two of SHYPS codeblocks is of constant depth; but were they to use the simple graph construction employed here, there will in general be arbitrary $\overline{\text{CNOT}}$ or $\overline{\text{SWAP}}$ circuits between two codeblocks, which would cost $\mathcal{O}(k)$ depth in the worst case on SHYPS codes.

### a. *Structure of permutations implementing logical gates on stabilizer codes*

First, Propositions 8 and 9 address the form of permutations that implement involutory logical gates (i.e. a gate which squares to the logical identity) on stabilizer codes.

**Proposition 8** (Even period of qubit permutations implementing involutory logical gates on stabilizer codes)**.** *Any permutation of physical qubits on a stabilizer code implementing an involutory logical gate must be of even period.*

*Proof.* Suppose otherwise, and let the odd period of the qubit permutation $\pi$ be $2p+1$. We denote the unitary representation of $\pi$ as $U_\pi$, such that $\pi^{2p+1}$ is the trivial permutation and $U_\pi^{2p+1} = \mathbb{I}$. Consider any code state $|\psi\rangle$. First, as the logical gate implemented by $\pi$ is of nontrivial logical action, it must be that $U_\pi |\psi\rangle \neq e^{i\phi} |\psi\rangle$ for any global phase $\phi$. But the logical gate is involutory and square to a logical identity, so we must have $U_\pi^2 |\psi\rangle = e^{i\theta} |\psi\rangle$ for some $\theta$. Now $U_\pi^{2p+1} |\psi\rangle = U_\pi (U_\pi^2)^p |\psi\rangle = e^{ip\theta} U_\pi |\psi\rangle = |\psi\rangle$, a contradiction. So the period of $\pi$ must be even. □

**Proposition 9** (Single-layer qubit swaps for involutory logical gates on stabilizer codes)**.** *Suppose a stabilizer code does not support any nontrivial permutation of qubits with trivial logical action (i.e. implements the logical identity). Then all permutation involutory logical gates on the code are implementable through period-two permutations (i.e. qubit involutions), which correspond to single parallelizable layers of qubit swaps.*

*Proof.* Suppose there is an involutory logical gate on the code whose implementation requires a qubit permutation $\pi$ of period $p > 2$, such that $\pi^2$ is a nontrivial permutation. But the logical gate is involutory, so $\pi^2$ implements the logical identity but is a nontrivial permutation of physical qubits, a contradiction to the premise. □

Propositions 8 and 9 apply to permutation $\overline{\text{CNOT}}$ and $\overline{\text{SWAP}}$ gates, which are involutory, on phantom codes. As we propose compiling away all physical qubit permutations by pulling them through the physical circuit, the structure of the permutations does not typically matter in practical use. However, in cases where one considers performing some of the qubit permutations physically, perhaps due to circuit compilation or hardware constraints, permutations that take the form of depth-one parallelizable qubit swaps as discussed in Proposition 9 may be of interest.

**Remark 4.** *There exist stabilizer codes for which period $>2$ qubit permutations are required to implement an involutory logical gate set.*

*Proof.* We give the smallest example of a CSS code that requires period $>2$ qubit permutations to implement a $\overline{\text{CNOT}}$ gate. Restricting to period-two qubit permutations entails the loss of the $\overline{\text{CNOT}}$. This minimal example is an $[\![10, 2, 3]\!]$ code defined by stabilizer generator matrices:

$$H_{\text{x}} = \begin{bmatrix} 1&1&1&1&0&0&0&0&0&0 \\ 0&0&1&0&1&1&1&0&0&0 \\ 0&1&0&0&1&0&0&1&1&0 \\ 0&0&0&1&0&0&1&1&0&1 \end{bmatrix}, \qquad H_{\text{z}} = \begin{bmatrix} 1&1&0&0&1&1&0&0&0&0 \\ 0&0&1&1&1&0&0&1&0&0 \\ 0&0&0&0&0&1&1&1&1&0 \\ 0&1&0&1&0&0&0&0&1&1 \end{bmatrix}. \tag{A16}$$

This is the smallest-$n$ example found via exhaustive code enumeration; determination of the permutation logical gate set of the code was by SAT solving (see App. C). Our search also uncovered $[\![n, k]\!]$ CSS codes for which restricting to period-two qubit permutations incurs a reduction in the size of their permutation logical gate set for all $n = 11, 12, 13, 14$ and $k = 2, 3$, and all $n = 12, 13, 14$ for $k = 4$; presumably examples likewise exist at higher $k$ but we did not cover those $k$ in our search. □

### b. *Permutation $\overline{\text{CNOT}}$ gate-set-preserving logical basis changes on phantom codes*

A phantom code realizes a complete set of individually addressable $\overline{\text{CNOT}}$ logical gates via qubit permutations for some logical basis. A natural question to be asked is what other logical bases can be used on the code so that the *same set* of qubit permutations likewise *generate* a complete set of individually addressable $\overline{\text{CNOT}}$ gates.

**Definition 4** (Phantom-gate-set-preserving logical basis change)**.** *Let $M$ be a set of physical qubit permutations that realize the logical gate set $G = \{\overline{\text{CNOT}}_{ab} : (a, b) \in [k]^2, a \neq b\}$ on a phantom stabilizer code for some choice of logical basis, and $\mathcal{M}$ be the group generated by $M$. A phantom-gate-preserving logical basis change is a transformation of the logical basis, so that a set of permutations $M' \subset \mathcal{M}$ still realizes $G$ on the code in the transformed logical basis.*

Note that, in Definition 4, the logical action of each permutation in $M$ may be different in the transformed logical basis—i.e. a permutation may now implement a different $\overline{\text{CNOT}}$ gate or a product of $\overline{\text{CNOT}}$ gates—but we demand that the complete set of $\overline{\text{CNOT}}$ gates on the code can still be implemented by products of permutations in $M$.

We give two immediate remarks. First, phantom-gate-set-preserving logical basis changes, by their nature, form a group. Second, the notion of a phantom-gate-set-preserving logical basis change is stricter than that of a *phantomness-preserving* logical basis change—a logical basis change of the first type is also necessarily one of the second type, but not the other way around. The reason is that a code that is phantom in a logical basis could still be phantom in a different logical basis using a completely *different* set of qubit permutations, not in the span of the original, to implement the $\overline{\text{CNOT}}$ gate set, but this case is not considered in Definition 4. To re-iterate the point: phantom-gate-set-preserving logical basis changes as defined in Definition 4 suffice to preserve the phantom property of a stabilizer code but may not be necessary.

Definition 4 places nontrivial restrictions on the admissible structure of logical basis change circuits, especially in regard to Hadamard-type gates in the circuits. We provide a definitive characterization of such circuits in Theorem 4.

**Theorem 4.** *Phantom-gate-set-preserving logical basis-change circuits are generated by the following logical gates: (a) for $k = 2$ codes, $\overline{H}^{\otimes 2}$, $\overline{\text{CZ}}$, and $\overline{\text{CNOT}}$ gates; or (b) for $k \geq 3$ codes, $H^{\otimes k}$ and $\overline{\text{CNOT}}$ gates.*

*Proof.* Let the logical basis change be $R \in \text{Sp}(2k, \mathbb{F}_2)$ in the symplectic representation (see App. A 1). Recall that $R$ being a symplectic matrix means $R^\mathsf{T}\Omega R = \Omega$. Let $Q = \binom{Q_\text{x}}{Q_\text{z}}$ where $Q_\text{x}, Q_\text{z} \in \mathbb{F}_2^{k\times 2n}$ be a logical basis in which the code is phantom, and let $M$ be a set of permutations that implements the complete set of individually addressable $\overline{\text{CNOT}}$ gates on the code. Denote by $\mathcal{M}$ the group generated by $M$. We assess whether there exists $M' \subset \mathcal{M}$ that implements the complete set of $\overline{\text{CNOT}}$s in the transformed logical basis $Q' = RQ$. We consider different gate generators for $R$:

- $R$ is generated by $\overline{\text{CNOT}}$ gates. This case subsumes the setting of Proposition 5. Recall that CNOT circuits on $k$ qubits take the form $\text{diag}(A, A^{-\mathsf{T}})$ for $A \in \text{GL}(k, \mathbb{F}_2)$ in the symplectic representation (see App. A 1). Thus $R = \text{diag}(C, C^{-\mathsf{T}})$ for some $C \in \text{GL}(k, \mathbb{F}_2)$. Let us consider a permutation $P^{(i)} \in M$, which implements a $\overline{\text{CNOT}}$ gate described by $\text{diag}(F_i, F_i^{-\mathsf{T}})$ in the $Q$ logical basis. By Eq. (A5a) of Proposition 1, this means

$$Q\big(P^{(i)} \oplus P^{(i)}\big)\Omega Q^\mathsf{T} = \begin{bmatrix} F_i & \\ & F_i^{-\mathsf{T}} \end{bmatrix}\Omega. \tag{A17}$$

  Now, in the $Q'$ logical basis,

$$\begin{aligned} Q'\big(P^{(i)} \oplus P^{(i)}\big)\Omega Q'^\mathsf{T} &= RQ\big(P^{(i)} \oplus P^{(i)}\big)\Omega Q^\mathsf{T} R^\mathsf{T} \\ &= \begin{bmatrix} C & \\ & C^{-\mathsf{T}} \end{bmatrix}\begin{bmatrix} F_i & \\ & F_i^{-\mathsf{T}} \end{bmatrix}\Omega\begin{bmatrix} C^\mathsf{T} & \\ & C^{-1} \end{bmatrix} = \begin{bmatrix} CF_iC^{-1} & \\ & \big(CF_iC^{-1}\big)^{-\mathsf{T}} \end{bmatrix}\Omega. \end{aligned} \tag{A18}$$

  As the code is phantom in the $Q$ logical basis, $\text{span}(\{F_i\})$ for all the permutations in $M$ is precisely $\text{GL}(k, \mathbb{F}_2)$, the space of $\overline{\text{CNOT}}$ circuits on the $k$ logical qubits. Conjugating $\text{GL}(k, \mathbb{F}_2)$ by a matrix $C \in \text{GL}(k, \mathbb{F}_2)$ does not change the group. It must therefore be possible to implement a complete set of individually addressable $\overline{\text{CNOT}}$ gates in the $Q'$ logical basis using some $M' \subset \mathcal{M}$, whose elements are generically products of permutations in $M$.

- In the same way as above, we can check that $R = \left[\begin{smallmatrix} \mathbb{I} \\ \mathbb{I} \end{smallmatrix}\right]$ corresponding to a $H^{\otimes k}$ circuit, and $R = \left[\begin{smallmatrix} \mathbb{I} & X \\ \mathbb{I} & 0 \end{smallmatrix}\right]$ for $k = 2$ where $\mathbb{I}, X$ are $2 \times 2$ Pauli matrices, are phantom-gate-set-preserving.

As phantom-gate-set-preserving logical basis changes form a group, all circuits generated by the gates above are phantom-gate-set-preserving. To show that only these circuits are admissible, consider an arbitrary basis change $R = \left[\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right]$. Substituting into Eq. (A18),

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} F_i & \\ & F_i^{-\mathsf{T}} \end{bmatrix}\Omega\begin{bmatrix} A^\mathsf{T} & C^\mathsf{T} \\ B^\mathsf{T} & D^\mathsf{T} \end{bmatrix} = \begin{bmatrix} AF_iB^\mathsf{T} + BF_i^{-\mathsf{T}}A^\mathsf{T} & AF_iD^\mathsf{T} + BF_i^{-\mathsf{T}}C^\mathsf{T} \\ CF_iB^\mathsf{T} + DF_i^{-\mathsf{T}}A^\mathsf{T} & CF_iD^\mathsf{T} + DF_i^{-\mathsf{T}}C^\mathsf{T} \end{bmatrix}, \tag{A19}$$

and we require this logical action to take the form

$$\begin{bmatrix} F_{\sigma(i)} & \\ & F_{\sigma(i)}^{-\mathsf{T}} \end{bmatrix}\Omega = \begin{bmatrix} & F_{\sigma(i)} \\ F_{\sigma(i)}^{-\mathsf{T}} & \end{bmatrix}, \tag{A20}$$

for some $F_{\sigma(i)} \in \text{GL}(k, \mathbb{F}_2)$. We treat the $k = 2$ and $k \geq 3$ cases separately:

- For $k = 2$, we enumerate all admissible $R$. Concretely, we enumerate all $4 \times 4$ binary matrix $R$ such that $R^{\mathsf{T}}\Omega R = \Omega$ and $AEB^{\mathsf{T}} + BE^{-\mathsf{T}}A^{\mathsf{T}} = CED^{\mathsf{T}} + DE^{-\mathsf{T}}C^{\mathsf{T}} = 0$ for all $E \in \mathrm{GL}(k, \mathbb{F}_2)$. The results are 36 matrices in the six families below

$$\begin{bmatrix} E & \\ & E^{-\mathsf{T}} \end{bmatrix}, \quad \begin{bmatrix} & E \\ E^{-\mathsf{T}} & \end{bmatrix}, \quad \begin{bmatrix} E & EX \\ & E^{-\mathsf{T}} \end{bmatrix}, \quad \begin{bmatrix} & E \\ E^{-\mathsf{T}} & E^{-\mathsf{T}}X \end{bmatrix}, \quad \begin{bmatrix} E & \\ E^{-\mathsf{T}}X & E^{-\mathsf{T}} \end{bmatrix}, \quad \begin{bmatrix} E & EX \\ E^{-\mathsf{T}}X & \end{bmatrix}, \tag{A21}$$

  where $E$ is any matrix in $\mathrm{GL}(k, \mathbb{F}_2)$ in each family. These $R$ are precisely circuits generated by $H \otimes H$, CZ, and CNOT gates.

- For $k \geq 3$, we note that the matrices $A$ and $B$ satisfies the conditions in Lemma 4, to be proved later, and so do $C$ and $D$. (The full rank condition of the matrices is satisfied as $R^{\mathsf{T}}\Omega R = \Omega$.) Applying Lemma 4, and noting that $A$ and $C$ cannot be both zero, $R$ can only be

$$\begin{bmatrix} A & \\ & D \end{bmatrix}, \quad \begin{bmatrix} & B \\ C & \end{bmatrix}. \tag{A22}$$

  Further noting that $R^{\mathsf{T}}\Omega R = \Omega$, it must be that $D = A^{-\mathsf{T}}$ in the first case with $A \in \mathrm{GL}(k, \mathbb{F}_2)$, or $C = B^{-\mathsf{T}}$ with $B \in \mathrm{GL}(k, \mathbb{F}_2)$ in the second case. These correspond to a $\overline{\mathrm{CNOT}}$ circuit, and a $\overline{\mathrm{CNOT}}$ circuit followed by $\overline{H}^{\otimes k}$, respectively.

□

For example, Table III shows the canonical choice of logical basis for the $[\![4, 2, 2]\!]$ code. However, we can choose a new basis corresponding to a basis-change circuit consisting of a single $\overline{\mathrm{CZ}}$ gate: $\overline{X}_1' = \overline{X_1 Z_2}$, $\overline{X}_2' = \overline{X_2 Z_1}$, $\overline{Z}_1' = \overline{Z_1}$, and $\overline{Z}_2' = \overline{Z_2}$. Then, the same set of permutation still realizes all logical CNOT circuits. We finish by proving a lemma required in the proposition above.

**Lemma 4.** *Let $A$ and $B$ be binary matrices of size $k \times k$ with $k \geq 3$. If (1) the augmented matrix $[A \,|\, B]$ has full rank, and (2) for all $E \in \mathrm{GL}(k, \mathbb{F}_2)$, $AEB^{\mathsf{T}} + BE^{-\mathsf{T}}A^{\mathsf{T}} = 0$; then either $A = 0$ or $B = 0$.*

*Proof.* When $E = \mathbb{I}$, condition (2) leads to $BA^{\mathsf{T}} = AB^{\mathsf{T}}$. More generally, consider the elementary matrix $\mathbb{I} + E_{ij}$ for distinct indices $i \neq j$, where $E_{ij}$ has a 1 at position $(i, j)$ and zero everywhere else. In $\mathbb{F}_2$, these matrices are self-inverse: $(\mathbb{I} + E_{ij})^2 = \mathbb{I} + 2E_{ij} + E_{ij}E_{ij} = \mathbb{I}$. Therefore, condition (2) leads to $A(\mathbb{I} + E_{ij})B^{\mathsf{T}} = B(\mathbb{I} + E_{ij})^{-\mathsf{T}}A^{\mathsf{T}} = B(\mathbb{I} + E_{ji})A^{\mathsf{T}}$, i.e., $AE_{ij}B^{\mathsf{T}} = BE_{ji}A^{\mathsf{T}}$ after cancelling $BA^{\mathsf{T}} = AB^{\mathsf{T}}$. Recall that $E_{ij} = e_i e_j^{\mathsf{T}}$ where $e_i$ is a column vector with 1 at index $i$ and zero everywhere else. The condition becomes

$$b_j a_i^{\mathsf{T}} = a_i b_j^{\mathsf{T}} \qquad \forall\, i, j \in \{1, ..., k\}, \quad i \neq j, \tag{A23}$$

where $a_x$ and $b_x$ denote the $x^{\text{th}}$ columns of $A$ and $B$, respectively. Over $\mathbb{F}_2$, the equality $uv^{\mathsf{T}} = vu^{\mathsf{T}}$ holds iff $u$ and $v$ are linearly dependent, i.e., either $u = 0$, $v = 0$, or $u = v \neq 0$.

Suppose, for contradiction, that $A \neq 0$ and $B \neq 0$. Choose indices $i$ and $j$ such that $a_i \neq 0$ and $b_j \neq 0$.

- Case of $i = j$. Then Eq. (A23) tells us that $a_\ell \in \mathrm{span}(b_i) = \{0, b_i\}$ and $b_\ell \in \mathrm{span}(a_i) = \{0, a_i\}$ for every $\ell \neq i$. This means $\mathrm{colspan}([A \,|\, B]) \subseteq \mathrm{span}(a_i, b_i)$, and accordingly $\mathrm{rank}([A \,|\, B]) \leq \dim \mathrm{span}(a_i, b_i) \leq 2$, contradicting condition (1) since $k \geq 3$.

- Case of $i \neq j$. Then Eq. (A23) constrains $a_i = b_j$. Moreover, we also have that $a_p \in \mathrm{span}(b_j) = \mathrm{span}(a_i)$ for every $p \neq j$, and $b_q \in \mathrm{span}(a_i)$ for every $q \neq i$. Thus all columns of $[A \,|\, B]$ lie in $\mathrm{span}(a_i)$ except possibly the two columns $a_j$ and $b_i$. If $a_j = 0$, then $\mathrm{colspan}([A \,|\, B]) \subseteq \mathrm{span}(a_i, b_i)$ and $\mathrm{rank}([A \mid B]) \leq 2$. Otherwise, applying Eq. (A23) tells us that $b_i \in \mathrm{span}(a_j)$. In this case, $\mathrm{colspan}([A \,|\, B]) \subseteq \mathrm{span}(a_i, a_j)$ and again $\mathrm{rank}([A \,|\, B]) \leq 2$. Either way, we find a contradiction with condition (1) since $k \geq 3$.

Thus $A$ and $B$ cannot both be nonzero. But they cannot both be zero because $[A \,|\, B]$ is full-rank. So $A = 0$ or $B = 0$. □

### c. Limitations on strictly transversal logical gate sets on stabilizer codes

That a stabilizer code supports a permutation logical gate places associated inherent limitations on the strictly transversal logical gates possible on the code. This has been formalized in Theorem 3 in the main text. We reproduce the statement below for reference and provide a proof.

**Theorem 3** (No strictly transversal logical gates non-commuting with permutation logical gates)**.** *A stabilizer code supporting a logical gate $\overline{U}$ via qubit permutations can admit no strictly transversal logical gate $\overline{V}$ acting on any number of codeblocks $b$ where $\left[U^{\otimes b}, V\right] \neq 0$.*

*Proof.* Suppose otherwise, that there exists an $[\![n, k, d]\!]$ stabilizer code which supports $\overline{U}$ implemented by a qubit permutation $\pi$ and a strictly transversal $\overline{V}$ across $b$ codeblocks. As before, we denote the unitary representation of $\pi$ as $U_\pi$. The strict transversality of $\overline{V}$ entails an implementation $\overline{V} = \prod_{i=1}^{n} W_{ii\cdots i}$, where the physical gate $W$ act on corresponding qubits across the codeblocks. The inverse gate can then be performed as $\overline{V}^\dagger = \prod_{i=1}^{n} W_{ii\cdots i}^\dagger$. We note that $U_\pi^{\otimes b}$ and $\prod_{i=1}^{n} W_{ii\cdots i}$ commute, as the same qubit permutation is applied to each codeblock.

Consider the logical gate sequence $\overline{V}\overline{U}^{\otimes b}\overline{V}^\dagger$, which is implemented by $(\prod_{i=1}^{n} W_{ii\cdots i}) U_\pi^{\otimes b} (\prod_{i=1}^{n} W_{ii\cdots i}^\dagger) = U_\pi^{\otimes b} (\prod_{i=1}^{n} W_{ii\cdots i})(\prod_{i=1}^{n} W_{ii\cdots i}^\dagger) = U_\pi^{\otimes b}$ at the physical level. But now we see that $U_\pi^{\otimes b}$ on the physical qubits implements both $\overline{V}\overline{U}^{\otimes b}\overline{V}^\dagger$ and $\overline{U}^{\otimes b}$, which have distinct logical actions and is therefore a contradiction when $U^{\otimes b}$ and $V$ do not commute. □

On CSS phantom codes, which support a complete set of individually addressable permutation $\overline{\text{CNOT}}$ gates, Theorem 3 rules out a large class of strictly transversal logical gates. This includes logical Hadamard ($\overline{H}$) or phase gates ($\overline{S}$), in-block and interblock $\overline{\text{CZ}}$, and magic gates, for all or a subset of logical qubits. The sole exception is the $\overline{H}^{\otimes 2}\overline{\text{SWAP}}$ logical action, which are achievable transversally by $H^{\otimes 2}$, on some $k = 2$ codes (see e.g. codes in Table I). Essentially, Theorem 3 implies that, if other logical gates are available on phantom codes, then they cannot be strictly transversal: they must involve non-uniform operations on the qubits of the code or qubit permutations in addition to single-qubit operations.

## Appendix B: SAT preliminaries

In this work, we made extensive use of SAT solving to accomplish tasks such as checking the permutation gate sets supported by a code and thereby phantomness, and to discover phantom codes with desired parameters. To facilitate discussion of these methods, we first establish some background on SAT.

To begin, as SAT instances are defined on Boolean variables subject to Boolean logic, while our problems and constraints are expressed mathematically in $\mathbb{F}_2$, a mapping is necessary. Conventionally, one associates the elements of $\mathbb{F}_2$ with Boolean values $0 \equiv$ FALSE, $1 \equiv$ TRUE, under which addition over $\mathbb{F}_2$ corresponds to Boolean XOR, $a + b \equiv a \veebar b$, and multiplication over $\mathbb{F}_2$ corresponds to Boolean AND, $a + b \equiv a \wedge b$.

To declare $\mathbb{F}_2$ vectors or matrices as variables means to declare a Boolean variable for each of their entries. To restrict the matrix to a certain type, constraints are added into the SAT instance or structure in the matrix can be exploited. The most relevant cases are:

- $A$ to be an $n \times n$ symmetric matrix in $\mathbb{F}_2$. Then $n(n+1)/2$ Boolean variables are declared to correspond to the upper triangular part of $A$; the lower triangular part is fixed by symmetry.
- $A$ to be an $n \times n$ permutation matrix. Then $A \in \mathbb{F}_2^{n\times n}$ is declared, and the constraint that each row and column contains a single 1 entry is imposed. Optionally, to restrict the permutation to be period-2, equivalently implementable by a single layer of swaps, either the constraint $A^2 = \mathbb{I}$ is directly imposed, or $A$ is declared to be symmetric and the row and column constraints imposed.
- $A \in \mathrm{GL}(n, \mathbb{F}_2)$. Then $A \in \mathbb{F}_2^{n\times n}$ and $A' \in \mathbb{F}_2^{n\times n}$ are declared and the constraint $AA' = \mathbb{I}$ is imposed (i.e. $A'$ is the inverse of $A$). In contexts where row and column basis ordering does not matter, the diagonals of $A, A'$ can be set to be all ones.
- $A \in \mathrm{Sp}(2n, \mathbb{F}_2)$. Then $A \in \mathbb{F}_2^{2n\times 2n}$ is declared and the constraint $A\Omega A^\mathsf{T} = \Omega$ is imposed.

The mapping described above converts $\mathbb{F}_2$ linear algebraic and arithmetic constraints prevalent in our problems to clauses involving XORs and ANDs. We convert these into a conjunctive normal form formulae through standard Tseitin transformation, which SAT solvers can read as input. Three computational outcomes are then possible: SAT,

which denotes that the formulae is satisfiable and a solution can be read off; UNSAT, which denotes that the formulae is provably unsatisfiable and no solution exists; or that the solver does not finish.

We used the state-of-the-art SAT solver *kissat* [32], and the PySAT toolbox [56] to aid in problem instance construction, throughout this study. We explored also the *cryptominisat* [57] solver, which natively supports XOR clauses and thus appears suitable for problems dominated by $\mathbb{F}_2$ matrix arithmetic constraints, but found no performance advantage. We were limited to 14 days of solving time per SAT instance.

## Appendix C: Exhaustive code enumeration

We exhaustively enumerated all $n \leq 14$ CSS codes and filtered them for the phantom property. This amounted to $2.71 \times 10^{10}$ inequivalent codes in total, of which 132 305 are phantom codes with distance $d \geq 2$. A subset of these results were discussed in the main text, and we report detailed results in Tables V and VI. As our approach extends in full generality to stabilizer codes, we first describe the general procedure and then explain the optimized specialization to CSS codes.

### 1. Iteratively generating codes

Our general strategy builds on Sec. 6 of Ref. [29]. To enumerate $n$-qubit codes, we start from the trivial $[\![n, n, 1]\!]$ code whose stabilizer group is empty. Iteratively, for each $k = n-1, \ldots, 2, 1$, we build up the set of $[\![n, k]\!]$ codes from $[\![n, k+1]\!]$ seed codes by taking each seed code and looping over all ways of appending a new (nontrivial) stabilizer generator into the stabilizer group. Accordingly, a valid $s$ must be linearly independent of and must commute with the stabilizers present. Equivalently, $s$ belongs to the logical group of the $[\![n, k+1]\!]$ seed code. We loop over the $2^{2k+2} - 1$ choices of $s$ to produce $[\![n, k]\!]$ codes from each seed code. This procedure exhaustively generates every code, as every code can be obtained from some sequence of stabilizer generator additions starting from the trivial code.

### 2. Efficient equivalence classification of codes via canonical forms

As the phantom property of codes is preserved under ΠH equivalence (see Proposition 7), it suffices to store a single member of each ΠH equivalence class of codes generated during enumeration. This avoids massive redundancy that would otherwise render enumeration infeasible. The deduplication of codes can be accomplished by computing a canonical form for each code, with the property that two codes are ΠH-equivalent iff their canonical forms are identical. Then, when generating $[\![n, k]\!]$ codes in the enumeration process, we retain only ones with distinct canonical forms.

To obtain a canonical form of an $[\![n, k]\!]$ code $\mathcal{C}$, we construct its expanded Tanner graph $G[\mathcal{C}]$, which is bipartite and comprises $n$ qubit vertices and $m = 2^{n-k} - 1$ stabilizer vertices enumerating the nontrivial stabilizer group elements. The $j^{\text{th}}$ stabilizer element $\bigotimes_{i_{\text{x}} \in \mathcal{I}_{\text{x}}} X_{i_{\text{x}}} \bigotimes_{i_{\text{z}} \in \mathcal{I}_{\text{z}}} Z_{i_{\text{z}}}$ contributes $X$- ($Z$-) coloured edges connecting the $j^{\text{th}}$ stabilizer vertex to the $i_{\text{x}}^{\text{th}}$ ($i_{\text{z}}^{\text{th}}$) qubit vertex for every $i_{\text{x}} \in \mathcal{I}_{\text{x}}$ ($i_{\text{z}} \in \mathcal{I}_{\text{z}}$). We compute the canonical labeling [58] of $G[\mathcal{C}]$ respecting vertex and edge colours, which removes the freedom in vertex and edge orderings, equivalently qubit and stabilizer permutations in $\mathcal{C}$. Lastly, we extract the biadjacency matrix $A_{\mathcal{C}} \in \mathbb{Z}_3^{m \times n}$ of the canonical $G[\mathcal{C}]$ with edge colours encoded in the nonzero entries. The canonical form of $\mathcal{C}$ is then defined to be $\min(A_{\mathcal{C}}, A_{H^{\otimes n}\mathcal{C}H^{\otimes n}})$, where ordering is lexicographic, to remove the freedom of global Hadamards.

As the canonical form of a code is a bitvector, equivalently a byte array or string, equivalence checking or deduplication of a set of canonical forms is highly efficient. Empirically, the comparison cost of canonical forms is entirely negligible relative to other parts of the code enumeration procedure.

### 3. Optimized enumeration of CSS codes

CSS codes affords three key simplifications. First, the splitting of the stabilizers and logicals into pure $X$- and $Z$-sectors enables a reduction of the "branching factor" when generating codes. In particular, for each $[\![n, k+1]\!]$ seed code, one needs only loop over $2^{k+1} - 1$ choices of appending an $X$- ($Z$-) logical into the $X$- ($Z$-) stabilizer group to generate $[\![n, k]\!]$ codes. Second, it suffices to enumerate codes with stabilizer ranks $r_{\text{x}} \leq r_{\text{z}}$, as $r_{\text{x}} > r_{\text{z}}$ codes are ΠH-equivalent to them. Third, the expanded Tanner graph $G[\mathcal{C}]$ of an $[\![n, k]\!]$ CSS code is simpler in structure and can be made smaller, in particular being tripartite and comprising $n$ qubit, $m_{\text{x}} = 2^{r_{\text{x}}} - 1$ $X$-stabilizer, and $m_{\text{z}} = 2^{r_{\text{z}}} - 1$ $Z$-stabilizer

vertices. Edge colours are no longer needed, as the vertex colours of the $X$- and $Z$-stabilizer vertices suffice to encode the Pauli operator types of the stabilizers. It follows also that the canonical $G[\mathcal{C}]$ can be characterized more compactly by an ordered pair of two biadjacency matrices, $A[\mathcal{C}] = (A_{\mathrm{x}}[\mathcal{C}], A_{\mathrm{z}}[\mathcal{C}])$, where $A_{\mathrm{x}}[\mathcal{C}] \in \mathbb{F}_2^{m_{\mathrm{x}} \times n}$, $A_{\mathrm{z}}[\mathcal{C}] \in \mathbb{F}_2^{m_{\mathrm{z}} \times n}$. These simplifications make CSS code enumeration cheaper in compute time and storage costs, and accounts for our ability to reach higher $n = 14$ on CSS codes.

### 4. Technical implementation leveraging massive compute

We used the well-established *bliss* algorithm [59] to compute canonical labelings of graphs. As canonical labelling algorithms generally benefit from finer-grained colour classes, we additionally coloured stabilizer vertices according to their weight, which are invariant under qubit permutations. We massively parallelized the code enumeration process to utilize $\sim 10\,000$ cores across hundreds of nodes, over a wall-clock time of 1.5 months.

In consideration of storage and memory performance, we stored the code data generated in `HDF5` format with `Zstandard` and `BLOSC` [60] compression, which allowed random-access chunked reading of large data files. Despite compression, the enumeration process used $> 80$ TB of storage during computation, and $\sim 10$ TB for the final database. We found that at such a scale of compute, low-level factors such as networked I/O latency and bandwidth, homogeneity of hardware, and load balancing between workers become important in addition to the intrinsic compute complexity of the algorithms employed; thus we invested considerable effort to fine-tune our implementation in these respects.

### 5. Alternative approaches

For comprehensiveness, we remark on several alternative approaches to code enumeration that we found to be less performant or were difficult to employ due to intrinsic limitations.

- First, in place of the described iterative method to generate codes, one could consider enumerating stabilizer generator matrices in standard form, as in Eq. (D1). The limitation here is that there are $\mathcal{O}(2^{n^2})$ such matrices to go over to generate $[\![n, k]\!]$ codes. Even after reasonable optimizations—such as imposing stabilizer commutation, which constrains the submatrices in the standard form, and quotienting out qubit permutations, which allows restricting the columns of $\left(A_2^{\mathsf{T}} \;\; C^{\mathsf{T}} \;\; E^{\mathsf{T}}\right)^{\mathsf{T}}$ to be in nondecreasing lexicographic order—the number of matrices is astronomical: for example, $> 10^{17}$ for $[\![14, 2]\!]$ CSS codes. Conceptually, the iterative approach mitigates this blow-up by deduplicating codes up to ΠH-equivalence at every $[\![n, k]\!]$ stage of the process.

- Second, instead of deduplicating codes through canonical forms, one could consider partitioning them into equivalence classes by pairwise comparisons—in particular, checking isomorphism of their expanded Tanner graphs—and retaining a single member of each class. However, deduplicating $E$ codes in this manner requires $\mathcal{O}(E^2 \mathcal{T}_{\mathrm{iso}})$ time, in comparison to $\mathcal{O}(E \mathcal{T}_{\mathrm{can}})$ for the canonical form approach, where $\mathcal{T}_{\mathrm{iso}}$ and $\mathcal{T}_{\mathrm{can}}$ are characteristic runtimes for graph isomorphism and canonical labelling, respectively. As $\mathcal{T}_{\mathrm{iso}} \approx \mathcal{T}_{\mathrm{can}}$ and $E$ easily exceeds $10^9$, the canonical form approach is vastly faster.

- Third, instead of starting from the $[\![n, n]\!]$ trivial code and decreasing $k$ iteratively, one could instead start from the $[\![n, 0]\!]$ end of the spectrum and increasing $k$. The $[\![n, 0]\!]$ codes in this context correspond to graph states. This approach, however, requires prior knowledge of the complete set of distinct graph states for each $n$ desired (up to $n = 14$ in this work), for which, to the best of our knowledge, there exists no publicly available database. We therefore found the $[\![n, n]\!] \to [\![n, 1]\!]$ direction more straightforward.

### 6. Code distances

We computed the exact distance $d$ of each distinct code produced by brute force, which is computationally quick at the scale of codes examined here. This amounts to recording the minimum weight of $s\overline{P}$ over all choices of stabilizers $s$ and nontrivial Pauli logical operators $\overline{P}$, which totals $2^{n-k} \times (2^{2k} - 1)$ choices for an $[\![n, k]\!]$ code—but as $n \leq 14$ this calculation is fast. An optimization is available for CSS codes: the $X$- and $Z$-distances, $d_{\mathrm{x}}$ and $d_{\mathrm{z}}$, can be computed separately, where $s$ and $\overline{P}$ are restricted to be $X$- or $Z$-type; then $d = \min(d_{\mathrm{x}}, d_{\mathrm{z}})$.

### 7. SAT solving for $\overline{\text{CNOT}}$ permutation gate sets and phantomness

We analyzed each distinct CSS code for the permutation $\overline{\text{CNOT}}$ gate sets they support using Propositions 4 and 6. In particular, we checked whether each code supports a complete set of individually addressable permutation $\overline{\text{CNOT}}$ gates on $p = 2, 3, \ldots, k$ out of $k$ logical qubits. The highest case of $p = k$ corresponds to phantom codes; lower $p$ reflect partial phantomness, and we refer to codes that possess such gate sets as *weak* phantom codes. For $p < k$ cases, we declare the logical basis rotation matrices $R_{\text{x}}, R_{\text{z}} \in \text{GL}(k, \mathbb{F}_2)$ as free variables, whereas for $p = k$ this degree of freedom is unnecessary (see Proposition 5); in all cases we declare permutation matrices $\{P^{a_i b_i}\}_{i=1}^{p}$ as free variables. The lower-level encoding of these variables and constraints into a SAT problem instance, and the solving process, follows the prescription in App. B. Altogether $k - 1$ SAT problem instances ($p = 2, 3, \ldots, k$) are solved for each generated code.

### 8. Breakdown of code enumeration results

We report an exhaustive breakdown of the numbers of ΠH-inequivalent $d \geq 2$ CSS codes, weak phantom, and phantom codes with $k = 2, 3, 4$ logical qubits of code sizes $n \leq 14$, stratified by their $(n, k, d_{\text{x}}, d_{\text{z}})$ parameters, in Table V. As described above, weak phantom codes are codes that support a complete set of individually addressable permutation $\overline{\text{CNOT}}$ gates on $2 \leq p < k$ logical qubits. The stringency of the demanded gate set is reflected in the rapidly diminishing numbers of distinct codes as $p$ is increased.

For completeness, we present also an exhaustive breakdown of the numbers of ΠH-inequivalent CSS codes up to $n = 14$ at all $k$, stratified by their $(n, k, d)$ parameters, in Table VI. These results may be of independent interest. Additionally, we provide a visualization of the number of inequivalent CSS codes at each $(n, k)$ in Fig. 7, which clearly illustrates the super-exponential growth in the number of codes with $n$ that underlies the core computational challenge of code enumeration. Generally, for a fixed $n$, the number of inequivalent codes increases with $k$ up to a (super-exponentially sharp) peak at $k^*$ and decreases thereafter; this peak location $k^*$ increases with $n$.

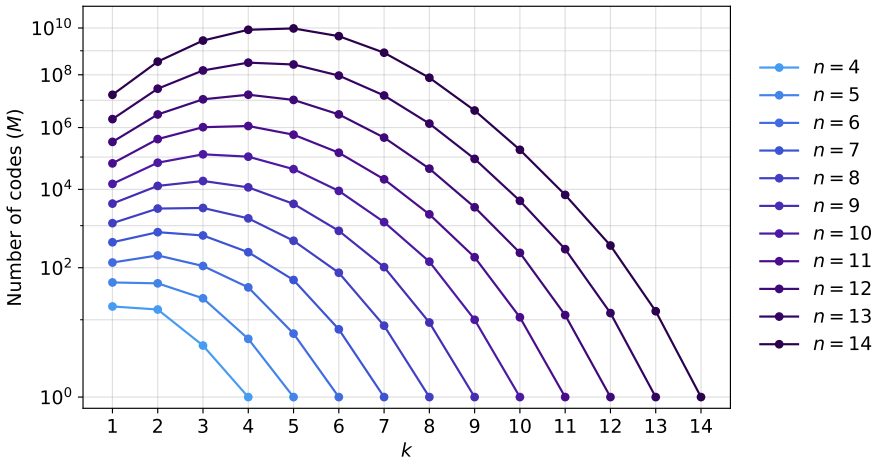



Figure 7. **Numbers of inequivalent $[\![\mathbf{n}, \mathbf{k}]\!]$ CSS codes**. The vertical axis scale is $\sqrt{\log M}$, chosen because a simple upper bound on the number of $n$-qubit CSS codes is $\mathcal{O}(2^{n^2})$. This upper bound is loose as it neglects equivalency of codes.

$k = 2$

| $n$ | $d_x$ | $d_z$ | $M$ | $M[K_1]$ | $M[K_2]$ |
|---|---|---|---|---|---|
| 4 | 2 | 2 | 1 | 1 | 1 |
| 5 | 2 | 2 | 2 | 2 | 2 |
| 6 | 2 | 2 | 15 | 15 | 9 |
| 7 | 2 | 2 | 59 | 56 | 22 |
| | | 3 | 2 | 2 | 2 |
| 8 | 2 | 2 | 352 | 284 | 61 |
| | | 3 | 20 | 17 | 8 |
| | | 4 | 2 | 2 | 2 |
| 9 | 2 | 2 | 1969 | 1219 | 164 |
| | | 3 | 205 | 140 | 33 |
| | | 4 | 11 | 11 | 11 |
| 10 | 2 | 2 | 13229 | 5985 | 469 |
| | | 3 | 2157 | 970 | 107 |
| | | 4 | 99 | 98 | 56 |
| 10 | 3 | 3 | 5 | 2 | 0 |
| 11 | 2 | 2 | 97043 | 30362 | 1371 |
| | | 3 | 25389 | 6665 | 334 |
| | | 4 | 967 | 794 | 229 |
| | | 5 | 5 | 5 | 5 |
| 11 | 3 | 3 | 103 | 12 | 1 |
| | | 4 | 1 | 1 | 1 |
| 12 | 2 | 2 | 830830 | 175410 | 4487 |
| | | 3 | 338823 | 46711 | 1096 |
| | | 4 | 13236 | 7163 | 960 |
| | | 5 | 74 | 63 | 31 |
| | | 6 | 4 | 4 | 4 |
| 12 | 3 | 3 | 4885 | 231 | 5 |
| | | 4 | 21 | 14 | 5 |
| 12 | 4 | 4 | 1 | 1 | 1 |
| 13 | 2 | 2 | 8311808 | 1153978 | 16111 |
| | | 3 | 5219100 | 352094 | 3601 |
| | | 4 | 238551 | 66603 | 3907 |
| | | 5 | 1342 | 810 | 176 |
| | | 6 | 29 | 29 | 29 |
| 13 | 3 | 3 | 187956 | 3375 | 35 |
| | | 4 | 879 | 226 | 31 |
| 13 | 4 | 4 | 3 | 3 | 3 |
| 14 | 2 | 2 | 100151088 | 9086873 | 67104 |
| | | 3 | 94277832 | 2946714 | 12305 |
| | | 4 | 5695469 | 686332 | 16650 |
| | | 5 | 34936 | 10499 | 853 |
| | | 6 | 391 | 374 | 196 |
| 14 | 3 | 3 | 7246836 | 46369 | 161 |
| | | 4 | 75374 | 5018 | 183 |
| | | 5 | 4 | 4 | 3 |
| 14 | 4 | 4 | 65 | 54 | 31 |

$k = 3$

| $n$ | $d_x$ | $d_z$ | $M$ | $M[K_1]$ | $M[K_2]$ | $M[K_3]$ |
|---|---|---|---|---|---|---|
| 6 | 2 | 2 | 2 | 2 | 2 | 0 |
| 7 | 2 | 2 | 16 | 16 | 11 | 0 |
| | | 3 | 1 | 1 | 1 | 1 |
| 8 | 2 | 2 | 142 | 142 | 56 | 0 |
| | | 3 | 4 | 4 | 4 | 2 |
| | | 4 | 1 | 1 | 1 | 1 |
| 9 | 2 | 2 | 1342 | 1242 | 302 | 0 |
| | | 3 | 37 | 37 | 21 | 4 |
| | | 4 | 4 | 4 | 4 | 4 |
| 10 | 2 | 2 | 14173 | 10912 | 1445 | 0 |
| | | 3 | 533 | 446 | 107 | 8 |
| | | 4 | 26 | 26 | 23 | 12 |
| 11 | 2 | 2 | 174822 | 100255 | 7287 | 0 |
| | | 3 | 11304 | 5906 | 550 | 16 |
| | | 4 | 219 | 219 | 122 | 34 |
| 11 | 3 | 3 | 6 | 4 | 3 | 0 |
| 12 | 2 | 2 | 2542097 | 1000105 | 39042 | 0 |
| | | 3 | 303742 | 79083 | 3042 | 32 |
| | | 4 | 3859 | 3200 | 732 | 93 |
| | | 5 | 2 | 2 | 2 | 0 |
| 12 | 3 | 3 | 254 | 48 | 12 | 0 |
| | | 4 | 2 | 2 | 2 | 0 |
| 13 | 2 | 2 | 44258048 | 11273064 | 231983 | 0 |
| | | 3 | 9507219 | 1110857 | 17283 | 68 |
| | | 4 | 115581 | 55042 | 4454 | 260 |
| | | 5 | 74 | 73 | 36 | 0 |
| 13 | 3 | 3 | 54220 | 1987 | 89 | 0 |
| | | 4 | 37 | 25 | 14 | 0 |
| 14 | 2 | 2 | 930863172 | 149252977 | 1586674 | 0 |
| | | 3 | 342695045 | 17000963 | 105686 | 148 |
| | | 4 | 5530021 | 1096275 | 30013 | 759 |
| | | 5 | 3620 | 2192 | 338 | 1 |
| | | 6 | 20 | 20 | 15 | 2 |
| 14 | 3 | 3 | 6236568 | 59587 | 489 | 1 |
| | | 4 | 6377 | 1143 | 138 | 3 |
| 14 | 4 | 4 | 2 | 2 | 2 | 0 |

$k = 4$

| $n$ | $d_x$ | $d_z$ | $M$ | $M[K_1]$ | $M[K_2]$ | $M[K_3]$ |
|---|---|---|---|---|---|---|
| 6 | 2 | 2 | 1 | 1 | 1 | 0 |
| 7 | 2 | 2 | 3 | 3 | 3 | 0 |
| 8 | 2 | 2 | 39 | 39 | 29 | 0 |
| 9 | 2 | 2 | 370 | 365 | 191 | 1 |
| | | 3 | 1 | 1 | 1 | 0 |
| 10 | 2 | 2 | 5732 | 5316 | 1512 | 5 |
| | | 3 | 34 | 26 | 7 | 1 |
| | | 4 | 1 | 1 | 1 | 0 |
| 11 | 2 | 2 | 104987 | 82236 | 11970 | 26 |
| | | 3 | 1288 | 783 | 96 | 6 |
| | | 4 | 9 | 7 | 5 | 0 |
| 12 | 2 | 2 | 2446585 | 1430851 | 104031 | 112 |
| | | 3 | 71376 | 22840 | 1114 | 30 |
| | | 4 | 338 | 257 | 64 | 2 |
| 12 | 3 | 3 | 6 | 4 | 3 | 0 |
| 13 | 2 | 2 | 70097191 | 27711646 | 1002835 | 480 |
| | | 3 | 4674483 | 665810 | 12209 | 150 |
| | | 4 | 15455 | 7934 | 729 | 18 |
| | | 5 | 2 | 2 | 2 | 0 |
| 13 | 3 | 3 | 859 | 114 | 14 | 0 |
| 14 | 2 | 2 | 2467108696 | 620290888 | 11226575 | 2061 |
| | | 3 | 335232851 | 20178079 | 141712 | 755 |
| | | 4 | 1457061 | 346812 | 9937 | 123 |
| | | 5 | 105 | 69 | 27 | 0 |
| | | 6 | 2 | 2 | 2 | 1 |
| 14 | 3 | 3 | 975140 | 8551 | 110 | 1 |
| | | 4 | 54 | 22 | 3 | 0 |

Table V. **Number of $d \geq 2$ CSS, weakly phantom, and phantom codes at $k = 2, 3, 4$ up to $n = 14$.** Entries are the number of distinct codes of parameters $(n, k, d_x, d_z)$ up to equivalence under qubit permutations and global Hadamard duality (IIH equivalence) found through exhaustive code enumeration. Counts cover codes where $d_x \leq d_z$; the $d_x > d_z$ counterparts are duals of $d_x < d_z$ codes. $M$, $M[K_1]$, $M[K_{p\geq 2}]$ denote number of CSS codes with no other constraints, supporting at least a single permutation $\overline{\text{CNOT}}$, and supporting at least a complete set of individually addressable $\overline{\text{CNOT}}$s on $p$ logical qubits, respectively; therefore $M[K_k]$ counts the number of CSS phantom codes. $M[K_4] = 0$ for all $n \leq 14$, $k = 4$.

| $n$ | $k$ | $d$ | $M$ |
|---|---|---|---|
| 4 | 1 | 1 | 16 |
| 4 | 1 | 2 | 1 |
| 4 | 2 | 1 | 14 |
| 4 | 2 | 2 | 1 |
| 4 | 3 | 1 | 4 |
| 4 | 4 | 1 | 1 |
| 5 | 1 | 1 | 44 |
| 5 | 1 | 2 | 5 |
| 5 | 2 | 1 | 45 |
| 5 | 2 | 2 | 2 |
| 5 | 3 | 1 | 24 |
| 5 | 4 | 1 | 5 |
| 5 | 5 | 1 | 1 |
| 6 | 1 | 1 | 109 |
| 6 | 1 | 2 | 21 |
| 6 | 2 | 1 | 173 |
| 6 | 2 | 2 | 15 |
| 6 | 3 | 1 | 107 |
| 6 | 3 | 2 | 2 |
| 6 | 4 | 1 | 38 |
| 6 | 4 | 2 | 1 |
| 6 | 5 | 1 | 6 |
| 6 | 6 | 1 | 1 |
| 7 | 1 | 1 | 299 |
| 7 | 1 | 2 | 84 |
| 7 | 1 | 3 | 1 |
| 7 | 2 | 1 | 621 |
| 7 | 2 | 2 | 61 |
| 7 | 3 | 1 | 547 |
| 7 | 3 | 2 | 17 |
| 7 | 4 | 1 | 220 |
| 7 | 4 | 2 | 3 |
| 7 | 5 | 1 | 55 |
| 7 | 6 | 1 | 7 |
| 7 | 7 | 1 | 1 |
| 8 | 1 | 1 | 827 |
| 8 | 1 | 2 | 332 |
| 8 | 1 | 3 | 2 |
| 8 | 2 | 1 | 2481 |
| 8 | 2 | 2 | 374 |
| 8 | 3 | 1 | 2817 |
| 8 | 3 | 2 | 147 |
| 8 | 4 | 1 | 1514 |
| 8 | 4 | 2 | 39 |
| 8 | 5 | 1 | 415 |
| 8 | 5 | 2 | 3 |
| 8 | 6 | 1 | 77 |
| 8 | 6 | 2 | 1 |
| 8 | 7 | 1 | 8 |
| 8 | 8 | 1 | 1 |
| 9 | 1 | 1 | 2507 |
| 9 | 1 | 2 | 1385 |
| 9 | 1 | 3 | 24 |
| 9 | 2 | 1 | 10499 |
| 9 | 2 | 2 | 2185 |
| 9 | 3 | 1 | 16402 |
| 9 | 3 | 2 | 1383 |
| 9 | 4 | 1 | 11111 |
| 9 | 4 | 2 | 371 |
| 9 | 5 | 1 | 3803 |
| 9 | 5 | 2 | 46 |
| 9 | 6 | 1 | 733 |
| 9 | 6 | 2 | 4 |
| 9 | 7 | 1 | 103 |
| 9 | 8 | 1 | 9 |
| 9 | 9 | 1 | 1 |

| $n$ | $k$ | $d$ | $M$ |
|---|---|---|---|
| 10 | 1 | 1 | 8204 |
| 10 | 1 | 2 | 6182 |
| 10 | 1 | 3 | 161 |
| 10 | 2 | 1 | 50158 |
| 10 | 2 | 2 | 15485 |
| 10 | 2 | 3 | 5 |
| 10 | 3 | 1 | 107996 |
| 10 | 3 | 2 | 14732 |
| 10 | 4 | 1 | 97088 |
| 10 | 4 | 2 | 5767 |
| 10 | 5 | 1 | 40178 |
| 10 | 5 | 2 | 897 |
| 10 | 6 | 1 | 8949 |
| 10 | 6 | 2 | 91 |
| 10 | 7 | 1 | 1232 |
| 10 | 7 | 2 | 4 |
| 10 | 8 | 1 | 135 |
| 10 | 8 | 2 | 1 |
| 10 | 9 | 1 | 10 |
| 10 | 10 | 1 | 1 |
| 11 | 1 | 1 | 30289 |
| 11 | 1 | 2 | 30793 |
| 11 | 1 | 3 | 1473 |
| 11 | 2 | 1 | 273325 |
| 11 | 2 | 2 | 123404 |
| 11 | 2 | 3 | 104 |
| 11 | 3 | 1 | 843528 |
| 11 | 3 | 2 | 186345 |
| 11 | 3 | 3 | 6 |
| 11 | 4 | 1 | 1028434 |
| 11 | 4 | 2 | 106284 |
| 11 | 5 | 1 | 539400 |
| 11 | 5 | 2 | 23181 |
| 11 | 6 | 1 | 136435 |
| 11 | 6 | 2 | 2163 |
| 11 | 7 | 1 | 19895 |
| 11 | 7 | 2 | 114 |
| 11 | 8 | 1 | 1984 |
| 11 | 8 | 2 | 5 |
| 11 | 9 | 1 | 171 |
| 11 | 10 | 1 | 11 |
| 11 | 11 | 1 | 1 |
| 12 | 1 | 1 | 128355 |
| 12 | 1 | 2 | 175795 |
| 12 | 1 | 3 | 14416 |
| 12 | 1 | 4 | 2 |
| 12 | 2 | 1 | 1768453 |
| 12 | 2 | 2 | 1182967 |
| 12 | 2 | 3 | 4906 |
| 12 | 2 | 4 | 1 |
| 12 | 3 | 1 | 8031258 |
| 12 | 3 | 2 | 2849700 |
| 12 | 3 | 3 | 256 |
| 12 | 4 | 1 | 13811647 |
| 12 | 4 | 2 | 2518299 |
| 12 | 4 | 3 | 6 |
| 12 | 5 | 1 | 9523416 |
| 12 | 5 | 2 | 798474 |
| 12 | 6 | 1 | 2877957 |
| 12 | 6 | 2 | 96580 |
| 12 | 7 | 1 | 441457 |
| 12 | 7 | 2 | 5158 |
| 12 | 8 | 1 | 42402 |
| 12 | 8 | 2 | 211 |
| 12 | 9 | 1 | 3093 |
| 12 | 9 | 2 | 6 |
| 12 | 10 | 1 | 215 |
| 12 | 10 | 2 | 1 |
| 12 | 11 | 1 | 12 |
| 12 | 12 | 1 | 1 |

| $n$ | $k$ | $d$ | $M$ |
|---|---|---|---|
| 13 | 1 | 1 | 649360 |
| 13 | 1 | 2 | 1191186 |
| 13 | 1 | 3 | 165529 |
| 13 | 1 | 4 | 21 |
| 13 | 2 | 1 | 13991607 |
| 13 | 2 | 2 | 13770830 |
| 13 | 2 | 3 | 188835 |
| 13 | 2 | 4 | 3 |
| 13 | 3 | 1 | 96173113 |
| 13 | 3 | 2 | 53880922 |
| 13 | 3 | 3 | 54257 |
| 13 | 4 | 1 | 240548389 |
| 13 | 4 | 2 | 74787131 |
| 13 | 4 | 3 | 859 |
| 13 | 5 | 1 | 228181293 |
| 13 | 5 | 2 | 36098973 |
| 13 | 5 | 3 | 5 |
| 13 | 6 | 1 | 86904718 |
| 13 | 6 | 2 | 6099111 |
| 13 | 7 | 1 | 14864068 |
| 13 | 7 | 2 | 392972 |
| 13 | 8 | 1 | 1371561 |
| 13 | 8 | 2 | 11931 |
| 13 | 9 | 1 | 86886 |
| 13 | 9 | 2 | 267 |
| 13 | 10 | 1 | 4684 |
| 13 | 10 | 2 | 6 |
| 13 | 11 | 1 | 263 |
| 13 | 12 | 1 | 13 |
| 13 | 13 | 1 | 1 |
| 14 | 1 | 1 | 4060337 |
| 14 | 1 | 2 | 9897944 |
| 14 | 1 | 3 | 2254065 |
| 14 | 1 | 4 | 539 |
| 14 | 2 | 1 | 140083911 |
| 14 | 2 | 2 | 200159716 |
| 14 | 2 | 3 | 7322214 |
| 14 | 2 | 4 | 65 |
| 14 | 3 | 1 | 1482677292 |
| 14 | 3 | 2 | 1279091878 |
| 14 | 3 | 3 | 6242945 |
| 14 | 3 | 4 | 2 |
| 14 | 4 | 1 | 5513412044 |
| 14 | 4 | 2 | 2803798715 |
| 14 | 4 | 3 | 975194 |
| 14 | 5 | 1 | 7453583787 |
| 14 | 5 | 2 | 2094974672 |
| 14 | 5 | 3 | 2118 |
| 14 | 6 | 1 | 3792773348 |
| 14 | 6 | 2 | 529029086 |
| 14 | 6 | 3 | 21 |
| 14 | 7 | 1 | 780207939 |
| 14 | 7 | 2 | 46309837 |
| 14 | 8 | 1 | 74486954 |
| 14 | 8 | 2 | 1582818 |
| 14 | 9 | 1 | 4110070 |
| 14 | 9 | 2 | 27189 |
| 14 | 10 | 1 | 172435 |
| 14 | 10 | 2 | 447 |
| 14 | 11 | 1 | 6918 |
| 14 | 11 | 2 | 7 |
| 14 | 12 | 1 | 320 |
| 14 | 12 | 2 | 1 |
| 14 | 13 | 1 | 14 |
| 14 | 14 | 1 | 1 |

Table VI. **Number of CSS codes up to $n = 14$.** Entries are the number of distinct CSS codes of parameters $(n, k, d)$ up to equivalence under qubit permutations and global Hadamard duality (ΠH equivalence) found through exhaustive code enumeration. We include $d = 1$ codes for completeness but they do not support error detection or correction capability.

## Appendix D: SAT-based code discovery

While code enumeration provides an exhaustive coverage of $n$-qubit phantom codes, the maximum $n$ reachable is computationally limited. To explore higher $n$, we employed a code discovery method based on Boolean constraint satisfaction (i.e. SAT solving) that produces phantom codes of desired $[\![n, k, d]\!]$ parameters in an automated fashion.

We note that our work is not the first to use SAT-based methods for quantum code discovery. Ref. [61] employs SAT to search for sparse quantum codes. By contrast, our approach is the first to search for codes via their gate sets.

### 1. Review of standard form of stabilizer generator matrices

To start, we review Gottesman's standard form for the stabilizer generator matrix of an $[\![n, k]\!]$ stabilizer code in symplectic form [62],

$$H = \left(\begin{array}{ccc|ccc} \mathbb{I} & A_1 & A_2 & B & 0 & C \\ 0 & 0 & 0 & D & \mathbb{I} & E \end{array}\right), \tag{D1}$$

which is parametrized by a rank $0 \leq r \leq n-k$. Letting $t \coloneqq n-r-k$, the submatrices $A_1 \in \mathbb{F}_2^{r\times t}$, $A_2 \in \mathbb{F}_2^{r\times k}$, $B \in \mathbb{F}_2^{r\times r}$, $C \in \mathbb{F}_2^{r\times k}$, $D \in \mathbb{F}_2^{t\times r}$, and $E \in \mathbb{F}_2^{t\times k}$. Each row of $H$ represents a stabilizer generator of the code in symplectic form (see App. A 1). For $H$ to represent a valid code, the stabilizers must commute ($H\Omega H^{\mathsf{T}} = 0$), which amounts to the constraints

$$D^{\mathsf{T}} = A_1 + A_2 E^{\mathsf{T}}, \tag{D2a}$$

$$B + C A_2^{\mathsf{T}} \quad \text{symmetric.} \tag{D2b}$$

Conveniently, a valid logical basis can also be written,

$$L_{\mathrm{x}} = \left(\begin{array}{ccc|ccc} 0 & E^{\mathrm{T}} & \mathbb{I} & C^{\mathrm{T}} & 0 & 0 \end{array}\right), \qquad L_{\mathrm{z}} = \left(\begin{array}{ccc|ccc} 0 & 0 & 0 & A_2^{\mathrm{T}} & 0 & \mathbb{I} \end{array}\right), \tag{D3}$$

with the anticommutation $L_{\mathrm{x}}\Omega L_{\mathrm{z}}^{\mathsf{T}} = \mathbb{I}$ guaranteed. Every qubit stabilizer code can be represented in standard form, thus taking $H, L_{\mathrm{x}}, L_{\mathrm{z}}$ in such a format for code discovery is without loss of generality.

### 2. SAT problem formulation for discovery of stabilizer phantom codes

Each SAT problem instance determines if there exists an $[\![n, r, k, d]\!]$ code that is phantom, and in positive cases, reports the solution. Enumerating the parameters $[\![n, r, k, d]\!]$ therefore discovers phantom codes. To build a SAT problem instance, we declare the submatrices of the standard form, Eq. (D1), as free variables, and impose the commutation constraints of Eq. (D2) to ensure a valid stabilizer code. We then use Proposition 2 to constrain the code to be phantom. In particular, we declare the logical basis change $R \in \mathrm{Sp}(2k, \mathbb{F}_2)$ and permutation matrices $\{P^{a_i b_i}\}_{i=1}^{p}$ as free variables, and impose the gate set constraints therein.

Imposing the distance-$d$ constraint on the code is more subtle. Our approach is to impose distance lower- and upper-bound constraints simultaneously. To have distance at least $d$ means that every Pauli error of weight at most $d-1$ must either generate nontrivial syndrome or is itself a stabilizer. This requirement can be expressed as

$$\left(\bigvee_{i=1}^{n-k} \left(H\Omega e^{\mathsf{T}}\right)_i\right) \vee \left[\left(\neg \bigvee_{i=1}^{n-k} \left(H\Omega e^{\mathsf{T}}\right)_i\right) \wedge \left(\neg \bigvee_{i=1}^{2k} \left(L\Omega e^{\mathsf{T}}\right)_i\right)\right], \tag{D4}$$

for every $e \in \mathcal{E}_{\geq} \coloneqq \{e \in \mathcal{P}_n : \mathrm{wt}(e) \leq d-1\}$ in binary symplectic representation. By applying De Morgan's laws and noting that the set of vectors $e$ is invariant under multiplication by $\Omega$, this can be simplified to

$$\left(\bigvee_{i=1}^{n-k} \left(He^{\mathsf{T}}\right)_i\right) \vee \left(\neg \bigvee_{i=1}^{2k} \left(Le^{\mathsf{T}}\right)_i\right), \tag{D5}$$

for every $e \in \mathcal{E}_{\geq}$—which we add as constraints. Next, to have distance at most $d$ means that there must exist a weight-$d$ Pauli error that generates no syndrome and is not a stabilizer. This can be expressed as

$$\bigvee_{e\in\mathcal{E}_{\leq}} \left\{\left(\neg \bigvee_{i=1}^{n-k} \left(H\Omega e^{\mathsf{T}}\right)_i\right) \wedge \left[\left(\bigvee_{i=1}^{n-k} \left(H\Omega e^{\mathsf{T}}\right)_i\right) \vee \left(\bigvee_{i=1}^{2k} \left(L\Omega e^{\mathsf{T}}\right)_i\right)\right]\right\}, \tag{D6}$$

where $\mathcal{E}_{\leq} \coloneqq \{e \in \mathcal{P}_n : \mathrm{wt}(e) = d\}$ in binary symplectic representation. Likewise, this can be simplified to

$$\bigvee_{e \in \mathcal{E}_{\leq}} \left[ \left( \neg \bigvee_{i=1}^{n-k} \left( H e^{\mathsf{T}} \right)_i \right) \wedge \left( \bigvee_{i=1}^{2k} \left( L e^{\mathsf{T}} \right)_i \right) \right], \tag{D7}$$

which we add as a constraint. Thus, Eqs. (D5) and (D7) together enforces that the distance of the code is exactly $d$. The lower-level encoding of these variables and constraints into a form suitable for treatment by a SAT solver follows the prescription in App. B.

### 3. SAT problem formulation for discovery of CSS phantom codes

Specializing the code discovery strategy to CSS codes affords key simplifications. First, $B = C = 0$ are fixed in the standard form of stabilizer generator matrices [Eq. (D1)] and the "half-symplectic" binary representations $H_{\mathrm{x}} = \begin{pmatrix} \mathbb{I} & A_1 & A_2 \end{pmatrix}$, $H_{\mathrm{z}} = \begin{pmatrix} D & \mathbb{I} & E \end{pmatrix}$, $L_{\mathrm{x}} = \begin{pmatrix} 0 & E^{\mathsf{T}} & \mathbb{I} \end{pmatrix}$, $L_{\mathrm{z}} = \begin{pmatrix} A_2^{\mathsf{T}} & 0 & \mathbb{I} \end{pmatrix}$ can be used. There is accordingly no need to declare $B$ and $C$ as free variables and only Eq. (D2a) applies as a commutation constraint. Second, we use the simpler Proposition 6 to constrain the code to be phantom. In particular, there is no need for arbitrary logical basis rotations, and we declare only the permutation matrices $\{P^{a_i b_i}\}_{i=1}^{p}$ as variables.

Third, we impose distance constraints on both the $X$- and $Z$-logical sectors separately, which also affords us the flexibility of individually specifying the desired $X$- and $Z$-distances $d_{\mathrm{x}}$ and $d_{\mathrm{z}}$ of the code. Each SAT problem instance therefore determines if there exists an $[\![n, r, k, (d_{\mathrm{x}}, d_{\mathrm{z}})]\!]$ CSS code that is phantom, and in positive cases, reports the solution. Explicitly, to enforce distance $d_\mu$ for $\mu \in \{X, Z\}$, we add the lower- and upper-bound constraints

$$\left( \bigvee_{i=1}^{r_\nu} \left( H_\nu e^{\mathsf{T}} \right)_i \right) \vee \left( \neg \bigvee_{i=1}^{k} \left( L_\nu e^{\mathsf{T}} \right)_i \right) \quad \forall e \in \mathcal{E}^{\mu}_{\geq}, \tag{D8a}$$

$$\bigvee_{e \in \mathcal{E}^{\mu}_{\leq}} \left[ \left( \neg \bigvee_{i=1}^{r_\nu} \left( H_\nu e^{\mathsf{T}} \right)_i \right) \wedge \left( \bigvee_{i=1}^{k} \left( L_\nu e^{\mathsf{T}} \right)_i \right) \right], \tag{D8b}$$

where $\nu$ is the conjugate sector to $\mu$, and the vector sets $\mathcal{E}^{\mu}_{\geq} \coloneqq \{e \in \mathbb{F}_2^n : \mathrm{wt}(e) \leq d_\mu - 1\}$ and $\mathcal{E}^{\mu}_{\leq} \coloneqq \{e \in \mathbb{F}_2^n : \mathrm{wt}(e) = d_\mu\}$.

We report results on the minimal block lengths of $k = 2, 3$ CSS phantom codes as certified through SAT solving in Table VII, which is an expanded version of the smaller Table II of the main text. Examples of codes found by SAT that possess exceptional properties, such as high encoding rate, distance, or large logical gate sets are highlighted in Table I of the main text.

### 4. SAT problem formulation for discovery of stabilizer and CSS codes without gate set constraints

In Table II of the main text, we reported minimal block lengths of CSS codes in general. These were obtained by running the SAT solver for code discovery without gate set constraints, which turns the tool into one for generating codes with specified parameters (or certifying that none exists). Specifically, the problem formulation in App. D 2 for stabilizer codes or App. D 3 for CSS codes are the same, including the distance constraints; but the permutation gate set or phantomness constraints of Proposition 2 or Proposition 6 are excluded.

## Appendix E: Phantom quantum Reed–Muller codes

Our numerical methods are tractable only for $k \leq 4$. To reach higher $k$, we construct an infinite family of CSS phantom codes, starting from quantum Reed–Muller (qRM) codes. By fixing selected logical qubits to $|\overline{0}\rangle$ or $|\overline{+}\rangle$, we promote the corresponding logical operators to $Z$- or $X$-type stabilizers, respectively. The resulting codes have parameters $[\![2^m, m-l+1, \min(2^{m-l}, 2^l)]\!]$. This construction is naturally described using the polynomial representation of qRM codes, which we briefly review below.

| $k$ | $d_x$ \ $d_z$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 4 | 7 | 8 | 11 | 12 | 15 | 16 | 18–19 | 20 |
| | 3 | | 11 | 11 | 14 | 15 | $\geq 17$ | $\geq 19$ | $\geq 20$ | $\geq 21$ |
| | 4 | | | 12 | 15 | 16 | $\geq 19$ | $\geq 20$ | $\geq 21$ | $\geq 22$ |
| | 5 | | | | 18 | 19 | $\geq 20$ | $\geq 21$ | $\geq 22$ | $\geq 22$ |
| | 6 | | | | | 20 | $\geq 21$ | $\geq 22$ | | |
| | 7 | | | | | | $\geq 23$ | $\geq 24$ | | |
| | 8 | | | | | | | $\geq 24$ | | |
| 3 | 2 | $\geq 15$ | 7 | 8 | 14 | 14 | 15 | 16 | 19–21 | 20–22 |
| | 3 | | 14 | 14 | $\geq 15$ | $\geq 16$ | 18–21 | $\geq 19$ | $\geq 21$ | $\geq 21$ |
| | 4 | | | 15 | $\geq 16$ | $\geq 17$ | | | | |
| | 5 | | | | $\geq 19$ | $\geq 20$ | | | | |
| | 6 | | | | $\geq 19$ | | | | | |

Table VII. **Smallest $n$ for CSS phantom codes with $k = 2$–3**. Entries show the minimal $n$ known for each $(d_x, d_z)$. Tables are symmetric along the diagonal since each code has a dual with $d_x$ and $d_z$ exchanged, assessed by deforming the code with Hadamards on every qubit ($H^{\otimes n}$); we therefore omit the $d_x > d_z$ entries (shaded grey). All $n \leq 14$ cells are from exhaustive code enumeration and the rest are from SAT code discovery. Ranges of $n$, for example $n_1$–$n_2$, are shown when SAT discovers a phantom code at size $n_2$ but cannot prove that no phantom code exists at size $n_1$ within time limits; lower bounds, for example $\geq n_b$, are shown when SAT code discovery has proven that no phantom code exists at sizes $< n_b$ but has not yet found a concrete solution. Time limit for SAT solving was constrained to be 14 days.

### 1. Review of Reed–Muller codes and polynomial formalism

First, we briefly review classical Reed–Muller codes. Reed–Muller (RM) codes are a family of classical linear codes denoted RM$(r, m)$, parameterized by the order $r$ and block length $2^m$, with code parameters $[2^m, \sum_{i=0}^{r} \binom{m}{i}, 2^{m-r}]$. The generators of RM codes can be constructed using the *polynomial formalism* [33, Chapter 13]. Since the dual of RM codes are themselves also RM codes, they can be constructed with the same formalism.

#### a. *Polynomial formalism*

Consider $m$ variables over the binary field $x_1, x_2, \ldots, x_m \in \mathbb{F}_2$. A *monomial* is a product of these variables that uses each variable at most once: for example, $x_1x_3$, $x_2x_4x_5$, or 1. Here 1 is the constant monomial, the product of zero variables. A *polynomial* is a sum (modulo 2) of monomials, such as $1 + x_1 + x_2x_3$. An important example is the NOT monomial $\bar{x}_i = 1 - x_i$, which evaluates to 1 precisely when $x_i = 0$.

It is useful to view a polynomial as a function $f : \mathbb{F}_2^m \to \mathbb{F}_2$. For instance, with input coordinates $(a_1, a_2, a_3) = (1, 0, 0)$, the polynomial $f(x_1, x_2, x_3) = x_1x_3$ evaluates to $f(1, 0, 0) = 1 \cdot 0 = 0$. Since for binary variables we have $x_i^2 = x_i$, every polynomial can be uniquely represented as a sum of square-free monomials.

Viewing polynomials as Boolean functions naturally associates to each polynomial a length-$2^m$ binary vector, obtained by evaluating it on all $2^m$ binary inputs in $\mathbb{F}_2^m$. For each input tuple $(a_1, \ldots, a_m)$, we substitute these values into the polynomial $f(x_1, \ldots, x_m)$ and record the output $f(a_1, \ldots, a_m) \in \mathbb{F}_2$. Collecting all outputs in lexicographic order (from $00\cdots0$ to $11\cdots1$), which fixes a canonical ordering of coordinates, gives the *evaluation vector*.

**Definition 5** (Evaluation vector)**.** *Given a polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_m]/\langle x_1^2 - x_1, \cdots, x_m^2 - x_m\rangle$, the evaluation vector is the vector of values $f$ takes on at all the $2^m$ possible coordinates. Note that the values are modulo 2.*

For example, below we present the evaluation vector for all monomials up to $m = 2$

| | | | | |
|---|---|---|---|---|
| $a_2$ | 1 | 1 | 0 | 0 |
| $a_1$ | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| $x_1$ | 1 | 0 | 1 | 0 |
| $x_2$ | 1 | 1 | 0 | 0 |
| $x_1x_2$ | 1 | 0 | 0 | 0 |

We now review some salient properties of polynomials.

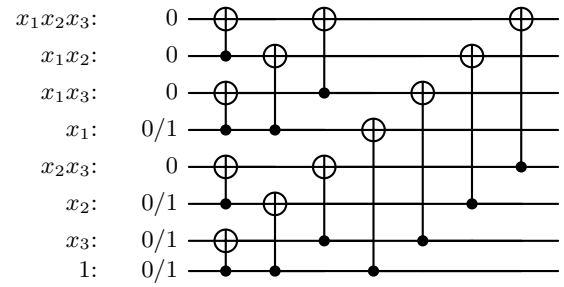



Figure 8. **Encoding circuit of RM(1, 3)**. Bits corresponding to monomials of degree more than 1 are fixed to 0, while the rest are variable. There are 4 variable bits, corresponding to the code dimension $k = 4$.

- *Algebraic conventions.* All arithmetic is over $\mathbb{F}_2$, so the polynomial ring has characteristic two. In particular, for any polynomial $f$ we have $2f = 0$, and hence addition and subtraction coincide. Moreover, since we quotient by the relations $x_i^2 = x_i$, every polynomial in $\mathbb{F}_2[x_1, \ldots, x_m]/\langle x_1^2 - x_1, \ldots, x_m^2 - x_m\rangle$ can be written as a sum of square-free monomials; higher powers satisfy $x_i^a = x_i$ for all $a \geq 1$.
- *Overlap.* Given two polynomials $f, g$, their *overlap* is defined as the bitwise AND of their evaluation vectors. Equivalently, the overlap is the evaluation vector of the product polynomial $fg$. For $m = 2$, let $f(x_1, x_2) = x_1$ and $g(x_1, x_2) = x_2$. Their product is $fg = x_1x_2$, whose evaluation vector is $(0, 0, 0, 1)$ in lexicographic order.
- *Weight.* The *weight* $\mathrm{wt}(f)$ of a polynomial $f$ is the Hamming weight of its evaluation vector. The following properties will be useful:
  - If $f$ is a monomial in $m$ variables that omits $i$ variables, then $\mathrm{wt}(f) = 2^i$.
  - If a polynomial $f$ omits $j$ variables (i.e., no monomial term depends on them), then $\mathrm{wt}(f)$ is divisible by $2^j$.

### *b. Reed–Muller codes*

**Definition 6** (Reed–Muller codes)**.** *For a positive integer $m$ and $r \leq m$, the Reed–Muller code* $\mathrm{RM}(r, m)$ *is the span of evaluation vectors of all monomials in $m$ variables of degree at most $r$.*

The code $\mathrm{RM}(r, m)$ has parameters $[2^m, \sum_{i=0}^{r} \binom{m}{i}, 2^{m-r}]$. The distance arises from the highest-degree (degree-$r$) monomials, whose evaluation vectors have weight $2^{m-r}$, giving $d = 2^{m-r}$. The dual code of $\mathrm{RM}(r, m)$ is $\mathrm{RM}(m - r - 1, m)$. Examples of generating monomials are

$$\begin{aligned} \mathrm{RM}(2, 2): &\quad \{1,\, x_1,\, x_2,\, x_1x_2\}, \\ \mathrm{RM}(1, 3): &\quad \{1,\, x_1,\, x_2,\, x_3\}. \end{aligned}$$

All Reed–Muller (RM) codes share the same structure of encoding circuit as illustrated in Fig. 8 [63]; different RM codes are obtained solely by choosing which input qubits are variable (initialized to either $|0\rangle$ or $|1\rangle$), with all remaining inputs fixed to $|0\rangle$. The variable inputs correspond to the generating monomials of the code. For example, $\mathrm{RM}(1, 3)$ has variable inputs associated with $1, x_1, x_2, x_3$, while $\mathrm{RM}(2, 2)$ uses $1, x_1, x_2, x_1x_2$; in both cases $k = 4$.

### *c. Affine transformations*

The automorphism group of a Reed–Muller code arises from affine transformations of the variables. Let $A \in \mathrm{GL}(m, \mathbb{F}_2)$ and $\mathbf{b} \in \mathbb{F}_2^m$, and define the affine map

$$T(x) = Ax + \mathbf{b}. \tag{E1}$$

Such a transformation induces (i) a permutation of the $2^m$ coordinates via $a \mapsto T(a)$, and (ii) an action on polynomials by substitution, $f(x) \mapsto f(T(x))$. Since affine transformations preserve degree, this substitution maps degree-$\leq r$ polynomials to degree-$\leq r$ polynomials. Moreover, applying the inverse of the induced coordinate permutation has

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $x_1$ | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| $x_2$ | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| $x_3$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

*Original generators*

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $x_1+x_2$ | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| $x_2$ | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| $x_3$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

*Affine map applied to monomials*

| | 6 | 7 | 5 | 4 | 2 | 3 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 110 | 111 | 101 | 100 | 010 | 011 | 001 | 000 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $x_1$ | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| $x_2$ | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| $x_3$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

*Affine map applied by permuting bits*

Figure 9. **Equivalence between affine transformation of variables and permutation of bits.** We illustrate an example of an affine transformation, as described in Eq. (E2), and a bit permutation that produces an identical action on the code.

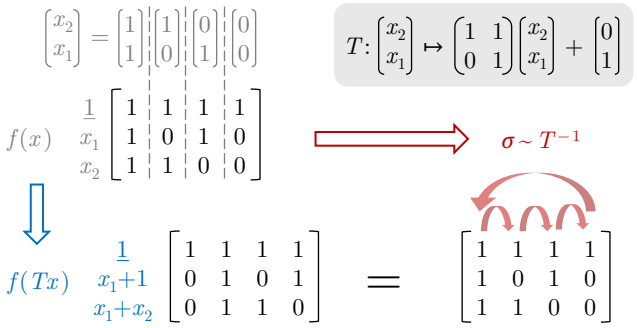



Figure 10. **Automorphism of a Reed–Muller code.** An affine transformation $T$ to the variables induces two equivalent operations on the generator matrix: 1) permuting the coordinates by $T^{-1}$, and 2) substituting $x_i \mapsto T(x_i)$ in each polynomial.

the same effect on evaluation vectors as polynomial substitution. Consequently, affine transformations act as automorphisms of $\mathrm{RM}(r, m)$, and

$$\mathrm{Aut}(\mathrm{RM}(r, m)) = \mathrm{AGL}(m, \mathbb{F}_2).$$

**Lemma 5** (Reed–Muller code automorphism [33]). *For $A \in \mathrm{GL}(m, \mathbb{F}_2)$ and $\mathbf{b} \in \mathbb{F}_2^m$, the coordinate permutation $\sigma \in S_{2^m}$ that maps $A\mathbf{x} + \mathbf{b}$ to $\mathbf{x}$ is a code automorphism, under which codewords transform as $f(\mathbf{x}) \mapsto f(T\mathbf{x})$. This* general affine group *formed by $A$ and $\mathbf{b}$ is the automorphism group of the RM codes in $m$ variables for any $1 \le r \le m-2$.*

The following example illustrates how transforming the monomial is equivalent to permuting bits. Consider

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{E2}$$

As shown in Fig. 9, this transformation is equivalent to swapping bits $(2, 3)$ and $(6, 7)$.

A second example is shown in Fig. 10, where the generator matrix is for $\mathrm{RM}(1, 2)$ and the variables are mapped as $T(x_2) = x_1 + x_2$ and $T(x_1) = x_1 + 1$. Thus, the coordinates (i.e., columns in the generator matrix) are permuted according to $T^{-1}$: $00 \mapsto 11$, $11 \mapsto 10$, $10 \mapsto 01$, and $01 \mapsto 00$. One can verify that this is equivalent to transforming the polynomials (i.e., rows of the generator matrix): $1 \mapsto 1$ (the constant polynomial), $x_1 \mapsto T(x_1) = x_1 + 1$, and $x_2 \mapsto T(x_2) = x_1 + x_2$.

### 2. Quantum Reed–Muller code construction

The stabilizers and logical operators of quantum Reed–Muller (qRM) codes can be described via classical Reed–Muller codes.

**Definition 7** (Quantum Reed–Muller codes [19]). *For positive $m$ and $l \le m$, the quantum code with $X$-type stabilizers given by $\mathrm{RM}(l-1, m)$ and $Z$-type stabilizers given by $\mathrm{RM}(m-l-1, m)$ has parameters $[\![2^m, \binom{m}{l}, \min(2^{m-l}, 2^l)]\!]$.*

Following Definition 7, on qRM codes, the degree-$l$ monomials serve as $X$-type logical operators, and the $Z$-type logical operators can be obtained by taking the corresponding negated complementing monomials: for a monomial $f$ of degree $l$, take $x_1 x_2 \dots x_m / f$ (a degree $m-l$ monomial) and replace each variable $x_i$ with its complement $\overline{x_i} = x_i + 1$.

While the codes in Definition 7 are valid CSS quantum codes (see e.g. Ref. [19] for proof)—i.e. their stabilizers commute, and their logical operators have the correct commutation structure between themselves and commute with stabilizers—they are not guaranteed to be phantom. Yet, as we will show, there is a way to construct phantom codes out of these codes.

**Proposition 10** (Hypercube codes are phantom)**.** *All codes with $l = 1$ from Definition 7 are phantom. These codes have parameters $[\![2^D, D, 2]\!]$ and are also known as hypercube codes.*

*Proof.* The automorphism of classical RM codes directly applies to this quantum code family. Specifically, under an affine transformation $T$ as in Eq. (E1), permuting the coordinates of physical qubits by $T^{-1}$ is equivalent to transforming the logical generator monomials by $T$. The affine component $\mathbf{b}$ acts as a logical identity: when applied to a degree-$l$ monomial (i.e., an $X$-type logical), $\mathbf{b}$ preserves the degree-$l$ part (the logical) and only introduces lower-degree terms (corresponding to stabilizers). Thus, distinct actions on the logical space are captured by $A \in \mathrm{GL}(m, \mathbb{F}_2)$. For example, consider the $[\![8, 3, 2]\!]$ code, and its higher-dimensional generalization, the $[\![2^D, D, 2]\!]$ hypercube code ($m = D$ and $l = 1$), where the $X$-type logical generators are the $D$ degree-1 monomials $x_1, \ldots, x_D$. This hypercube code is phantom, as every possible logical $\overline{\mathrm{CNOT}}$ circuit one-to-one corresponds to an $A$ describing a qubit permutation. □

An immediate way to build higher-distance phantom codes is to concatenate the $[\![2^D, D, (d_{\mathrm{x}} = 2^{D-1}, d_{\mathrm{z}} = 2)]\!]$ hypercube code with some $k = 1$ code. For example, to obtain a code with approximately equal $X$- and $Z$-sector error-correction capability (i.e. distances)[9], one can balance the distance by concatenating with a $[\![2^{D-2}, 1, (d_{\mathrm{x}} = 1, d_{\mathrm{z}} = 2^{D-2})]\!]$ phase-flip repetition code [18]. The resulting phantom code has parameters $[\![2^{2D-2}, D, 2^{D-1}]\!]$ and possesses $2^{2D-2} - 2^D + 1$ $X$-type stabilizers and $2^D - D - 1$ $Z$-type stabilizers. Despite being balanced in distances, these codes nonetheless contain significantly more stabilizers in one basis than the other, which translates to a practical difference in logical error rate performance in the two sectors.

Next, we show that it is possible to produce more balanced phantom codes by direct construction from the $[\![2^m, \binom{m}{l}, \min(2^{m-l}, 2^l)]\!]$ qRM codes. The strategy is to retain a few logicals while carefully promoting the other logicals to either $X$- or $Z$-type stabilizers. We begin by considering the simplest case where all other logicals are fixed into $Z$-type stabilizers.

**Theorem 5** (Phantom quantum Reed–Muller codes)**.** *From the $[\![2^m, \binom{m}{l}, \min(2^{m-l}, 2^l)]\!]$ qRM code, retain the following $m-l+1$ $X$-logical generators: $x_1x_2\ldots x_{l-1}x_l$, $x_1x_2\ldots x_{l-1}x_{l+1}$, ..., $x_1x_2\ldots x_{l-1}x_m$, and fix the other logicals to the $|\overline{0}\rangle$ state—effectively extending the $Z$-type stabilizers with the negated complements of the other $X$-type logicals. The resulting code is phantom with parameters*

$$[\![2^m, m - l + 1, \min(2^{m-l}, 2^l)]\!].$$

*Proof.* By construction, the code has $2^m$ physical qubits and $m-l+1$ logical qubits. The $X$-type logicals and stabilizers set is a subcode of $\mathrm{RM}(l, m)$, so the $X$-distance of the new code cannot decrease—it in fact remains $2^{m-l}$, since the monomial $x_1x_2\ldots x_l$ still has that weight. The $Z$-type logicals and stabilizers remain within $\mathrm{RM}(m-l, m)$, keeping the $Z$-distance at $2^l$. To enable arbitrary $\overline{\mathrm{CNOT}}$s, it suffices to use the $\mathrm{GL}(m-l+1, \mathbb{F}_2)$ subgroup of $\mathrm{GL}(m, \mathbb{F}_2)$, acting invertibly on $\{x_l, x_{l+1}, \ldots, x_m\}$ but fixing $x_1, \ldots, x_{l-1}$. The action of this subgroup on the $X$-type logical monomials corresponds one-to-one to the set of possible permutation $\overline{\mathrm{CNOT}}$ circuits. □

One can build a more balanced version of phantom qRM codes as follows. We still consider the above $\mathrm{GL}(m-l+1, \mathbb{F}_2)$ action which invertibly transforms $\{x_l, x_{l+1}, \ldots, x_m\}$. We can divide the original $X$-type logicals, namely the degree-$l$ monomials into invariant subsets (orbits) under this action (the invariance is up to degree $\leq l-1$ monomials, i.e., stabilizers), then promote each orbit, either completely to $X$-type stabilizers, or (their negated complement) to $Z$-type stabilizers. To see why the resulting code is still phantom, note that one only needs to verify for the $X$-type sector that stabilizers (resp. logicals) transform to stabilizers (resp. logicals) up to stabilizers. Making the same choice on each orbit precisely guarantees this.

Taking the $[\![64, 4, 8]\!]$ phantom qRM as an example, we divide the $\binom{6}{3} = 20$ degree-3 monomials into four orbits:

$$\{x_1x_2x_3, x_1x_2x_4, x_1x_2x_5, x_1x_2x_6\}, \tag{E3}$$

---

[9] Another reason one might be interested in such balanced codes is that a transversal Hadamard together with qubit permutations might be a valid logical gate. However, we suspect that phantom codes with $ZX$-duality for $k > 2$ may not exist. One can show that transversal $H^{\otimes n}$ implementing logical $\overline{H}^k$ up to any logical $\overline{\mathrm{CNOT}}$ for $k > 2$ is not possible using Theorem 3 and Lemma 4, but it is not clear whether transversal $H$ on a subset of qubits and/or composed with permutations that are not necessarily a valid logical operation can overcome this barrier. In fact, we have not found any $k = 3$ phantom code with $d_{\mathrm{x}} = d_{\mathrm{z}}$ and $H_{\mathrm{x}}, H_{\mathrm{z}}$ having the same rank through code enumeration up to $n = 14$ and non-exhaustive SAT search up to $n \sim 21$.

$$\{x_1x_3x_4, x_1x_3x_5, x_1x_3x_6, x_1x_4x_6, x_1x_5x_6\}, \tag{E4}$$
$$\{x_2x_3x_4, x_2x_3x_5, x_2x_3x_6, x_2x_4x_6, x_2x_5x_6\}, \tag{E5}$$
$$\{x_3x_4x_5, x_3x_5x_6, x_3x_4x_6, x_3x_4x_5\}. \tag{E6}$$

**Definition 8** (Balanced $[\![64, 4, 8]\!]$ phantom qRM code used in benchmarking)**.** *Consider the* $[\![64, 4, 8]\!]$ *phantom qRM code with the orbit* (E3) *chosen to be logical representatives, the negated complement of* (E4) *set to* $Z$*-type stabilizers, and* (E5) *and* (E6) *retained as* $X$*-type stabilizers. The resulting code has* 32 $X$*-type and* 28 $Z$*-type stabilizers. This is the code used in our benchmarking study (Sec. V).*

We found in preliminary simulations that the $[\![64, 4, 8]\!]$ code defined in Theorem 5 has a logical $Z$-sector error rate $\sim 50\times$ worse than the $X$-sector, under circuit-level depolarizing noise (see App. I 1). Therefore, we chose to use the more balanced version in Definition 8 for our benchmarking study, whose logical error rates in both sectors are similar at roughly the geometric mean of the less balanced one.

We give a few further remarks on our phantom qRM construction:

1. The specific choice of setting every orbit to $Z$-type stabilizers (except leaving one as the logicals) allows addressable diagonal logical gates such as $\overline{S}_i\overline{S}_j$ and $\overline{\mathrm{CZ}}_{ij}$ via a physical fold-type circuit ($S$ gates on the fold line and CZ on each pair of qubits mirrored across the fold line), as established in the next subsection (App. E 3).

2. The different balancing choices can be seen as different gauge-fixings of the parent $[\![2^m, \binom{m}{l}, \min(2^{m-l}, 2^l)]\!]$ code. Code-switching between them can be transversally performed via Steane EC [19], which can be understood as a kind of logical teleportation. Then one can implement the $\overline{\mathrm{CNOT}}$s using the most balanced version, and resort to a less balanced version for diagonal logical gates. Fault-tolerant state preparation schemes for both versions are needed to enable the transversal code-switching.

3. On the $[\![64, 4, 8]\!]$ example, choosing both the first (E3) and last (E6) orbits are chosen as logicals is effectively coupling a phantom code with its dual (see App. G 3 a for more details). While the resulting code is not phantom, it inherits phantom-like gates in the form of pairwise products of $\overline{\mathrm{CNOT}}$s; this code then supports permutation logical Hadamards, and transversal $S$ operation produces a logical $\overline{\mathrm{CZ}}$ action.

We additionally remark that a punctured variant of the hypercube codes is also phantom.

**Theorem 6** (Punctured hypercube codes are phantom)**.** *Puncturing the coordinate* $0\cdots00$ *from the hypercube codes yield a family of phantom codes with parameters* $[\![2^D - 1, D, (d_{\mathrm{x}} = 2^{D-1} - 1, d_{\mathrm{z}} = 2)]\!]$.

*Proof.* Recall that the original $[\![2^D, D, (d_{\mathrm{x}} = 2^{D-1}, d_{\mathrm{z}} = 2)]\!]$ hypercube code has a single $X$-type stabilizer supported on every qubit, and $Z$-type stabilizers $\mathrm{RM}(m-1, m)$. The punctured hypercube code again only has one $X$-type stabilizer, which is supported on all the $2^D - 1$ qubits. The $D$ $X$-type logical generators remain the same: they are still expressed as the degree-1 monomials $x_1, \ldots,$ and $x_D$ (these monomials evaluate to zero at the punctured coordinate $0\cdots00$). The $X$-type logicals and stabilizers together form the punctured $\mathrm{RM}(1, D)^*$ code, and $\mathrm{GL}(D, \mathbb{F}_2)$ is its automorphism group [33, Ch. 13, Thm. 24]. Therefore, this punctured version is also phantom for the same reason as the original version. □

For completeness, we state the $Z$-type logicals and stabilizers that form the shortened $\overline{\mathrm{RM}}(D-1, D)$ code: these are precisely the codewords from $\mathrm{RM}(D-1, D)$ that vanish at the coordinate $00\cdots0$, with that coordinate subsequently deleted. Here, the paired $Z$-type logical of the $X$-type logical $x_i$ is its complement monomial $x_1 \ldots x_{i-1}x_{i+1} \ldots x_D$ (no negation).

While the original $[\![2^D, D, 2]\!]$ code supports $D^{\text{th}}$ Clifford-hierarchy-level logical magic via transversal $Z^{1/2^{D-1}}$ rotations, the punctured $[\![2^D - 1, D, 2]\!]$ version loses the highest-level magic gate. The $\leq (D-1)$ level magic gates persists (by addressing subcubes), and the logical $\overline{S}_i\overline{S}_j$ gates via folding described in the next subsection also applies to this punctured family.

Indeed, the $[\![7, 3, (d_{\mathrm{x}} = 3, d_{\mathrm{z}} = 2)]\!]$ phantom code in Table I is the punctured version of the $[\![8, 3, 2]\!]$ hypercube code:

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $x_1$ | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| $x_2$ | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| $x_3$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Concatenating this code with the $[\![2, 1, d_{\mathrm{x}} = 1 \;/\; d_{\mathrm{z}} = 2]\!]$ phase-flip repetition code yields a $[\![14, 3, (d_{\mathrm{x}} = 3, d_{\mathrm{z}} = 4)]\!]$ phantom code that has 8 $X$-type and 3 $Z$-type stabilizers, which was found also by our exhaustive code enumeration. By running the gate solver described in App. H 2 c, we find fold-diagonal gates that implement $\overline{S}_i\overline{S}_j$ or $\overline{\mathrm{CZ}}$ on the

Hadamard-dual of this code. Note that there is also a fold-diagonal gate on the original $[\![7,3,(d_{\rm x}=3,d_{\rm z}=2)]\!]$ phantom code, which can be derived by undoing the CNOTs from the phase-flip repetition code encoding, applying $S$ gates on a subcube of the $[\![7,3,2]\!]$, and redoing the encoding CNOTs; these together can be compiled as $\mathrm{CNOT}_{12}S_2\mathrm{CNOT}_{12} = \mathrm{CZ}S_1S_2$.

There are two other $[\![14,3,(d_{\rm x}=3,d_{\rm z}=4)]\!]$ phantom codes found by enumeration. They have 7/5 $X$-type and 4/6 $Z$-type stabilizers respectively, and they too admit fold-diagonal gates.

### 3. Addressable diagonal logical gates via folding

#### a. *Involutions and associated fold-type circuits on qRM phantom codes*

Here we show that the phantom qRM code family of Theorem 5, where all the extra logical operators are promoted to $Z$-type stabilizers, support $\overline{S}_i\overline{S}_j$ and $\overline{\mathrm{CZ}}_{ij}$ gates through fold-type circuits.

We begin with the $m=2l$ code family with parameters $[\![2^{2l}, l+1, 2^l]\!]$; there, degree $<l$ monomials are both $X$- and $Z$-type stabilizers, the $X$-type logical generators are $x_1x_2\dots x_{l-1}x_l$, $x_1x_2\dots x_{l-1}x_{l+1}$, $\dots$, $x_1x_2\dots x_{l-1}x_{2l}$, and the negated complement of the other degree $l$ monomials are $Z$-type stabilizers. We associate a fold-type circuit to an *involution* $\tau$ permuting the $2^{2l}$ coordinates: single-qubit $S$ or its powers are applied to the fixed points, and CZ between every $(i,\tau(i))$ pair where $i\neq\tau(i)$.

**Proposition 11** (Fold-diagonal logical gates on $m=2l$ phantom qRM codes)**.** *For the $m=2l$ phantom qRM code family of Theorem 5: (a) the involution $\tau_{SS}$ that maps $x_i$ to $\overline{x}_{2l+1-i}$ for $i\in[1,2l]$ implements $\overline{SS}$ on $X$-type logicals $x_1x_2\dots x_{l-1}x_l$ and $x_1x_2\dots x_{l-1}x_{l+1}$; (b) the involution $\tau_{\rm CZ}$ that maps $x_i$ to $\overline{x}_{2l+1-i}$ for $i\in[1,2l]\backslash\{l,l+1\}$ and $x_l=\overline{x}_l$, $x_{l+1}=\overline{x}_{l+1}$ implements $\overline{\mathrm{CZ}}$ between $X$-type logicals $x_1x_2\dots x_{l-1}x_l$ and $x_1x_2\dots x_{l-1}x_{l+1}$.*

*Proof.* Since the physical gates are diagonal and commute with $Z$, the $Z$-type stabilizers of the code are left invariant; we need only check that the $X$-type stabilizers are mapped to stabilizers and that $X$-type logicals transform as desired. It is clear that both $\tau$ stated above are indeed involutions, i.e., $\tau^2=\mathbb{I}$.

The image of an $X$-type stabilizer generator $f$ (degree $\leq l-1$ monomials) under both $\tau_{SS}$ and $\tau_{\rm CZ}$ is the product of itself and the $Z$-type $\tau(f)$, which is likewise a degree $\leq l-1$ monomial and is hence a $Z$-type stabilizer. The phase accumulated is $i^{|f\wedge\tau(f)|}=1$: the product of $f$ and $\tau(f)$ is a degree $\leq 2l-2$ monomial and lacks at least two variables from $\{x_1,\dots,x_{2l}\}$, so by the weight property (see App. E 1), $|f\wedge\tau(f)|$ is divisible by 4.

We now consider the logical action of the two involutions:

- For $\tau_{SS}$, the $X$-type logical $x_1x_2\dots x_{l-1}x_l$ is mapped to the product of itself and the $Z$-type $\overline{x}_{2l}\overline{x}_{2l-1}\dots\overline{x}_{l+2}\overline{x}_{l+1}$, precisely its paired $Z$-type logical, with a phase $i$; and $x_1x_2\dots x_{l-1}x_{l+1}$ is mapped to the product of itself and $Z$-type $\overline{x}_{2l}\overline{x}_{2l-1}\dots\overline{x}_{l+2}\overline{x}_l$, again its paired $Z$-type logical and with a phase $i$. The other $X$-type logical generators are left invariant up to a $Z$-type stabilizer $\tau_{SS}(f)$. This is because $\tau_{SS}(f)$ would contain $\overline{x}_s$, where $1\leq s\leq l-1$, so the $Z$-type operator defined by $\tau_{SS}(f)$ cannot be the paired $Z$-type logical operator of any $X$-type logical.

- For $\tau_{\rm CZ}$, $x_1x_2\dots x_{l-1}x_l$ is mapped to the product of itself and the $Z$-type $\overline{x}_{2l}\overline{x}_{2l-1}\dots\overline{x}_{l+2}\overline{x}_l$, precisely the paired $Z$-type logical of the $X$-type logical $x_1x_2\dots x_{l-1}x_{l+1}$, with a trivial phase. The transformation of the other $X$-type logicals can be similarly verified.

□

To give a concrete example, for the $[\![16,3,4]\!]$ phantom qRM code, $\tau_{SS}$ maps each coordinate as [see Eq. (E1)]

$$\begin{pmatrix}x_4\\x_3\\x_2\\x_1\end{pmatrix}\mapsto\begin{pmatrix}0&0&0&1\\0&0&1&0\\0&1&0&0\\1&0&0&0\end{pmatrix}\begin{pmatrix}x_4\\x_3\\x_2\\x_1\end{pmatrix}+\begin{pmatrix}1\\1\\1\\1\end{pmatrix}. \tag{E7}$$

As the qRM codes are phantom, $\overline{S}_i\overline{S}_j$ and $\overline{\mathrm{CZ}}_{ij}$ fold-type gates are realizable between any two logical qubits (logical $\overline{\mathrm{SWAP}}$s implemented by qubit permutations can be performed on top of any $\tau_{SS}$ and $\tau_{\rm CZ}$ solution). To obtain an individually addressable $\overline{S}_i$ gate, a convenient method is to inject an $\overline{S}$ gate using a targeted $\overline{\mathrm{CNOT}}$ as shown in Fig. 12(b), which can be implemented using two transversal $\overline{\mathrm{CNOT}}$s according to Eq. (A9). It is important to insert a round of (Steane) EC between the two transversal $\overline{\mathrm{CNOT}}$s, so that the targeted $\overline{\mathrm{CNOT}}$ gate is distance preserving.

Lastly, we remark that Proposition 11 can be adapted to other types of phantom qRM codes discussed in this appendix, and comment on the use of fold gates in other scenarios:

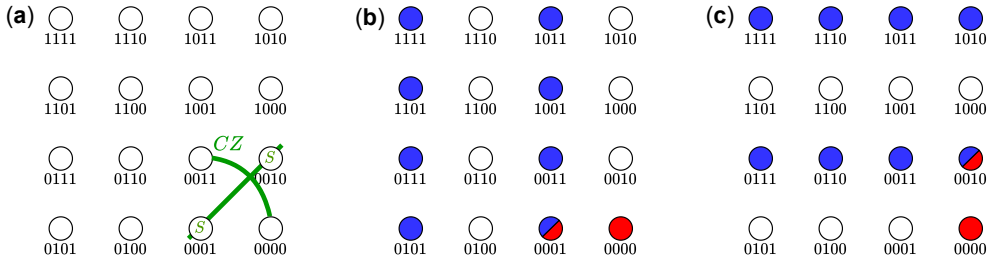



Figure 11. **Example of partial fold-type circuit on phantom qRM codes. (a)** An example partial fold-type circuit for the $[\![16, 4, 2]\!]$ hypercube code, performing an $\overline{SS}$ gate on the two $X$-type logicals $x_1, x_2$. The coordinates are shown in the order of $x_4x_3x_2x_1$, and this circuit only addresses the $\overline{x}_3\overline{x}_4$ subcube, i.e., the coordinates $x_4x_3x_2x_1 = 00{**}$. **(b)** The $X$-type logical $x_1$ is mapped into its paired $Z$-type logical $\overline{x}_2\overline{x}_3\overline{x}_4$. **(c)** The $X$-type logical $x_2$ is mapped into its paired $Z$-type logical $\overline{x}_1\overline{x}_3\overline{x}_4$.

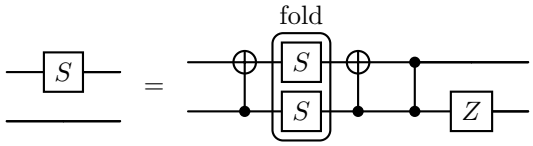
- Note that Proposition 11 applies also to the $l < m/2$ phantom qRM codes of Theorem 5, but the fold-type circuit should only address the $C = \overline{x}_{2l+1}\overline{x}_{2l+2}\cdots\overline{x}_m$ subcube, i.e., the coordinates $x_m \ldots x_1 = \underbrace{00\cdots0}_{m-2l}\underbrace{{**}\cdots*}_{2l}$.

  We refer to this as a *partial* fold-type circuit. An example is shown in Fig. 11 for the $[\![16, 4, 2]\!]$ code, where the circuit only addresses the subcube $\overline{x}_3\overline{x}_4$ and implements $\overline{SS}$ on the $X$-type logicals $x_1$ and $x_2$. More generally, a monomial $f$ denoting an $X$-type logical or stabilizer generator transforms under the partial fold $\tau$ to the product of itself and a $Z$-type $\tau(f \wedge C)$. For example, one can verify that the $Z$-type contribution in the transformed $X$-type logical $x_1x_2\cdots x_{l-1}x_l$ is $\tau_{SS}(x_1x_2\cdots x_{l-1}x_l\overline{x}_{2l+1}\cdots\overline{x}_m) = \overline{x}_{2l}\overline{x}_{2l-1}\cdots\overline{x}_{l+2}\overline{x}_{l+1}\overline{x}_{2l+1}\cdots\overline{x}_m$, which is its paired $Z$-type logical. The action on other $X$-type monomials can be similarly verified and we omit them for brevity.

- We do not examine $l > m/2$, because one can instead consider $l' \leftarrow m - l$ and the resulting $[\![2^m, m - l' + 1, \min(2^{m-l'}, 2^{l'})]\!]$ code has the same distance but a higher $k$.

- On the $[\![2^D - 1, D, (d_x = 2^{D-1} - 1, d_z = 2)]\!]$ punctured hypercube codes, to perform a fold-$\overline{SS}$ gate on $X$-type logicals $x_1, x_2$, one uses $\tau_{SS}$ that maps $x_1$ to $x_2$, and $x_2$ to $x_1$, and restricts to the $x_3\cdots x_m$ subcube. For fold-$\overline{\mathrm{CZ}}$ on $x_1, x_2$, one can simply apply single-qubit $S$ gates to the $x_3\cdots x_m$ subcube.

- To perform an addressable $\overline{S}$ gate, one can use a targeted $\overline{\mathrm{CNOT}}$ to teleport in a $\overline{S}$ gate (see Fig. 12b) from a freshly prepared all-plus state acted on by fold-$\overline{SS}$. This approach discussed further in the following subsection (App. E 3 b). Alternatively, one can decompose the desired $\overline{S}$ through the following circuit identity:




  The $\overline{\mathrm{CZ}}$ can be performed through folding or automorphism (if available). In the case of the former, the logical performance of this decomposition suffers if the fold-$\overline{\mathrm{CZ}}$ is not distance-preserving.

### b. *Distilling $\overline{SS}$ states on qRM and self-dual binarization-and-concatenation phantom codes*

A subtlety is that fold-type gates are not distance-preserving in general, as they involve two-qubit physical interactions. Indeed, for example, we find by SAT solving (using built-in tools from `stim` [64]) that on the $[\![64, 4, 8]\!]$ phantom qRM code, the fold-diagonal logical gates as described in Proposition 11 suffer from reduced distance: $\tau_{SS}$ exhibits $(d_x, d_z) = (7, 4)$, while $\tau_{\mathrm{CZ}}$ exhibits $(d_x, d_z) = (6, 4)$. While the circuit-level distance $d = 4 > 3$ and therefore error-correction capability is still retained, the decrease in distance generally results in poorer logical error rate performance as compared to distance-preserving (e.g. transversal or automorphism) logical gates.

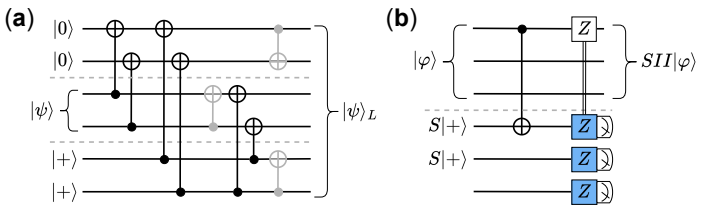



Figure 12. **Distillation scheme for $\overline{SS}$ resource states and teleportation.** **(a)** Encoding circuit of $[\![6,2,2]\!]$, to be implemented at the logical level using three $k=2$ phantom codeblocks, as separated by the dashed grey lines. The grey $\overline{\text{CNOT}}$ gates are permutation gates and can be implemented by qubit relabelling; only the three transversal $\overline{\text{CNOT}}$s in black need to be performed physically. **(b)** Injection of a single $\overline{S}$ gate using a targeted $\overline{\text{CNOT}}$. The ancilla codeblock hosting the $\overline{SS}$ resource state is measured transversally in the $Z$-basis and a conditional $\overline{Z}$ correction performed. Besides the two logical qubits in the $\overline{S}|\overline{+}\rangle$ state, the other logical qubit(s) of the ancilla codeblock can be in an arbitrary state that is not entangled with the first two.

To retain distance, we discuss an alternative scheme employing the following $[\![6,2,2]\!]$ code to distill logical $\overline{SS}$ resource states:

$$H_{\rm x} = H_{\rm z} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}, \quad L_{\rm x} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}. \tag{E8}$$

This code support a logical $\overline{SS}$ gate implemented by transversal $S^\dagger$ operations, and can therefore serve as the outer code for distillation. In particular, we use three (resp. two) codeblocks of $k=2$ (resp. $k>2$) phantom codes to implement the distillation protocol illustrated in Fig. 12. The encoding circuit of the $[\![6,2,2]\!]$ code shown in Fig. 12 uses three interblock $\overline{\text{CNOT}}$s between the phantom codeblocks. When $k=2$ phantom codes are used, the interblock $\overline{\text{CNOT}}$s are straightforwardly implementable transversally; for $k>2$, we can again use transversal $\overline{\text{CNOT}}$ gates rather than addressable interblock $\overline{\text{CNOT}}$s (following Lemma 2), at the expense of ruining the other $k-2$ logicals; this is acceptable since targeted $\overline{\text{CNOT}}$(s) will be used when teleporting in the $\overline{S}$ gate(s). After encoding, we perform fold-$\overline{SS}$ on the phantom codes, which is realizable in the qRM and the self-dual binarization-and-concatenation codes (see App. F), and on various other numerically identified phantom codes (see Table I). Lastly, we perform unencoding of the $[\![6,2,2]\!]$ outer code by running the $\overline{\text{CNOT}}$s in reverse order, followed by transversal measurement of the first and third phantom codeblocks in the $Z$- and $X$-basis, respectively. The distillation is deemed successful when $|\overline{00}\rangle$ and $|\overline{++}\rangle$ are respectively recovered in the first and last codeblocks.

Although the distillation protocol entails transversal $\overline{\text{CNOT}}$s, we expect it to bring in advantage when folding is significantly noisier than the transversal $\overline{\text{CNOT}}$ (e.g. one to two orders of magnitude worse in logical error rate). Such a difference in logical error rates can arise at low physical error rates when the fold-type gates are not distance-preserving.

Lastly, we remark that the $[\![6,2,2]\!]$ code used here is the same as that discussed in Eq. (F3), and is the smallest of the $[\![n,n-4,2]\!]$ for ($n$ even) "$H$-code" family introduced in Ref. [65], for which logical Hadamards ($\overline{H}^{\otimes k}$) can be implemented transversally. One can verify that $\overline{S}^{\otimes k}$ gates are implemented by transversal $S^\dagger$ operations for this family, hence providing a route for higher-rate $\overline{S}$ distillation.

### 4. Full logical Clifford group for qRM phantom codes

The $m>2$ phantom qRM codes lack a transversal logical Hadamard gate. However, with the help of diagonal Clifford gates achieved through folding and targeted injection, an addressable logical Hadamard gate can be implemented. This thereby allows the implementation of the full logical Clifford group on these codes.

Figure 13b illustrates the central idea for a $k=4$ phantom code. The protocol combines the three gadgets in Fig. 13a: Hadamard teleportation (left), substituting $\overline{\text{CZ}}$ with $\overline{S}$, $\overline{S}^\dagger$ (middle), and $\overline{\text{CNOT}}$, and the logical one-bit teleportation protocol (right). Both the teleportation gadgets measure the first codeblock in the $X$-basis, and combining them in Fig. 13b allows one to measure all logicals in the $X$-basis through a transversal $X$-basis measurement. The $\overline{S}$, $\overline{S}^\dagger$ gates can be performed by (distillation followed by) targeted injection as explained in the previous subsection. The mixed-basis state $|\overline{+000}\rangle$ can be prepared using a targeted $\overline{\text{CNOT}}$ gate as shown in Fig. 13c. The states of the other three logical qubits do not affect the protocol, so one could prepare the second codeblock in $|\overline{++++}\rangle$ and upon teleportation, measure and preselect on the other qubits being in $|\overline{+}\rangle$ to further improve the chance of having the first logical qubit in $|\overline{+}\rangle$ (and thereby a successful $\overline{H}$ teleported).

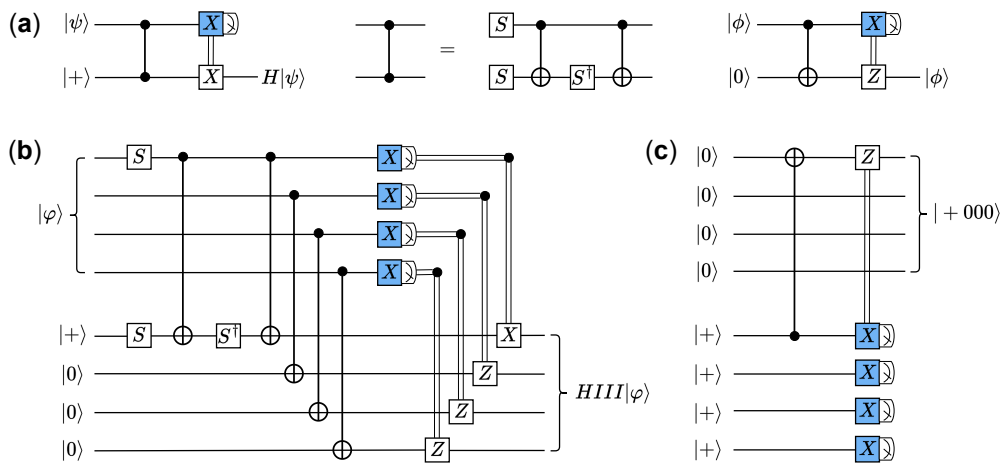



Figure 13. **Addressable logical Hadamard via teleportation and mixed-basis state preparation.** **(a)** Gadget components for **(b)** teleporting a single logical Hadamard gate, where the first codeblock is measured transversally in the $X$-basis and corresponding logical Pauli corrections are performed. The $\overline{S}$ and $\overline{S}^\dagger$ gates in (b) can be injected using a targeted $\overline{\text{CNOT}}$ gate as Fig. 12b and the $|\overline{+000}\rangle$ mixed-basis state preparation following **(c)**. The scheme in (c) is used in our logical GHZ state preparation benchmarking (see Fig. 5b).

To perform an $\overline{H}^{\otimes 2}\overline{\mathbb{I}}^{\otimes 2}$ gate instead, we add $\overline{S}$ gates on the second logical qubit on both codeblocks, and another $\overline{\text{CNOT}}$ between them. The $\overline{\text{CNOT}}$ can again be achieved physically through two transversal $\overline{\text{CNOT}}$ gates (see Lemma 2). Similar to the first logical qubit, the conditional correction on the second logical qubit changes to $X$-type.

On our phantom codes, this scheme of performing an addressable logical Hadamard has a lower overhead compared to the conventional scheme of utilizing the identity $(HS)^3 \propto \mathbb{I}$—concretely, for example, at $k = 2$ and with the ability to perform addressable $\overline{S}$, the sequence $(H\otimes H)(S\otimes\mathbb{I})(H\otimes H)(S\otimes\mathbb{I})(H\otimes H)(S\otimes\mathbb{I}) \propto \mathbb{I}\otimes H$ produces an addressable Hadamard.

### 5. Magic gates on qRM phantom codes

The infinite family of quantum Reed–Muller phantom codes we identified allows a distance-limited mechanism for magic gates. We start from the observation that the $[\![16, 3, 4]\!]$ phantom code can be decoupled into two $[\![8, 3, 2]\!]$ codes, as illustrated in Fig. 14, by undoing the encoding CNOTs connecting the $x_1 = 0$ and $x_1 = 1$ subcubes. One can then perform a $\overline{\text{CCZ}}$ gate using transversal physical $T$ gates on the target $[\![8, 3, 2]\!]$ code, and then couple them back to restore the $[\![16, 3, 4]\!]$ code. Alternatively, one can compile this CNOT–$T$–CNOT sequence as $\text{CS}_{12}^\dagger T_1 T_2$ on the physical qubit pairs acted on.

More generally, our phantom qRM codes of parameters $[\![2^m, m-l+1, \min(2^{m-l}, 2^l)]\!]$ (balanced or not) can be decoupled into the hypercube codes $[\![2^{m-l+1}, m-l+1, 2]\!]$ by undoing the last $l-1$ layers of their hypercube encoding circuits and thereby only keeping the $x_1 = x_2 = \cdots = x_{l-1} = 1$ subcube. There, logical $\text{C}^{l_0}\text{Z}$ gates are accessible for all $l_0 \le m - l$ through physical $Z$-basis rotations on the qubits of suitable subcubes [66]. For example, the $[\![64, 4, 8]\!]$ code reduces to a $[\![16, 4, 2]\!]$ block, where a transversal $\sqrt{T}/\sqrt{T}^\dagger$ implements a $\overline{\text{CCCZ}}$ gate, and transversal $T/T^\dagger$ on an 8-qubit subcube gives a $\overline{\text{CCZ}}$ [66]. The limitation is that the intermediate codes have a distance of only two.

Lastly, we comment that on the parent $[\![64, 15, 4]\!]$ qRM code (where the $X$-type logicals are the $\binom{6}{2}$ degree-2 monomials) of the $[\![64, 5, 4]\!]$ phantom code, transversal $T$ gates lead to a logical hypergraph $\overline{\text{CCZ}}$ circuit. The $\overline{\text{CCZ}}$s are on every triplet of degree-2 monomials that multiply to $x_1x_2x_3x_4x_5x_6$ [66, 67]. For example, $(x_1x_2, x_3x_4, x_5x_6)$, $(x_1x_2, x_3x_5, x_4x_6)$, etc.; each triple intersection has support at one coordinate $x_6 = \cdots = x_1 = 1$. The phase polynomial framework in App. H 2 b allows a simpler proof of this result than in Refs. [66, 67]. However, the $[\![64, 5, 4]\!]$ phantom qRM code, where $x_1x_2, x_1x_3, x_1x_4, x_1x_5, x_1x_6$ serve as the $X$-type logicals, does not have such a triplet. One could consider swapping the monomials in the parent $[\![64, 15, 4]\!]$ qRM code (e.g. swap $x_1x_3$ with $x_3x_5$, and $x_1x_4$ with $x_4x_6$) through interleaving qubit permutations with transversal $\overline{\text{CNOT}}$s with an ancilla codeblock [19], applying transversal $T$ gates, and then swapping back. The disadvantage of this scheme, however, is the high depth of the circuit. Therefore, though in principle we can realize $\overline{\text{CCZ}}$s of fault distance four this way, it might be cheaper in practice to distill $\overline{\text{CCZ}}$ states through the distance-two decoupling scheme above and inject them where needed.

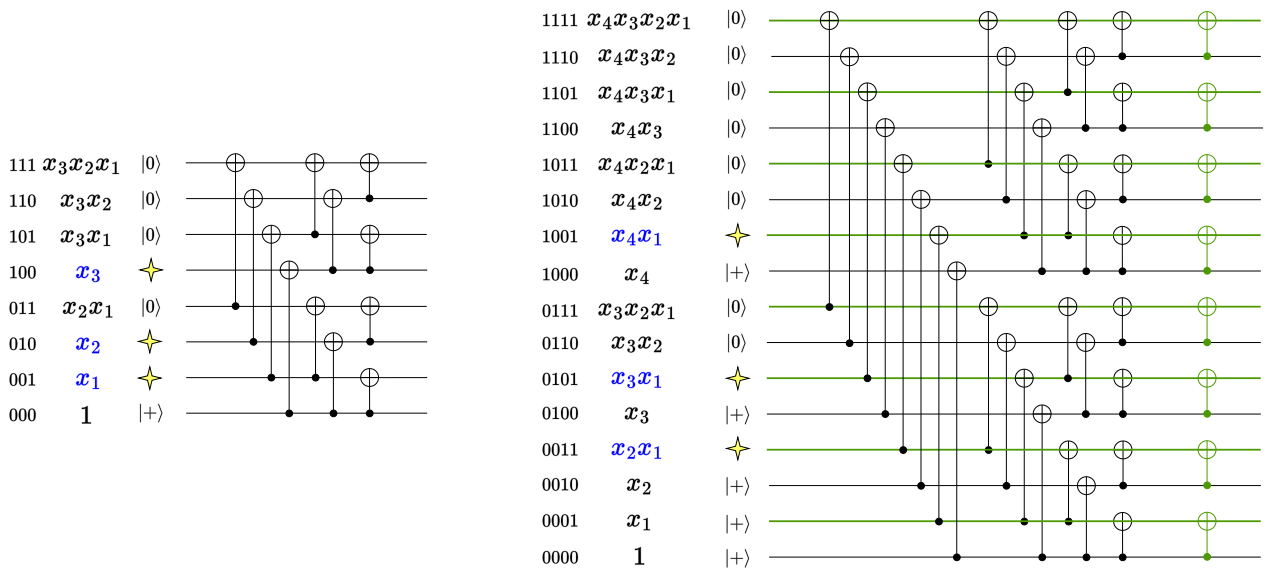



Figure 14. **Encoding circuits and decoupling scheme for phantom qRM codes**. The hypercube encoding circuit of the $[\![8,3,2]\!]$ (left) and $[\![16,3,4]\!]$ (right) phantom qRM codes. This recursively constructed encoding circuit applies in fact to the entire qRM code family [19]. On the input side of the circuit, $x_3x_2x_1 = 111, \ldots, 000$ etc., label the coordinate of each qubit; each monomial indicates what an $X$ operator placed on this qubit propagates to through this encoding circuit. This can be thought of as stabilizer propagation: if a qubit is initialized to $|+\rangle$, which is stabilized by a single-qubit $X$, the propagation result of the $X$ is an $X$-type stabilizer of the code. Next, we take the $[\![16,3,4]\!]$ (right) as an example to illustrate decoupling. The hypercube circuit couples qubits whose coordinates differ at $x_4$ in the first round using transversal CNOTs, couples those that differ at $x_3$ in the next round, and so on. By undoing the last round of transversal CNOTs (green) and focusing on the coordinates where $x_1 = 1$, the $[\![16,3,4]\!]$ is decoupled to the $[\![8,3,2]\!]$ code. For the $[\![2^m, m-l+1, \min(2^{m-l}, 2^l)]\!]$ phantom code, we undo the $x_1, \ldots, x_{l-1}$ layers of CNOTs, and keep those coordinates where $x_1 = 1, \ldots, x_{l-1} = 1$; the resulting code is the $[\![2^{m-l+1}, m-l+1, 2]\!]$ hypercube code.

### 6. Space-time efficient analog rotation

Space-time efficient analog rotation (STAR) has been shown to be useful for implementing arbitrary small-angle single-qubit rotations in early fault-tolerant quantum computers [49, 68]. In this section, we provide an implementation for the $[\![64,4,8]\!]$ qRM phantom code and discuss some restrictions of STAR on related codes.

#### *a. STAR for the $[\![64,4,8]\!]$ qRM phantom code*

Figure 15 illustrates the STAR protocol for a small-angle $\overline{R}_Z$ gate. We discuss the rotation as applied to the first logical qubit of the code; rotations on other logical qubits work similarly (alternatively, as the code is phantom, logical swaps can be performed through qubit permutations). Concretely, physical $R_Z(\theta)$ rotations are applied on the support of the weight-8 logical operator $\overline{Z}_1$, $R_Z^{\otimes 8}(\theta) = [\cos(\theta/2)\mathbb{I} - i\sin(\theta/2)Z]^{\otimes 8}$, which can be expanded as a sum of tensor-product terms with different Pauli weights. At this point, a round of error detection is performed. Note that since the code distance is 8, $Z$-type Pauli operators with weight between 1 and 7 correspond to detectable errors. After preselecting for no detected error, the induced logical operation is proportional to $\cos^8(\theta/2)\overline{\mathbb{I}} + \sin^8(\theta/2)\overline{Z}_1$. Since the logical qubit is initialized in the $|\overline{+}\rangle$ state, this effective logical action is equivalent to $\overline{R}_Y(-\gamma)$ for an angle $\gamma$ given by $\tan(\gamma/2) = \tan^8(\theta/2)$.

Following this resource state preparation, a transversal $\overline{\text{CNOT}}$ from the target (data) codeblock to the ancillary codeblock is applied, followed by a fold-$\overline{S}_1\overline{S}_2$ operation (see App. E 3), and finally the ancillary codeblock is measured in the $X$-basis. Note that the $\overline{\text{CNOT}}$s only act non-trivially on the first logical qubit in the ancillary codeblock since the other qubits are in $|+\rangle$ states.

The net effect of the circuit on the target codeblock is a $\overline{R}_Z(\pm\gamma)$ on the first logical qubit conditioned on the measurement outcome $m$ of the first logical qubit in the ancillary codeblock: with probability $1/2$ we obtain $m = 1$ and realize the desired $Z$-rotation, while with probability $1/2$ we obtain $m = -1$, in which case we must compensate

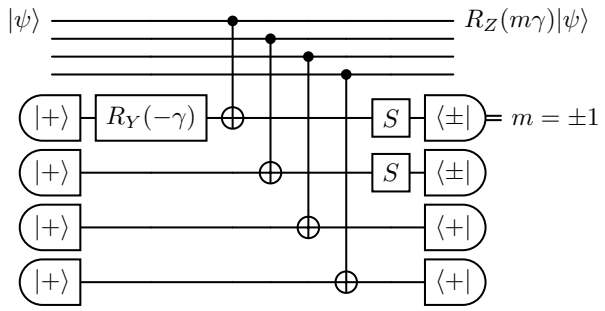



Figure 15. **Teleportation of STAR-prepared small-angle rotation resource state.** The circuit illustrated here is for a $k = 4$ phantom code (e.g. the $[\![64, 4, 8]\!]$ code). The target codeblock, on which we wish to implement a logical rotation, is depicted on top, and an ancillary codeblock is introduced below. The ancillary block is initialized to the all-$|\overline{+}\rangle$ state, after which the STAR scheme is used to implement the $R_Y$ logical gate.

by applying a $2\gamma$ rotation. This compensation procedure again succeeds with probability $1/2$, and otherwise requires a further "fix-up" rotation with twice the angle. From this probabilistic structure, it follows that, on average, we need to perform the STAR circuit twice to implement a single logical $Z$-rotation gate.

*b. Limitations of STAR for CSS codes with even-weight stabilizer generators*

The reason for the $\overline{S}$ gates in Fig. 15 is that the STAR implements an $\overline{R}_Y$ on the ancillary block, while we intend to realize $\overline{R}_Z$ on the target block. This is in fact a limitation due to the code structure. We begin with the following fact:

**Lemma 6.** *For a CSS code such that any pure-$X$ logical operator or stabilizer is of even weight and any pure-$Z$ logical operator or stabilizer is of even weight, any such logical operator is of even weight: a single $Z$-logical and any $X$-logical on the other logical qubits; any such logical operator are of odd weight: a single $Y$-logical and any $X$-logical on the other logical qubits.*

*Proof.* Without loss of generality, consider the operator $\overline{Z}_1 \cdot \overline{X}_{\mathcal{L}}$ where $\overline{X}_{\mathcal{L}}$ is any product of $\overline{X}$ operators excluding $\overline{X}_1$. Such a logical operator corresponds to physical Pauli operators of the form $\mathcal{Z} \cdot \mathcal{X}$. Since $\mathcal{X}$ is pure-$X$ and $\mathcal{Z}$ is pure-$Z$, and they commute (because $\overline{Z}_1$ and $\overline{X}_{\mathcal{L}}$ commute), their support overlap on an even number locations, which turns to an even number of physical $Y$s and a $\pm$ sign. Thus, $\overline{Z}_1 \cdot \overline{X}_{\mathcal{L}}$ is the remaining (even number of) $X$s in $\mathcal{X}$, the remaining (even number of) $Z$s in $\mathcal{Z}$ and an even number of $Y$s, which sums up to an even weight.

Without loss of generality, consider operator $\overline{Y}_1 \cdot \overline{X}_{\mathcal{L}} = i\overline{X}_{\mathcal{L}\cup\{1\}}\overline{Z}_1$ where, again, $\overline{X}_{\mathcal{L}}$ is any product of $\overline{X}$ operators excluding $\overline{X}_1$. Such a logical operator corresponds to physical Pauli operators of the form $i\mathcal{X}\cdot\mathcal{Z}$. Since $\mathcal{X}$ is pure-$X$, $\mathcal{Z}$ is pure-$Z$, and they anticommute (because $\overline{X}_{\mathcal{L}\cup\{1\}}$ and $\overline{Z}_1$ anticommute), their support overlap on an odd number of locations, which turns to an odd number of physical $Y$s and an odd number of $i$s. Adding also the $i$ in the expression, the total sign is a $\pm$. Thus, $\overline{Y}_1 \cdot \overline{x}_{\mathcal{L}}$ is the remaining (odd number of) $X$s in $\mathcal{X}$, the remaining (odd number of) $Z$s in $\mathcal{Z}$, and an odd number of $Y$s, which sums up to an odd weight. □

This restricts the accessible rotation bases of the STAR scheme, as we formalize below.

**Proposition 12.** *For a CSS code such that any pure-$X$ logical operator or stabilizer is of even weight and any pure-$Z$ logical operator or stabilizer is of even weight, the logical action of such a STAR protocol cannot be $\overline{R}_Z$: 1) prepare logical all-$|+\rangle$ state, 2) apply general single-qubit physical gates, 3) perform error detection.*

*Proof.* Per Lemma 6, since any physical Pauli operator corresponding to $\overline{Z}_1\overline{X}_{\mathcal{L}}$ with $1 \notin \mathcal{L}$, has an even weight, the coefficient for such a term in $\prod_q e^{i\theta\vec{\sigma}_q\cdot\vec{n}_q}$ is real, which rules out actions like $(\cos\alpha\overline{\mathbb{I}} + i\sin\alpha\overline{Z}_1\overline{X}_{\mathcal{L}})\,|\overline{+}\rangle^{\otimes k} = (\cos\alpha\overline{\mathbb{I}} + i\sin\alpha\overline{Z}_1)\,|\overline{+}\rangle^{\otimes k}$. Moreover, since any $\overline{Y}_1\overline{X}_{\mathcal{L}}$ with $1 \notin \mathcal{L}$ has an odd weight, the coefficient for such a term is purely imaginary, which rules out actions like $(\cos\alpha\overline{\mathbb{I}} + \sin\alpha\overline{Y}_1\overline{X}_{\mathcal{L}})\,|\overline{+}\rangle^{\otimes k} = (\cos\alpha\overline{\mathbb{I}} - i\sin\alpha\overline{Z}_1)\,|\overline{+}\rangle^{\otimes k}$ (because $\overline{Y}_1\overline{X}_{\mathcal{L}}\,|\overline{+}\rangle^{\otimes k} = -i\overline{Z}_1\,|\overline{+}\rangle^{\otimes k}$). □

## Appendix F: Binarization and concatenation scheme for CSS phantom codes

Here, we construct a class of $k = 2$ CSS phantom codes with high distance. The overarching recipe is to take a qudit CSS code over GF(4) encoding one logical qudit, and encode every GF(4) qudit—implementable using two qubits via binarization—through the qubit $[\![4, 2, 2]\!]_2$ code (i.e. a code concatenation procedure).

We refer to this scheme as binarization-and-concatenation (B&C). We show that, if the qudit code is of parameters $[\![n, 1, d]\!]_4$, then the resulting qubit code from this B&C construction has parameters $[\![4n, 2, \geq 2d]\!]_2$. We give a sufficient condition that the GF(4) code should satisfy in order for the resulting qubit code to be phantom. We then use a family of small GF(4) cyclic codes that naturally satisfy this condition to concretely construct a family of phantom qubit codes; these codes are (Hermitian) self-dual and have extremal distances at small block lengths $n$.

We begin with an introduction to GF(4) codes and the binarization step in App. F 1, and then explain the motivation for concatenating with the $[\![4, 2, 2]\!]_2$ code in App. F 2, thereby deriving the condition on the GF(4) code to guarantee that the resulting qubit code is phantom. Next, we introduce a family of small GF(4) cyclic codes in App. F 3 and show that they satisfy this condition. Finally, in App. F 3 b, we explore the logical gates of the resulting qubit codes.

### 1. GF(4) CSS quantum codes and binarization

Recall that $\mathrm{GF}(4) \cong \mathbb{F}_2[z]/(z^2 + z + 1) \cong \{0, 1, \omega, \omega^2\}$ where $\omega^2 = \omega + 1$. We establish the following basics:

- The conjugate $\overline{x}$ of an element $x \in \mathrm{GF}(4)$ is its square: $\overline{x} = x^2$.
- The trace of an element $x \in \mathrm{GF}(4)$ is $\mathrm{tr}(x) := x + x^2 \in \{0, 1\}$. The trace is an $\mathbb{F}_2$-linear function, meaning for $a, b \in \mathbb{F}_2$, $x, y \in \mathrm{GF}(4)$, $\mathrm{tr}(ax + by) = a\,\mathrm{tr}(x) + b\,\mathrm{tr}(y)$.

We work with GF(4) CSS quantum codes [2, Ch. 8.3.1], whose $X$- and $Z$-type stabilizer groups are themselves GF(4)-linear classical codes. By GF(4)-linear, we not only require stabilizers to be closed under multiplication, but also demand that if $\bigotimes_{j=1}^n Z^{\gamma_j}$ is a $Z$-type stabilizer, $\gamma_j \in \mathrm{GF}(4)$, then for any $\gamma_0 \in \mathrm{GF}(4)$, $\bigotimes_{j=1}^n Z^{\gamma_0\gamma_j}$ is also a $Z$-type stabilizer; and likewise for $X$-type stabilizers. That a $Z$-type stabilizer $\otimes_{j=1}^n Z^{\gamma_j}$ and a $X$-type stabilizer $\otimes_{j=1}^n X^{\eta_j}$ commute means that $\sum_{j=1}^n \gamma_j\eta_j = 0$, $\gamma_j, \eta_j \in \mathrm{GF}(4)$.

A GF(4)-qudit can be implemented using two qubits through binarization [2, Ch. 8.1.2]. This process can be performed as follows. Taking the self-dual normal basis $\{\omega, \omega^2\}$, where $\mathrm{tr}(\omega\omega) = \mathrm{tr}\big(\omega^2\omega^2\big) = 1$ and $\mathrm{tr}\big(\omega\,\omega^2\big) = 0$, we note that any $\alpha \in \mathrm{GF}(4)$ can be written as $\alpha = \alpha_1\omega + \alpha_2\omega^2$ where $\alpha_1 = \mathrm{tr}(\alpha\omega), \alpha_2 = \mathrm{tr}\big(\alpha\omega^2\big) \in \mathbb{F}_2$. In this way, expressing $\alpha$ as $(\alpha_1, \alpha_2)$ therefore provides a bijective map between GF(4) and two $\mathbb{F}_2$ spaces. Explicitly:

- $X^\alpha$ (resp. $Z^\alpha$) where $\alpha \in \mathrm{GF}(4)$ is mapped to $X^{\alpha_1}X^{\alpha_2}$ (resp. $Z^{\alpha_1}Z^{\alpha_2}$).
- An $X$-type stabilizer $\otimes_{j=1}^n X^{\eta_j}$ where $\eta_j \in \mathrm{GF}(4)$ is mapped to $X^{\eta_{1,1}}X^{\eta_{1,2}} \cdots X^{\eta_{n,1}}X^{\eta_{n,2}}$, by expanding each $\eta_j = \eta_{j,1}\omega + \eta_{j,2}\omega^2$, $\eta_{j,1}, \eta_{j,2} \in \mathbb{F}_2$. Likewise for $Z$-type stabilizers.

Indeed, the commutation relationship of $X$- and $Z$-type stabilizers on the GF(4) code translates to the qubit level,

$$\sum_{j=1}^n \gamma_j\eta_j = 0 \implies \sum_{j=1}^n [\gamma_{j,1}\eta_{j,1} + \gamma_{j,2}\eta_{j,2}] = 0, \tag{F1}$$

and therefore the resulting code after binarization is a valid CSS qubit code. This result can be seen by expanding the left-hand side $\sum_{j=1}^n (\gamma_{j,1}\omega + \gamma_{j,2}\omega^2)(\eta_{j,1}\omega + \eta_{j,2}\omega^2) = 0$, taking the trace on both sides, and invoking the $\mathbb{F}_2$-linearity of the trace.

*Examples.* We illustrate an example on the following $[\![3, 1, 2]\!]_4$ code, defined by stabilizer generator matrices

$$H_{\mathrm{x}} = H_{\mathrm{z}} = \begin{pmatrix} 1 & \omega & \omega^2 \end{pmatrix}. \tag{F2}$$

We remind that, since the code is GF(4)-linear, each stabilizer multiplied by $\omega$ or $\omega^2$ is also a stabilizer. For each row, we binarize each GF(4) entry as described above. For example, $\begin{pmatrix} \omega & \omega^2 & 1 \end{pmatrix}$ becomes $\begin{pmatrix} 10 & 01 & 11 \end{pmatrix}$, and $\begin{pmatrix} \omega^2 & 1 & \omega \end{pmatrix}$ becomes $\begin{pmatrix} 01 & 11 & 10 \end{pmatrix}$. The resulting qubit code after binarization has parameters $[\![6, 2, 2]\!]_2$ and are defined by stabilizer generator matrices

$$H_{\mathrm{x}}^{\mathrm{bin}} = H_{\mathrm{z}}^{\mathrm{bin}} = \begin{pmatrix} 10 & 01 & 11 \\ 01 & 11 & 10 \end{pmatrix}, \tag{F3}$$

where we retained linearly independent rows. Likewise, we can binarize the following $[\![5,1,3]\!]_4$ code,

$$H_{\rm x} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & \omega & \omega^2 & 1 \end{pmatrix}, \qquad H_{\rm z} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & \omega^2 & \omega & 1 \end{pmatrix}. \tag{F4}$$

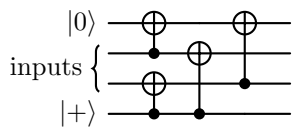
to obtain a $[\![10,2,3]\!]_2$ code,

$$H_{\rm x}^{\rm bin} = \begin{pmatrix} 10 & 10 & 10 & 10 & 00 \\ 01 & 01 & 01 & 01 & 00 \\ 00 & 10 & 01 & 11 & 10 \\ 00 & 01 & 11 & 10 & 01 \end{pmatrix}, \qquad H_{\rm z}^{\rm bin} = \begin{pmatrix} 10 & 10 & 10 & 10 & 00 \\ 01 & 01 & 01 & 01 & 00 \\ 00 & 10 & 11 & 01 & 10 \\ 00 & 01 & 10 & 11 & 01 \end{pmatrix}. \tag{F5}$$

These two examples will be reused throughout the following subsections.

### 2. Concatenation with the $[\![4,2,2]\!]$ code

The binarized $[\![6,2,2]\!]_2$ and $[\![10,2,3]\!]_2$ codes are not phantom (the phantomness of a given code can be concretely checked by a SAT formulation, see App. C). To obtain phantom codes, we concatenate the binarized code with the $[\![4,2,2]\!]_2$ code, by passing every qubit pair encoding one GF(4)-qudit through the $[\![4,2,2]\!]_2$ encoding circuit.




i.e., $[\![4,2,2]\!]_2$ is used at the inner level to encode the two qubits representing a GF(4)-qudit at the outer level. The binarization and concatenation taken as a whole perform the following mapping from a GF(4)-qudit to four qubits: $X^\omega \mapsto XXII$ and $Z^{\omega^2} \mapsto IIZZ$; $X^{\omega^2} \mapsto XIXI$ and $Z^\omega \mapsto IZIZ$; and $X^1 \mapsto IXXI$ and $Z^1 \mapsto IZZI$, because $1 = \omega^2 + \omega$. As a standard consequence of code concatenation, the resulting qubit code also contains sets of weight-four $X$- and $Z$-type stabilizer generators arising from the stabilizers of the $[\![4,2,2]\!]_2$ codeblocks used for each GF(4) qudit.

**Proposition 13** (Distance of binarization-and-concatenation codes)**.** *A* GF(4) *CSS quantum code of distance $d$ produces a CSS qubit code of distance at least $2d$ after binarization and concatenation with the $[\![4,2,2]\!]_2$ code.*

*Proof.* As seen above, the weight of any single nontrivial Pauli operator on a GF(4)-qudit is doubled to weight-two on two qubits through the binarization and concatenation procedure. Therefore, if all logical operators on the GF(4)-qudit code are of weight $\geq d$, then all logical operators on the resulting qubit code are of weight $\geq 2d$. □

The $[\![6,2,2]\!]_2$ and $[\![10,2,3]\!]_2$ codes yield $[\![12,2,4]\!]_2$ and $[\![20,2,6]\!]_2$ codes after concatenation, which are phantom, and are in fact independently discovered by our SAT-based code discovery effort (see App. D). We delay the rigorous proof of phantomness for the concatenated codes after introducing the GF(4) quadratic residue code family in App. F 3, in which the $[\![3,1,2]\!]_4$ and $[\![5,1,3]\!]_4$ serve as the smallest examples.

For now, we give the intuition behind why the binarization-and-concatenation procedure works in creating phantom codes. A central fact we utilize is that the following two operations on a GF(4)-qudit $\gamma$:

1. $\gamma \mapsto \alpha\gamma$ where $\alpha \in \{1, \omega, \omega^2\}$, and
2. the Frobenius transform $\gamma \mapsto \gamma^2$,

together generate the invertible transforms $\mathrm{GL}(2,\mathbb{F}_2)$ on the two qubits implementing $\gamma$. On the logical space of the $[\![4,2,2]\!]_2$ phantom code, $\mathrm{GL}(2,\mathbb{F}_2)$ is the space of $\overline{\mathrm{CNOT}}$ logical circuits and are implementable by physical qubit permutations. To be explicit, consider the transformation $\gamma \mapsto \omega^2\gamma$ as an example. In the self-dual normal basis $\{\omega, \omega^2\}$, this is expanded as $\gamma_1\omega + \gamma_2\omega^2 \mapsto \gamma_1\omega^3 + \gamma_2\omega^4 = (\gamma_1+\gamma_2)\omega + \gamma_1\omega^2$ where $\gamma_1, \gamma_2 \in \mathbb{F}_2$; therefore binarizing the GF(4)-qudit into two qubits, the transformation corresponds to $(\gamma_1, \gamma_2) \mapsto (\gamma_1+\gamma_2, \gamma_1)$, which is precisely the action of a $\overline{\mathrm{CNOT}}$ when the two qubits are encoded in the logical space of the $[\![4,2,2]\!]_2$ code. Likewise, one finds that $\gamma \mapsto \omega\gamma$ corresponds to $(\gamma_1, \gamma_2) \mapsto (\gamma_2, \gamma_1+\gamma_2)$, and the Frobenius transform $\gamma_1\omega + \gamma_2\omega^2 \mapsto \gamma_1\omega^2 + \gamma_2\omega$ corresponds to a swap of qubits, $(\gamma_1, \gamma_2) \mapsto (\gamma_2, \gamma_1)$.

Let us consider an arbitrary $[\![n,1]\!]_4$ code. In general, we can write the computational-basis logical states of a $k=1$ GF(4) CSS code as $|\gamma\rangle_{\rm L} = \sum_{s_{\rm x}\in{\rm rs}(H_{\rm x})} |\gamma l_{\rm x} + s_{\rm x}\rangle$, where $\gamma \in \mathrm{GF}(4)$ and $\mathbf{l}_{\rm x}$ is the logical $X$ operator of the code represented as a GF(4) vector. For the resulting qubit code after binarization and concatenation to be phantom, it must be that $|\gamma\rangle_{\rm L} = |\gamma_1\omega + \gamma_2\omega^2\rangle_{\rm L}$ on the qubit code can be transformed into $|\gamma_1'\omega + \gamma_2'\omega^2\rangle_{\rm L}$ using qubit permutations, where $(\gamma_1 \ \ \gamma_2)^{\mathsf{T}} \mapsto A(\gamma_1' \ \ \gamma_2')^{\mathsf{T}}$ for any invertible $A \in \mathrm{GL}(2,\mathbb{F}_2)$. This means that the qubit code can implement a complete set of individually addressable $\overline{\mathrm{CNOT}}$ gates via qubit permutations.

One sees that any $[\![n,1]\!]_4$ code, after going through the B&C process with $[\![4,2,2]\!]$, can do the $|\gamma\rangle_L \mapsto |\alpha\gamma\rangle_L$ using just permutations: map each qudit $\beta$ to $\alpha\beta$ by permuting the $[\![4,2,2]\!]$ code it is encoded in. By GF(4)-linearity, the

stabilizers are preserved. The logical $X$ operator $\gamma\, l_{\rm x}$ is mapped to $\alpha\gamma\, l_{\rm x}$. Recall that $\gamma \mapsto \alpha\gamma$ where $\alpha \in \{1, \omega, \omega^2\}$, and $\gamma \mapsto \gamma^2$ (Frobenius transform) together generate the invertible transform $\mathrm{GL}(2, \mathbb{F}_2)$ on the two qubits implementing $\gamma$. Therefore, if we can find a way to implement $|\gamma\rangle_L \to |\gamma^2\rangle_L$ using permutation on the B&C code, then the resulting B&C code is phantom. For this purpose, we give the following sufficient condition on the $[\![n, 1]\!]_4$ code.

**Proposition 14.** *Call $H_{\rm x}$ (resp. $H_{\rm z}$) the $X$ (resp. $Z$) stabilizer of the* GF(4)*-linear CSS code $[\![n, 1]\!]_4$, and $l_{\rm x} \in \mathbb{F}_4^n$ an $X$ logical representative. Denote the coordinate-wise conjugation of the two matrices and vector $l_{\rm x}$ by $\overline{H_{\rm x}}$, $\overline{H_{\rm z}}$ and $\overline{l_{\rm x}}$. If there exists a permutation $\pi$ of the $n$ coordinates such that the stabilizers are preserved, i.e., $rs(\pi(\overline{H_{\rm x}})) = rs(H_{\rm x})$ and $rs(\pi(\overline{H_{\rm z}})) = rs(H_{\rm z})$, and $\pi(\overline{\gamma l_{\rm x}})$ equals $\gamma^2 l_{\rm x}$ up to $X$ stabilizers, then the resulting B&C code is phantom.*

*Proof.* We first do a coordinate-wise conjugation $c_i \mapsto c_i^2$, $i \in [n]$ using permutation on $[\![4, 2, 2]\!]$, this operation transforms $H_{\rm x}$ to $\overline{H_{\rm x}}$, $H_{\rm z} \to \overline{H_{\rm z}}$ and $\gamma l_{\rm x} \to \overline{\gamma l_{\rm x}} = \gamma^2 \overline{l_{\rm x}}$.

Next, we apply $\pi$ to each qudit coordinate. On the B&C code, this is effectively permuting the $[\![4, 2, 2]\!]$ blocks associated to each qudit coordinate.

□

### 3. Small cyclic codes

We now present a concrete family of GF(4)-linear codes that (i) yield good distances and (ii) satisfy the sufficient condition of Proposition 14, guaranteeing that the corresponding B&C qubit codes are phantom. The two running examples from App. F 1, namely the $[\![3, 1, 2]\!]_4$ and $[\![5, 1, 3]\!]_4$ codes, are the smallest members of this family.

*Cyclic codes over* GF(4). We briefly recall standard facts about cyclic codes following [33, Ch. 7]. Cyclic codes can be defined over any finite field $F$; in the present context we take $F = \mathrm{GF}(4)$. A linear code $\mathscr{C} \subseteq F^n$ is *cyclic* if it is closed under cyclic shifts: whenever $(c_0, c_1, \ldots, c_{n-1}) \in \mathscr{C}$, then $(c_{n-1}, c_0, \ldots, c_{n-2}) \in \mathscr{C}$. Identify $(c_0, \ldots, c_{n-1})$ with the residue class of the polynomial

$$c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1} \in F[x]/(x^n - 1).$$

Then $\mathscr{C}$ is an ideal in $F[x]/(x^n - 1)$, hence $\mathscr{C} = \langle g(x) \rangle$ for a unique monic *generator polynomial* $g(x)$ dividing $x^n - 1$.

Let $h(x) := (x^n - 1)/g(x)$ be the *parity-check polynomial*. Any $c(x) \in \mathscr{C}$ can be written as $c(x) = f(x)g(x)$ for some $f(x) \in F[x]/(x^n - 1)$, and satisfies $h(x)c(x) = 0$.

The Euclidean dual $\mathscr{C}^\perp$ is also cyclic, with generator polynomial

$$g^\perp(x) = x^{\deg h(x)} h(x^{-1}).$$

We will also use the *conjugate* code $\overline{\mathscr{C}}$, obtained by applying the Frobenius automorphism entrywise:

$$(c_0, \ldots, c_{n-1}) \in \mathscr{C} \quad \Longleftrightarrow \quad (\overline{c}_0, \ldots, \overline{c}_{n-1}) = (c_0^2, \ldots, c_{n-1}^2) \in \overline{\mathscr{C}}.$$

The generator polynomial of a cyclic code of length $n$ over $F$ must be a factor of $x^n - 1$. Since we are interested in the case $F = \mathrm{GF}(4)$, let $m$ be the smallest integer such that $n$ divides $4^m - 1$. $\mathrm{GF}(4^m)$ is the splitting field of $x^n - 1$, i.e., $x^n - 1 = \prod_{i=0}^{n-1}(x - \xi^i)$, where $\xi \in \mathrm{GF}(4^m)$ is a primitive $n^{th}$ root of unity,

The *cyclotomic coset* mod $n$ over GF(4) which contains $s$ is $C_s^{(4)} = \{s, 4s, 4^2 s, \ldots, 4^{k(s)-1} s\}$, where $k(s)$ is the least integer such that $4^{k(s)} \equiv s \mod n$. Note that $m = k(1)$.

A polynomial $f(x) = \prod_{i \in K}(x - \xi^i)$ for $K$ a subset of $\{0, 1, \ldots, n-1\}$ has coefficients in GF(4) if and only if $k \in K \implies 4k \pmod n \in K$. Therefore, for the generator polynomial $g(x) = \prod_{i \in K}(x - \xi^i)$ of a cyclic code $\mathscr{C}$ to have coefficients in GF(4), $K$ is a union of cyclotomic cosets. The $n^{th}$ roots of unity $\{\xi^i : i \in K\}$ are called *the zeros of the code*. $c(x)$ belongs to $\mathscr{C}$ iff $c(\xi^i) = 0$ for all $i \in K$.

**Lemma 7.** *Given a cyclic code $\mathscr{C}$ with generator polynomial $g(x) = \prod_{i \in K}(x - \xi^i)$,*
*(a) the zeros of the dual code $\mathscr{C}^\perp$ are $\{\xi^i : -i \notin K\} = \{\xi^i : i \in -(\{0, 1, \ldots, n-1\} \backslash K) \mod n\}$;*
*(b) the zeros of the conjugate code $\overline{\mathscr{C}}$ are $\{\xi^{2i} : i \in K\}$.*

*Proof.* For (a), the dual code $\mathscr{C}^\perp$ is cyclic and has generator polynomial $g^\perp(x) = x^{\deg h(x)} h(x^{-1})$.
To see (b), note that for $(c_0, c_1, \ldots, c_{p-1}) \in \mathscr{C}$, one has $i \in K \Rightarrow c(\xi^i) = \sum_{j=0}^{n-1} c_j \xi^{ij} = 0$, then the conjugated codeword $(c_0^2, c_1^2, \ldots, c_{n-1}^2)$ satisfies $\sum_{j=0}^{n-1} c_j^2 \xi^{2ij} = (\sum_{j=0}^{n-1} c_j \xi^{ij})^2 = 0$ (2 times everything is zero in GF(4)), so $\xi^{2i}$ is a zero of the conjugated code. □

The classical QR code $Q_p$ is the $[p, \frac{1}{2}(p+1)]$ cyclic code with generator polynomial $\prod_{i\in R}(x-\xi^i)$, where $p$ is an odd prime, and $R$ is the subset of $\{1,\dots,p-1\}$ that are quadratic residue modulo $p$, i.e., can be written as $j^2 \mod p$ for some $j$. Note that we exclude 0 from $R$. Denote by $N$ the nonresidues modulo $p$, then $R \bigsqcup N = \{1,\dots,p-1\}$. The parity check generator of $Q_p$ is $(x-1)\prod_{i\in N}(x-\xi^i)$. Therefore, the generator polynomial of $Q_p^\perp$ is $(x-1)\prod_{i\in N}(x-\xi^{-i}) = \prod_{i\in -N\cup\{0\}}(x-\xi^i)$.

Next, we use the classical QR codes to construct a GF(4) CSS $[\![p,1]\!]_4$ quantum QR code. We choose the $X$ stabilizers to be $Q_p^\perp$. We show that, depending on $p \mod 8$, either taking $Q_p^\perp$ for the $Z$ stabilizers (self-orthogonal) or taking the Hermitian conjugate of $Q_p^\perp$ (hermitian self-orthogonal) satisfies the CSS constraint. This follows from [34], which we summarize below.

By the law of quadratic reciprocity: $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$, $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}$, which means 2 is a quadratic residue mod $p$ if $p \equiv \pm 1 \mod 8$, and $-1$ is a quadratic residue mod $p$ if $p \equiv 1, 5 \mod 8$. If $a$ and $b$ are both nonresidues mod $p$, or $a, b$ are both residues, then $ab$ is a residue. If one of $a, b$ is a residue and the other is a nonresidue, then $ab$ is a nonresidue.

**Proposition 15.** *It is possible to construct a* GF(4) *CSS* $[\![p,1]\!]_4$ *quantum quadratic residue code for a prime number* $p$ *that equals* $3, 5, 7$ *modulo* 8. *Take* $X$ *stabilizers to be* $Q_p^\perp$ *generated by* $\prod_{i\in -N\cup\{0\}}(x-\xi^i)$.
*(a) For* $p = 8k+3$, $Q_p^\perp$ *is self-orthogonal, i.e.,* $Q_p^\perp \subset (Q_p^\perp)^\perp = Q_p$, *and we can take* $Z$ *stabilizers to be* $Q_p^\perp$ *as well.*
*(b) For* $p = 8k-1$ *or* $p = 8k-3$, $Q_p^\perp$ *is hermitian self-orthogonal, i.e.,* $\overline{Q_p^\perp} \subset Q_p$, *and we can take* $Z$ *stabilizers to be* $\overline{Q_p^\perp}$.
*We call the constructed quantum code in (a) self-dual and in (b) hermitian self-dual. In both cases, an* $X$*-logical representative can be taken to be* $\prod_{i\in -N}(x-\xi^i)$.

*Proof.* To show that one cyclic code is contained in the other, we just need to show that its zero set contains the other. For primes of the form $8k+3$, we have $-1$ is a nonresidue and hence $-N = R$. The zeros of $Q_p^\perp$ are $\xi^i$ where $i \in -N \cup \{0\}$. Recall that the zeros of $Q_p$ are $\xi^j$ where $j \in R$. Since $R = -N \subset (-N \cup \{0\})$, we have $Q_p^\perp \subset Q_p$. An $X$-logical representative belongs to $Q_p \backslash Q_p^\perp$ and one can see $\prod_{i\in -N}(x-\xi^i)$ is indeed a valid choice. For $p = 8k-1$, $-1$ is a nonresidue but 2 is a residue. For $p = 8k-3$, $-1$ is a residue but 2 is a nonresidue. Therefore, $-2N = R$ in both cases. Therefore, by Lemma 7(b), the zeros of $\overline{Q_p^\perp}$ are $\xi^i$ where $i \in -2N \cup \{0\} = R \cup \{0\}$, while the zeros of $Q_p$ are $\{\xi^j \mid j \in R\}$. An $X$-logical representative belongs $(\overline{Q_p^\perp})^\perp \backslash Q_p^\perp = \overline{Q_p} \backslash Q_p^\perp$ and the zeros of $\overline{Q_p}$ are $\{\xi^j \mid j \in -N \cup \{0\}\}$, so $\prod_{i\in -N}(x-\xi^i)$ again works.

□

Note that for $p = 8k-1$, we can always find a binary parity check matrix [34, Thm. 38] for $Q_p^\perp$. This is not very interesting because one has binary QR codes of the same parameter, and we can obtain a phantom code by encoding each qubit of the $[\![4,2,2]\!]$ code using this binary QR code. In other words, B&C does not give any advantage over the usual $K = 1$ concatenation.

For primes of the form $8k+1$, the usual $R$ formed by residues will not work because both $-1$ and 2 are residues. However, it is sometimes possible to construct a hermitian self-dual quantum cyclic code of this length. For example, using the $[17, 9, 7]_4$ in [34, Table VI] with $K = C_1^{(4)} \bigcup C_3^{(4)}$, one can construct a $[\![17,1,7]\!]_4$ hermitian self-dual quantum cyclic code, and in general this is possible if $C_s^{(4)} \neq -2C_s^{(4)}$ for all $1 \le s \le p-1$ [34, Cor. 34]. Our observations for logical gates in the next two subsections also apply to these hermitian self-dual cyclic codes.

The qudit distance of the GF(4) quantum QR codes we used to construct those B&C codes in Table I is lower bounded by the distance listed in [34, Table VI] minus one. Therefore, by Proposition 13, the qubit distance of the B&C code is at least twice that of the qudit code. We numerically confirm that these are the actual code distances as listed in Table I.

### *a. Proof of phantom property*

We are now ready to prove that the B&C codes constructed from the above GF(4) cyclic codes of length $p$ (odd) are phantom. Recall from Proposition 14 that we only need to show that the logical level Frobenius transform $|\gamma\rangle_L$ to $|\gamma^2\rangle_L$ can be achieved through permutation only. Here we only show $|\gamma\rangle_L \mapsto |\alpha\gamma^2\rangle_L$ for some constant $\alpha \in \{1, \omega, \omega^2\}$ that depends on the code; this suffices because we can compose it with the $|\gamma\rangle_L \mapsto |\alpha^{-1}\gamma\rangle_L$ transform obtained from permutations inside each $[\![4,2,2]\!]$ codeblock.

At the qudit level, we first do a coordinate-wise conjugation $c_j \mapsto c_j^2$, then we permute the coordinates as $j \mapsto 2j \mod p$. At the qubit level of the B&C code, the first operation can be realized through permutations inside each $[\![4,2,2]\!]$ code, and the second can be achieved by permuting across the $[\![4,2,2]\!]$ codeblocks.

Going back to the qudit picture, let us first show that the $X$ and $Z$ stabilizers are preserved, again we just need to verify the zeros. Take an arbitrary stabilizer $(c_0, c_1, \ldots, c_{p-1})$ and let $\xi^i$ be a zero of it, i.e., $\sum_{j=0}^{p-1} c_j \xi^{ij} = 0$, then the two steps maps $c_j \mapsto c_j^2 \mapsto c_{j/2}^2$, where $j/2 = \frac{p+1}{2} j \mod p$. The polynomial of the transformed codeword becomes $c'(x) = \sum_{j=0}^{p-1} c_{j/2}^2 x^j$. One can see that $\xi^i$ is still a zero of this codeword, because $c'(\xi^i) = \sum_{j=0}^{p-1} c_{j/2}^2 \xi^{ij} = \sum_{j=0}^{p-1} c_j^2 \xi^{2ij} = (\sum_{j=0}^{p-1} c_j \xi^{ij})^2 = 0$. Since $(\sum_{j=0}^{p-1} c_j \xi^{ij})^2 = 0$ implies $\sum_{j=0}^{p-1} c_j \xi^{ij} = 0$, one can similarly show that if $\xi^i$ is a zero of $c'(x)$, then it is also a zero of $c(x)$. Consequently, the zeros of the common divisor, i.e., the generator polynomial, of the stabilizer codewords, are preserved under this mapping. Therefore, the stabilizers are preserved.

Next, we show that the $X$-logical $\gamma l_{\mathrm{x}}$ is mapped to $\alpha\gamma^2 l_{\mathrm{x}}$ up to stabilizers, for a constant $\alpha \in \{1, \omega, \omega^2\}$. Recall from Proposition 15 that the $X$-stabilizers are generated by $\prod_{i \in -N \cup \{0\}} (x - \xi^i)$ and we can choose $l(x) = \prod_{i \in -N} (x - \xi^i) := \sum_{i=0}^{p-1} l_i x^i$ to represent the $X$ logical $l_{\mathrm{x}}$. By the codeword to polynomial correspondence, we can write $l_{\mathrm{x}} = (l_0, \cdots, l_{p-1})$, and $l_0, \cdots, l_{p-1} \in \mathrm{GF}(4)$. Moreover, $l(\xi^i) = \sum_{j=0}^{p-1} l_j \xi^{ij} = 0$ for $i \in -N$. Next, we show that the constant $\alpha$ is $l(1) = \sum_{i=0}^{p-1} l_i$. Note that $l(1) \in \mathrm{GF}(4)$ because $l_0, \cdots, l_{p-1} \in \mathrm{GF}(4)$; $l(1)$ is not zero because $\xi^0 = 1$ is not a root of $l(x)$.

Under the two-step transform above, $\gamma l_{\mathrm{x}}$ transforms as $(\gamma l_0, \ldots, \gamma l_{p-1}) \mapsto (\gamma^2 l_0^2, \ldots, \gamma^2 l_{p-1}^2) \mapsto (\gamma^2 l_{0/2}^2, \ldots, \gamma^2 l_{(p-1)/2}^2)$. Again, the new generating polynomial $l'(x) = \sum_{j=0}^{p-1} \gamma^2 l_{j/2}^2 x^j$ has the zeros exactly at the same places as $l(x)$. Moreover, $\gamma l_{\mathrm{x}}$ is mapped to $l(1)\gamma^2 l_{\mathrm{x}}$, i.e., $\alpha = l(1) = \sum_{j=0}^{p-1} l_j \in \{1, \omega, \omega^2\}$. This is because $l'(x)$ and $l(1)\gamma^2 l_{\mathrm{x}}$ differs by an $X$-stabilizer: $\Delta l(x) := l'(x) - l(1)\gamma^2 l(x) = \gamma^2(\sum_{j=0}^{p-1} l_{j/2}^2 x^j - l(1) \sum_{j=0}^{p-1} l_j x^j)$ can be divided by $\prod_{i \in -N \cup \{0\}} (x - \xi^i)$. For $i \in -N$, one has $\Delta l(\xi^i) = \gamma^2(0 - 0) = 0$ and $\Delta l(\xi^0) = \gamma^2(\sum_{j=0}^{p-1} l_{j/2}^2 - l(1) \sum_{j=0}^{p-1} l_j) = \gamma^2(l(1)^2 - l(1)^2) = 0$.

### *b. Logical gates*

Next, we establish other Clifford logical gates that the B&C codes admit besides $\overline{\mathrm{CNOT}}$; they are by no means exhaustive. Denote by $H_{\mathrm{x}}$ (resp. $H_{\mathrm{z}}$) and $l_{\mathrm{x}}$ (resp. $l_{\mathrm{z}}$) the $X$ (resp. $Z$) type PCMs and logical of the $[\![p,1]\!]_4$ GF(4) codes; entries are in GF(4). For the code obtained after B&C, denote by $H'_{\mathrm{x}}$ and $H'_{\mathrm{z}}$ its binary PCMs.

The per qudit $[\![4,2,2]\!]$ encoding gives extra stabilizers $XXXX$ and $ZZZZ$, and it maps the $X$ and $Z$ operators on each qudit to a Pauli string on four qubits as follows (up to these extra stabilizers). $X^\omega \mapsto XXII \equiv IIXX$ and $Z^{\omega^2} \mapsto IIZZ$; $X^{\omega^2} \mapsto XIXI \equiv IXIX$ and $Z^\omega \mapsto IZIZ$; $X^1 \mapsto IXXI$ and $Z^1 \mapsto IZZI$.

For simplicity, denote the per qudit $[\![4,2,2]\!]$ encoding as $\mathrm{enc}(\cdot)$ and rewrite the above as, e.g., $\mathrm{enc}(X^\omega) = 1100 \equiv 0011 = \mathrm{enc}(Z^{\omega^2})$. One can see that in the binary PCM, $X^\gamma$ is mapped to the equivalent thing as $Z^{\gamma^2}$, i.e., $\mathrm{enc}(X^\gamma) = \mathrm{enc}(Z^{\gamma^2})$, $\forall \gamma \in \mathrm{GF}(4)$.

The two logical pairs of the B&C code can be written as $(\mathrm{enc}_{\mathrm{x}}(\omega l_{\mathrm{x}}),\ \mathrm{enc}_{\mathrm{z}}(\omega^2 l_{\mathrm{x}}))$ and $(\mathrm{enc}_{\mathrm{x}}(\omega l_{\mathrm{z}}),\ \mathrm{enc}_{\mathrm{z}}(\omega^2 l_{\mathrm{z}}))$. Here, $\mathrm{enc}_{\mathrm{x/z}}(l)$ for $l = (l_0, \ldots, l_{p-1}) \in \mathbb{F}_4^p$ means $\mathrm{enc}_{\mathrm{x}}(l) := (\mathrm{enc}(X^{l_0}), \ldots, \mathrm{enc}(X^{l_{p-1}})) \in \mathbb{F}_2^{4p}$ for $X$-type and $\mathrm{enc}_{\mathrm{z}}(l) := (\mathrm{enc}(Z^{l_0}), \ldots, \mathrm{enc}(Z^{l_{p-1}})) \in \mathbb{F}_2^{4p}$ for $Z$-type. We can shorthand the relation $\mathrm{enc}(X^\gamma) = \mathrm{enc}(Z^{\gamma^2})$ as $\mathrm{enc}_{\mathrm{x}}(\gamma) = \mathrm{enc}_{\mathrm{z}}(\gamma^2)$.

**Proposition 16.** *For the phantom B&C codes constructed from the hermitian self-dual* GF(4) *codes (not necessarily cyclic), transversal Hadamard implements logical $HH$ up to* $\mathrm{SWAP}_{12}$*, and transversal $S$ implements logical $CZ$ up to Pauli corrections.*

*Proof.* By definition, $H_{\mathrm{z}} = \overline{H_{\mathrm{x}}}$, and by $\mathrm{enc}_{\mathrm{x}}(\gamma) = \mathrm{enc}_{\mathrm{z}}(\gamma^2)$, we have $\mathrm{rs}(H'_{\mathrm{x}}) = \mathrm{rs}(H'_{\mathrm{z}})$. Therefore, stabilizers are preserved under transversal Hadamard. For the logical action, first notice that we can take $l_{\mathrm{z}} = \overline{l_{\mathrm{x}}}$ because $l_{\mathrm{z}} \in H_{\mathrm{x}}^\perp \backslash H_{\mathrm{z}} = H_{\mathrm{x}}^\perp \backslash \overline{H_{\mathrm{x}}}$ and $l_{\mathrm{x}} \in H_{\mathrm{z}}^\perp \backslash H_{\mathrm{x}} = \overline{H_{\mathrm{x}}}^\perp \backslash H_{\mathrm{x}}$. Then one can see that the $X$-type logical $\mathrm{enc}_{\mathrm{x}}(\omega l_{\mathrm{x}})$ is mapped to $Z$-type logical $\mathrm{enc}_{\mathrm{x}}(\omega l_{\mathrm{x}}) = \mathrm{enc}_{\mathrm{z}}(\omega^2 \overline{l_{\mathrm{x}}}) = \mathrm{enc}_{\mathrm{z}}(\omega^2 l_{\mathrm{z}})$ under transversal Hadamard. Similarly, the $X$-type logical $\mathrm{enc}_{\mathrm{x}}(\omega^2 l_{\mathrm{x}})$ is mapped to the $Z$-type logical $\mathrm{enc}_{\mathrm{z}}(\omega l_{\mathrm{z}})$. Therefore, the logical action of transversal Hadamard is $HH$ and $\mathrm{SWAP}_{12}$.

Since $\mathrm{rs}(H'_{\mathrm{x}}) = \mathrm{rs}(H'_{\mathrm{z}})$, one can also see that stabilizers are preserved under transversal $S$ but possibly up to a phase. We show that the phase is $+1$ and so the codespace is preserved as follows. An $X$ stabilizer $s_{\mathrm{x}}$ of the GF(4) code is mapped to itself times a $Z$ stabilizer $\overline{s_{\mathrm{x}}}$ (on the qudit level) with a phase $i^{2\,\mathrm{wt}_\omega(s_{\mathrm{x}}) + 2\,\mathrm{wt}_{\omega^2}(s_{\mathrm{x}}) + 2\,\mathrm{wt}_1(s_{\mathrm{x}})}$ (on the qubit level), where $\mathrm{wt}_\omega(s_{\mathrm{x}}), \mathrm{wt}_{\omega^2}(s_{\mathrm{x}}), \mathrm{wt}_1(s_{\mathrm{x}})$ count the number of $\omega, \omega^2, 1$ appearing in $s_{\mathrm{x}}$. Since $s_{\mathrm{x}} \cdot \overline{s_{\mathrm{x}}} = 0$, we have

$\mathrm{wt}_\omega(s_\mathrm{x}) + \mathrm{wt}_{\omega^2}(s_\mathrm{x}) + \mathrm{wt}_1(s_\mathrm{x})$ even. The logical CZ action resulting from transversal $S$ can be verified similarly to the proof above for the transversal Hadamard. □

This proposition is relevant for $p = 8k+5$ and $p = 8k+1$ if a hermitian self-dual quantum code can be constructed. Since CZ, $HH$ and phantom CNOTs do not generate the full Clifford group, we want to find a way to do logical $SS$ via fold diagonal gates. However, although the solver in [41] says logical $SS$ from fold is indeed possible for $[\![20, 2, 6]\!]$ constructed from $[\![5, 1, 3]\!]_4$, we do not know how to generalize these findings to larger codes in this family. It is known that the automorphism group of the classical extended QR codes contains a subgroup isomorphic to $\mathrm{PSL}(2, \mathbb{F}_p)$, and the base code[10] $[6, 3, 4]_4$ of $[\![20, 2, 6]\!]$ is the only known case over GF(4) in which the group is bigger than this [33, Ch. 16]. Therefore, it is also possible that $[\![20, 2, 6]\!]$ is the only one in the family that allows fold diagonal gates.

**Proposition 17.** *For the phantom B&C codes constructed from the self-dual* GF(4) *QR codes of length $p = 8k + 3$, transversal Hadamard together with the qudit coordinate permutation $i \mapsto 2i \mod p$ implements logical $HH$ up to* $\mathrm{SWAP}_{12}$. *Logical $SS$ can be achieved up to Pauli corrections through folding each $[\![4, 2, 2]\!]$ codeblock using the involution $\tau_{SS}$: CZ between qubits* 2 *and* 3 *and $S$ on qubits* 1 *and* 4. *Logical CZ can also be achieved up to Pauli corrections through folding; $\tau_{\mathrm{CZ}}$ is the involution on the qudit coordinate $i \mapsto -i \mod p$.*

*Proof.* Since $\mathrm{enc}_\mathrm{x}(\gamma) = \mathrm{enc}_\mathrm{z}(\gamma^2)$, the qubit-level transversal Hadamard transforms the qudit PCM $H_\mathrm{x}$ into its conjugate on the qudit level. Since $H_\mathrm{z} = H_\mathrm{x}$, we need to counteract this conjugation through coordinate permutation. As we have seen before, $i \mapsto 2i \mod p$ effectively doubles the exponents of the zeros of a cyclic code and transforms the code into its conjugate. Therefore, the codespace is preserved under this SWAP-transversal Hadamard. For the logical action, note that we can take $l_\mathrm{z} = l_\mathrm{x}$ for the self-dual codes. The $X$-logical $\mathrm{enc}_\mathrm{x}(\omega l_\mathrm{x})$ is mapped to the $Z$-type $\mathrm{enc}_\mathrm{x}(\omega l_\mathrm{x}) = \mathrm{enc}_\mathrm{z}(\omega^2 \overline{l_\mathrm{x}}) = \mathrm{enc}_\mathrm{z}(\omega^2 \overline{l_\mathrm{z}})$ under transversal Hadamard, and the coordinate permutation further maps $\omega^2 \overline{l_\mathrm{z}}$ to $l(1)\omega^2 l_\mathrm{z} = \omega^2 l_\mathrm{z}$ (up to some $Z$ stabilizers). The constant $l(1) = \sum_{i=0}^{p-1} l_i$, where $l(x) = \sum_{i\in -N}(x - \xi^i) = \sum_{i=0}^{p-1} l_i x^i$, appearing in App. F 3 a for this coordinate permutation turns out to be 1 for self-dual codes constructed from $p = 8k+3$ QR codes.

One can verify $l(1) = \omega + \omega^2 + 0 = 1$ for $p = 3$ directly. For $p > 3$, $l(1) = \prod_{i\in -N}(1 - \xi^i) = \prod_{i\in -N} \xi^i(\xi^{-i} - 1) = \xi^{-\sum_{i\in N} i} \prod_{i\in N}(1 - \xi^i) = \prod_{i\in N}(1 - \xi^i)$, where in the second equality we use $1 = -1$ in GF(4) and in the last step $\sum_{i\in N} i \equiv \sum_{i\in R} i = \sum_{j=1}^{(p-1)/2} j^2 = p(p^2 - 1)/24 \equiv 0 \mod p$. Since $-1$ is a nonresidue for $p \equiv 3 \mod 4$, one has $-N = R$, and $l(1)^2 = \prod_{i=1}^{p-1}(1 - \xi^i) = \frac{\xi^p - 1}{\xi - 1} = 1 \Rightarrow l(1) = 1$.

To summarize the logical action of transversal $H$ together with the coordinate permutation $i \mapsto 2i \mod p$, the $X$-logical $\mathrm{enc}_\mathrm{x}(\omega l_\mathrm{x})$ transforms to $Z$-logical $\mathrm{enc}_\mathrm{z}(\omega^2 l_\mathrm{z})$. One can similarly show that the $X$-logical $\mathrm{enc}_\mathrm{x}(\omega^2 l_\mathrm{x})$ transforms to $Z$-logical $\mathrm{enc}_\mathrm{z}(\omega l_\mathrm{z})$.

Under the fold gate following involution $\tau_{SS}$, each $X$-type qudit $\gamma$ is mapped to$\tau(\mathrm{enc}_\mathrm{x}(\gamma)) = \mathrm{enc}_\mathrm{x}(\gamma^2) = \mathrm{enc}_\mathrm{z}(\gamma)$. Therefore, the per-coordinate folding maps an $X$ stabilizer $\mathrm{enc}_\mathrm{x}(s_\mathrm{x})$ to itself and the $Z$-type $\mathrm{enc}_\mathrm{z}(s_\mathrm{x})$ and a phase $i^{\mathrm{wt}_\omega(s_\mathrm{x}) + \mathrm{wt}_{\omega^2}(s_\mathrm{x}) + 2\,\mathrm{wt}_1(s_\mathrm{x})} = \pm 1$ because $s_\mathrm{x} \cdot s_\mathrm{x} = 0$; one can fixes the $-1$ phase by applying destabilizers [22]. The $X$-logical $\mathrm{enc}_\mathrm{x}(\omega l_\mathrm{x})$ is mapped to itself times its pairing $Z$-logical $\mathrm{enc}_\mathrm{z}(\omega l_\mathrm{x}) = \mathrm{enc}_\mathrm{z}(\omega l_\mathrm{z})$ with a phase $i^{\mathrm{wt}_\omega(\omega l_\mathrm{x}) + \mathrm{wt}_{\omega^2}(\omega l_\mathrm{x}) + 2\,\mathrm{wt}_1(\omega l_\mathrm{x})} = i^{\mathrm{wt}_1(l_\mathrm{x}) + \mathrm{wt}_w(l_\mathrm{x}) + 2\,\mathrm{wt}_{\omega^2}(l_\mathrm{x})} = \pm i$. The last step follows because $l(1) = 1 \Rightarrow \mathrm{wt}_\omega(l_\mathrm{x}) \equiv \mathrm{wt}_{\omega^2}(l_\mathrm{x}) \not\equiv \mathrm{wt}_1(l_\mathrm{x}) \mod 2$. We can apply a $Z$-logical operator to fix the phase if it is $-i$. One can similarly show that the other $X$-logical is mapped to itself times its pairing $Z$-logical with a $\pm i$ phase. The logical effect of this fold gate is thus $SS$ up to Pauli corrections.

Lastly, we show that the fold gate following the involution $\tau_{\mathrm{CZ}}$ implements logical CZ. Note that under $i \mapsto -i \mod p$, a zero of a polynomial is mapped to its inverse. Therefore, the generator polynomial for $X/Z$ stabilizers $\prod_{i\in -N\cup\{0\}}(x - \xi^i)$ is mapped to $\prod_{i\in N\cup\{0\}}(x - \xi^i) = \prod_{i\in -2N\cup\{0\}}(x - \xi^i)$, the last step $N = -2N$ is because both $-1$ and 2 are nonresidues modulo $p = 8k + 3$. Also, recall that coordinate-wise conjugation doubles the exponent of zeros. Therefore, by $\mathrm{enc}_\mathrm{x}(\gamma) = \mathrm{enc}_\mathrm{z}(\gamma^2)$, each $X$ stabilizer is mapped to itself times a $Z$ stabilizer. Since the only fixed point is the qudit at coordinate 0, the phase accumulated is either 1 or $-1$, and the latter case can be fixed by applying a destabilizer. The $X$-logical $\mathrm{enc}_\mathrm{x}(\omega l_\mathrm{x}) = (\mathrm{enc}_\mathrm{x}(\omega l_0), \ldots, \mathrm{enc}_\mathrm{x}(\omega l_{p-1}))$ is mapped to itself times $Z$ type $(\mathrm{enc}_\mathrm{x}(\omega l_{-0}), \ldots, \mathrm{enc}_\mathrm{x}(\omega l_{-(p-1)})) = (\mathrm{enc}_\mathrm{z}(\omega^2 l^2_{-0}), \ldots, \mathrm{enc}_\mathrm{z}(\omega^2 l^2_{-(p-1)}))$. The polynomial of this new codeword $\omega^2 \sum_{i=0}^2 l^2_{-i} x^i$ has the same zero as $\omega^2 l_\mathrm{x} = \omega^2 l_\mathrm{z} = \omega^2 \prod_{i\in -N}(x - \xi^i)$ (again because of both $-1$ and 2 are nonresidues). Moreover, their difference is a $Z$-stabilizer, i.e., $(x - \xi^0)$ divides the difference, because $l(1) = 1$. Therefore, the $X$-logical $\mathrm{enc}_\mathrm{x}(\omega l_\mathrm{x})$ is mapped to itself and the $Z$-logical $\mathrm{enc}_\mathrm{z}(\omega^2 l_\mathrm{z})$. Similar arguments hold for the other $X$-logical $\mathrm{enc}_\mathrm{x}(\omega^2 l_\mathrm{x})$. The logical effect of the fold gate is thus CZ. □

[10] When puncturing the extended QR codes at $\infty$ to construct a $K = 1$ quantum code, one of the generator $y \mapsto -\frac{1}{y}$ of $\mathrm{PSL}(2, \mathbb{F}_p)$ becomes unavailable. The only possible qudit coordinate involution is $y \mapsto -y$ if $-1$ is a residue.

## Appendix G: Other code constructions

Here we discuss additional avenues for code construction. First, in App. G 1, we describe the construction of CSS phantom codes from suitable pairs of classical linear codes via the hypergraph product. In App. G 2, we examine a family of $d = 2$ phantom codes built by gluing $[\![4, 2, 2]\!]$ codes. Lastly, in App. G 3, we examine codes built from phantom codes. These codes are not phantom as they support products rather than individually addressable $\overline{\text{CNOT}}$ gates implemented by qubit permutations, but are related to prior literature.

### 1. Hypergraph product phantom codes

To start, we first review the notion of automorphisms on classical linear codes.

**Definition 9** (Code automorphism of a classical linear code)**.** *For an $[n, k]$ linear code over $\mathbb{F}_2$ with generator matrix $G \in \mathbb{F}_2^{k\times n}$, a code automorphism is a coordinate permutation $\sigma \in \mathrm{S}_n$ such that $G\sigma = VG$ for some invertible matrix $V \in \mathrm{GL}(k, \mathbb{F}_2)$.*

In other words, permuting the coordinates of bits by $\sigma$ maps a codeword basis to another codeword basis, and the codespace spanned by these bases is preserved.

**Definition 10** (Tanner graph automorphism of a classical linear code [35])**.** *For an $[n, k]$ linear code over $\mathbb{F}_2$ with parity-check matrix $H \in \mathbb{F}_2^{m\times n}$, a Tanner graph automorphism is a code automorphism such that (additionally) $H\sigma = \pi H$ where $\pi \in \mathrm{S}_m$.*

To illustrate the difference, consider the $[3, 1]$ repetition code with generator and parity-check matrices

$$G = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \qquad H = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}. \tag{G1}$$

Any cyclic permutation of the three coordinates leaves $G$ unchanged and so is a code automorphism. However, a rightward cyclic shift of the coordinates maps $H$ to

$$H' = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \tag{G2}$$

which is not the same as $H$ up to row permutations, and therefore this permutation is not a Tanner graph automorphism of the code. In contrast, swapping the first and the last coordinates preserves $H$ up to swapping the two rows, and is a Tanner graph automorphism. This highlights the distinction: Tanner graph automorphisms are a subset of code automorphisms.

We construct CSS phantom codes using the hypergraph product of classical linear codes with Tanner graph automorphisms, as a corollary to Ref. [35, Thm. 4.2].

**Theorem 7** (Phantom hypergraph product codes)**.** *Let $H_1$ be an $m_1 \times n_1$ parity-check matrix of an $[n_1, k, d_1]$ classical linear code whose Tanner graph automorphism group is $\mathrm{GL}(k, \mathbb{F}_2)$. Let $H_2$ be an $m_2 \times n_2$ parity-check matrix of an $[n_2, 1, d_2]$ classical linear code. Then, the hypergraph product of $H_1$ and $H_2$ yields a CSS phantom code with parameters*

$$[\![n_1 n_2 + m_1 m_2,\ k,\ \min(d_1, d_2)]\!]. \tag{G3}$$

*Proof.* In the hypergraph product construction, the $X$- and $Z$-type stabilizer generator matrices take the forms

$$H_{\mathrm{x}} = (H_1 \otimes I_{n_2} \ \mid \ I_{m_1} \otimes H_2^{\top}), \qquad H_{\mathrm{z}} = (I_{n_1} \otimes H_2 \ \mid \ H_1^{\top} \otimes I_{m_2}), \tag{G4}$$

where the vertical bar separates the "left" data sector with $n_1 n_2$ qubits from the "right" data sector with $m_1 m_2$ qubits. We adopt the canonical logical basis

$$L_{\mathrm{x}} = (\{e_j\} \otimes g_2 \ \mid \ 0), \qquad L_{\mathrm{z}} = (G_1 \otimes \epsilon \ \mid \ 0), \tag{G5}$$

where $G_1$ is a $k \times n_1$ generator matrix for the $[n_1, k, d_1]$ code, $\epsilon$ is a length-$n_2$ row vector outside the row span of $H_2$, $\{e_j\}$ denotes a $k \times n_1$ matrix of linearly independent unit vectors outside the row span of $H_1$, and $g_2$ is the only codeword of the $[n_2, 1, d_2]$ code.

Because the commutation relations of logical Pauli operators are preserved under physical qubit permutations, it suffices to track the action on $L_z$ to verify that a permutation implements a desired logical Clifford. In particular, for a $\overline{\text{CNOT}}$ from logical qubit $a$ to $b$, take

$$V = \mathbb{I}_k + e_{ba}, \tag{G6}$$

where $e_{ba}$ is the matrix with a 1 in the $(b, a)$ position and zeros elsewhere. Note that $V \in \mathrm{GL}_k(\mathbb{F}_2)$. Since the $[n_1, k, d_1]$ code admits the full $\mathrm{GL}(k, \mathbb{F}_2)$ Tanner graph automorphism group, for such a $V$ there exist permutations $\sigma_V \in S_{n_1}$ and $\pi_V \in S_{m_1}$ such that

$$G_1\sigma_V = VG_1, \qquad H_1\sigma_V = \pi_V H_1. \tag{G7}$$

By Ref. [35, Thm. 4.2], the qubit permutation

$$\rho_V = (\sigma_V \otimes \mathbb{I}_{n_2}) \oplus (\pi_V \otimes \mathbb{I}_{m_2}) \tag{G8}$$

preserves the stabilizer group and acts on the logical operators as

$$L_z\rho_V = (G_1\sigma_V \otimes \epsilon\,\mathbb{I}_{n_2} \mid 0) = (VG_1 \otimes \epsilon \mid 0) = VL_z, \tag{G9}$$

noting that $\epsilon$ is a single row. Thus, $\rho_V$ implements a $\overline{\text{CNOT}}$ from qubit $a$ to $b$. □

As a result of Theorem 7, we can construct CSS phantom codes from the hypergraph product of a classical simplex code with a repetition code. Let $H_1$ be the parity-check matrix of the $[n_1 = 2^k - 1, k, 2^{k-1}]$ simplex code, where the $m_1 = n_1(n_1 - 1)/6$ rows correspond to all weight-3 checks (i.e. all weight-3 codewords of the dual of simplex code, the Hamming code). Let $H_2$ be the parity-check matrix of the $[n_2 = 2^{k-1}, 1, 2^{k-1}]$ repetition code, with $m_2 = n_2 - 1$ rows. Then the hypergraph product of $H_1$ and $H_2$ yields a CSS phantom code with parameters

$$[\![n_1n_2 + m_1m_2 = \mathcal{O}(2^{3k}), k, 2^{k-1}]\!]. \tag{G10}$$

The smallest members of this family are the $[\![7, 2, 2]\!]$ and $[\![49, 3, 4]\!]$ codes.

### 2. Glued $[\![4, 2, 2]\!]$ codes as a family of $d = 2$ phantom codes

Detailed examination of our exhaustive code enumeration (see App. C) results revealed that a class of $n$-optimal CSS phantom codes for a given $d_z$, when $k = 2$ and $d_x = 2$, possess a common underlying structure and can be understood as gluings of the the $[\![4, 2, 2]\!]$ code. The Hadamard-dual of these codes exchange $d_x, d_z$ distances (thereby fixing $d_z = 2$ instead). We describe this construction below.

**Theorem 8** (Glued $[\![\mathbf{4, 2, 2}]\!]$ phantom codes)**.** *There exist CSS phantom codes with parameters* $[\![n = 4m, k = 2, d_x = 2, d_z = 2m]\!]$ *and* $[\![n = 4m - 1, k = 2, d_x = 2, d_z = 2m - 1]\!]$ *for any integer* $m \geq 1$.

*Proof.* We illustrate the constructions in Fig. 16. First we describe the $n = 4m$ construction. We use $m$ copies of the $[\![4, 2, 2]\!]$ code and glue them together with $X$-type stabilizers to form a "tube". Labelling the physical qubits as shown in Fig. 16a, a basis for the logical operators of the glued code is $\overline{X}_1 = X_0X_1 \equiv \cdots \equiv X_{4i}X_{4i+1}$, $\overline{X}_2 = X_1X_2 \equiv \cdots \equiv X_{4i+1}X_{4i+2}$ for all integer $0 \leq i < m$, where $\equiv$ denotes equivalence up to stabilizers, and $\overline{Z}_1 = Z_1Z_2Z_5Z_6 \cdots Z_{4m-3}Z_{4m-2}$, $\overline{Z}_2 = Z_0Z_1Z_4Z_5 \cdots Z_{4m-4}Z_{4m-3}$. The $\overline{\text{CNOT}}_{12}$ gate can be implemented by swapping the qubits $(1, 2), (5, 6), \ldots, (4m-3, 4m-2)$, and $\overline{\text{CNOT}}_{21}$ by $(0, 1), (4, 5), \ldots, (4m-4, 4m-3)$. From this construction, we can further remove a physical qubit, as shown in Fig. 16b. Consequently, an $X$-type stabilizer is removed and a $Z$-type stabilizer is reshaped to a triangle. This accounts for the $n = 4m - 1$ family of codes. □

### 3. Codes built from phantom codes

As discussed in the main text, the $[\![2^{2r}, \binom{2r}{r}, 2^r]\!]$ quantum Reed–Muller codes [6, 19] implement products of $\overline{\text{CNOT}}$ gates across logical qubits via qubit permutations. However, they do not support individually addressable $\overline{\text{CNOT}}$ gates, except in the special case of $r = 1$, which reduces to the hypercube codes. These codes (for $r > 1$) therefore lie outside our definition of phantom codes. Here we comment on constructions for such kinds of "phantom-like" codes to connect with existing literature, starting from genuine phantom codes.

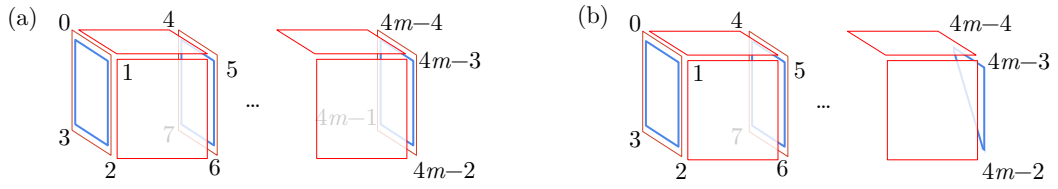



Figure 16. **Glued $[\![4,2,2]\!]$ phantom codes.** Red and blue polygons denote $X$- and $Z$-type stabilizers, respectively.

*a. Connecting a CSS code and its Hadamard-dual*

We describe a construction that generates code families from phantom codes, though the resulting codes are not phantom. This scheme generalizes the well-known procedure that produces the $[\![16,6,4]\!]$ tesseract code from the $[\![8,3,2]\!]$ code and its dual.

**Proposition 18** (Connecting a CSS code and its Hadamard-dual)**.** *Consider connecting an $[\![n,k,(d_\mathrm{x},d_\mathrm{z})]\!]$ CSS code (the primal code) and its Hadamard-dual, by performing transversal* CNOT *gates controlled on the primal and targeting the dual codeblock. The resulting code has distance $d'=d'_\mathrm{x}=d'_\mathrm{z}$ given by*

$$d' = \min\left[d_\mathrm{z},\ \min_{\substack{l_\mathrm{x}\in\mathcal{L}_\mathrm{x}\\ g_\mathrm{z}\in\mathrm{rs}(H_\mathrm{x})^\perp}} (2|l_\mathrm{x}|+|g_\mathrm{z}|-2|l_\mathrm{x}\cap g_\mathrm{z}|)\right], \tag{G11}$$

*where the stabilizer generators of the primal code are denoted $H_\mathrm{x},H_\mathrm{z}$ in symplectic representation, $\mathcal{L}_\mathrm{x}=\mathrm{rs}(H_\mathrm{z})^\perp\setminus\mathrm{rs}(H_\mathrm{x})$ is accordingly the vector space of $X$-type logical operators of the primal code, and $\mathrm{rs}(H_\mathrm{x})^\perp$ is the vector space of $Z$-type stabilizers and logical operators of the primal code.*

*Proof.* The stabilizer generator matrix of the primal code in the symplectic representation (see App. A 1) has the structure $\mathrm{diag}(H_\mathrm{x}\,|\,H_\mathrm{z})$, where the $H_\mathrm{x}$ and $H_\mathrm{z}$ blocks describe the $X$- and $Z$-type stabilizer generators respectively. The stabilizer generator matrix of the dual code is accordingly $\mathrm{diag}(H_\mathrm{z}\,|\,H_\mathrm{x})$. Taking the tensor product of these codes (i.e. placing them side-by-side) gives an $[\![2n,2k]\!]$ code with the stabilizer generator matrix $\mathrm{diag}(H_\mathrm{x},H_\mathrm{z}\,|\,H_\mathrm{z},H_\mathrm{x})$. Applying transversal CNOT gates between these two codes, where the CNOT gates are controlled by qubits in the primal code and target corresponding qubits in dual code, yields an $[\![2n,2k]\!]$ code with the stabilizer generator matrix

$$H = \left(\begin{array}{cc|cc} H_\mathrm{x} & H_\mathrm{x} & & \\ & H_\mathrm{z} & & \\ \hline & & H_\mathrm{z} & \\ & & H_\mathrm{x} & H_\mathrm{x} \end{array}\right). \tag{G12}$$

To observe that Eq. (G11) describes the distance of the resulting code $H$, we write out explicitly the set of logical operators on the code. By definition, the $Z$-type logical operators for the code must commute with all $X$-type stabilizers but are not $Z$-type stabilizers themselves. Denoting the bitstring representing a $Z$-type logical operator as $(u,v)$, where both $u$ and $v$ are of length $n$, the former condition means $H_\mathrm{x}v^\mathsf{T}=0$ and $H_\mathrm{x}(u^\mathsf{T}+v^\mathsf{T})=0$—i.e., $v\in\mathrm{rs}(H_\mathrm{z})^\perp$ and $u+v\in\mathrm{rs}(H_\mathrm{x})^\perp$. For simplicity, denote $v$ as $g_\mathrm{x}$ and $u+v$ as $g_\mathrm{z}$. Analogous conditions apply for the $X$-type logical operators of the code. The vector space of logical operators of the code $H$ can thereby be written:

$$\begin{aligned}\mathcal{L}'_\mathrm{x} &= \left\{(g_\mathrm{x},g_\mathrm{x}+g_\mathrm{z}) \;\middle|\; g_\mathrm{x}\in\mathrm{rs}(H_\mathrm{z})^\perp, g_\mathrm{z}\in\mathrm{rs}(H_\mathrm{x})^\perp\right\}\setminus\left\{(s_\mathrm{x},s_\mathrm{x}+s_\mathrm{z}) \;\middle|\; s_\mathrm{x}\in\mathrm{rs}(H_\mathrm{x}), s_\mathrm{z}\in\mathrm{rs}(H_\mathrm{z})\right\},\\ \mathcal{L}'_\mathrm{z} &= \left\{(g_\mathrm{x}+g_\mathrm{z},g_\mathrm{x}) \;\middle|\; g_\mathrm{x}\in\mathrm{rs}(H_\mathrm{z})^\perp, g_\mathrm{z}\in\mathrm{rs}(H_\mathrm{x})^\perp\right\}\setminus\left\{(s_\mathrm{z}+s_\mathrm{x},s_\mathrm{x}) \;\middle|\; s_\mathrm{x}\in\mathrm{rs}(H_\mathrm{x}), s_\mathrm{z}\in\mathrm{rs}(H_\mathrm{z})\right\}.\end{aligned} \tag{G13}$$

The weight of a logical operator is $w=|g_\mathrm{z}|+2|g_\mathrm{x}|-2|g_\mathrm{x}\cap g_\mathrm{z}|$ for both $X$- and $Z$-types. We consider the minimum weight by cases:

- In the case that $g_\mathrm{x}:=l_\mathrm{x}\in\mathrm{rs}(H_\mathrm{z})^\perp\setminus\mathrm{rs}(H_\mathrm{x})=\mathcal{L}_\mathrm{x}$, that is, $g_\mathrm{x}$ is a nontrivial logical operator of the primal code, the weight $w$ reduces to the $(2|l_\mathrm{x}|+|g_\mathrm{z}|-2|l_\mathrm{x}\cap g_\mathrm{z})$ term in Eq. (G11).
- Otherwise $g_\mathrm{x}:=s_\mathrm{x}\in\mathrm{rs}(H_\mathrm{x})$. Then $w=|g_\mathrm{z}|+2(|s_\mathrm{x}|-|s_\mathrm{x}\cap g_\mathrm{z}|)\geq|g_\mathrm{z}|$ where equality holds when $s_\mathrm{x}=0$, and moreover $g_\mathrm{z}\notin\mathrm{rs}(H_\mathrm{z})$ in order for $(g_\mathrm{x},g_\mathrm{x}+g_\mathrm{z})$ or $(g_\mathrm{x}+g_\mathrm{z},g_\mathrm{x})$ to be a logical operator (otherwise the operator is simply a stabilizer). That is, $g_\mathrm{z}$ is a $Z$-type logical operator of the primal code, and therefore $|g_\mathrm{z}|\geq d_\mathrm{z}$. This accounts for the $d_\mathrm{z}$ term in Eq. (G11).

In fact, given a set of logical Pauli operators for the primal code, $\{l_{\mathrm{x},i}\}_{i=1}^{k}$ and $\{l_{\mathrm{z},i}\}_{i=1}^{k}$, a logical basis for the resulting code can be explicitly written as:

$$l'_{\mathrm{x},i} = \begin{cases} (l_{\mathrm{x},i},\, l_{\mathrm{x},i}), & i = 1,\ldots,k, \\ (0,\, l_{\mathrm{z},i-k}), & i = k+1,\ldots,2k, \end{cases} \qquad l'_{\mathrm{z},i} = \begin{cases} (l_{\mathrm{z},i},\, 0), & i = 1,\ldots,k, \\ (l_{\mathrm{x},i-k},\, l_{\mathrm{x},i-k}), & i = k+1,\ldots,2k. \end{cases} \tag{G14}$$

□

As expressed in Eq. (G11), the distance $d'$ of the resulting code depends on the stabilizer structure of the primal code—specifically, on the overlaps between logical operators of one Pauli type and stabilizers and logical operators of the opposite type. Consequently, it is not possible to make (meaningful) guarantees on $d'$ knowing only the $[\![n, k, (d_{\mathrm{x}}, d_{\mathrm{z}})]\!]$ parameters of the primal code in general. We make further comments on particular cases below:

- That $d_{\mathrm{z}}$ appears as a term in Eq. (G11) implies that one must pick a primal code with $d_{\mathrm{x}} < d_{\mathrm{z}}$ for the construction to be possibly productive in distance. In the opposite case, where $d_{\mathrm{x}} \geq d_{\mathrm{z}}$, we have $d' \leq d_{\mathrm{z}} = d$ and the resulting code has distance at most that of the primal.

- As a concrete example, consider the $[\![8, 3, (d_{\mathrm{x}} = 2, d_{\mathrm{z}} = 4)]\!]$ hypercube code as the primal. Its $X$-type stabilizers are supported on the faces of a cube, while the single $Z$-type stabilizer acts on all qubits. One finds from the geometrical structure that $|L_{\mathrm{x}} \cap g_{\mathrm{z}}| \leq 2$, and the resulting distance is thereby $d' = \min(d_{\mathrm{z}}, 2d_{\mathrm{x}}) = 4$. The resulting code is the $[\![16, 6, 4]\!]$ tesseract code.

- A second illustrative example arises from the $[\![7, 3, (d_{\mathrm{x}} = 2, d_{\mathrm{z}} = 3)]\!]$ punctured hypercube code as the primal. Here, every $g_{\mathrm{z}}$ has weight at least three, and the stabilizer group contains only weight-four nontrivial operators. Consequently,

$$2|l_{\mathrm{x}}| + |g_{\mathrm{z}}| - 2|l_{\mathrm{x}} \cap g_{\mathrm{z}}| \geq |g_{\mathrm{z}}| + 2(|l_{\mathrm{x}}| - |l_{\mathrm{x}} \cap g_{\mathrm{z}}|) \geq |g_{\mathrm{z}}| \geq 3, \tag{G15}$$

  with equality achieved when $l_{\mathrm{x}}$ lies entirely within $g_{\mathrm{z}}$ and $g_{\mathrm{z}}$ is a weight-three logical $\overline{Z}$ operator. As $d_{\mathrm{z}} = 3$, the construction produces a $[\![14, 6, (d_{\mathrm{x}} = 3, d_{\mathrm{z}} = 3)]\!]$ code.

Individually addressable permutation $\overline{\mathrm{CNOT}}$ gates on the primal code implies pairwise products of $\overline{\mathrm{CNOT}}$ gates implemented by qubit permutations on the resulting code. Specifically, let $\pi$ denote a qubit permutation implementing $\overline{\mathrm{CNOT}}_{ij}$ on the primal code. Then the qubit permutation $\pi \oplus \pi$—where the same permutation $\pi$ acts on the two sets of $n$ qubits arising from the primal and dual codes—implements $\overline{\mathrm{CNOT}}_{ij}\overline{\mathrm{CNOT}}_{(j+k)(i+k)}$ on the resulting code.

### *b. Two-sub-lattice CSS codes by doubling a non-CSS code*

A non-CSS phantom code can give rise to a CSS code by doubling the number of qubits, though the CSS code will not be phantom. This follows from the well-known doubling procedure:

**Theorem 9** (Non-CSS to CSS code doubling; Thm. 1 in Ref. [69])**.** *Suppose an $[\![n, k, d]\!]$ stabilizer code has parity check matrix $(A|B)$, then the CSS code with parity matrix*

$$\left(\begin{array}{cc|cc} A & B & & \\ & & B & A \end{array}\right) \tag{G16}$$

*has parameter $[\![2n, 2k, d']\!]$ where $d \leq d' \leq 2d$.*

Given a logical basis of the original non-CSS code $\{l_{\mathrm{x},i} = (u_i|v_i)\}_{i=1}^{k}$ and $\{l_{\mathrm{z},i} = (s_i|t_i)\}_{i=1}^{k}$, a logical basis for the resulting CSS code can be written as

$$l'_{\mathrm{x},i} = (u_i, v_i|0, 0), \qquad l'_{\mathrm{x},i+k} = (s_i, t_i|0, 0), \qquad l'_{\mathrm{z},i} = (0, 0|t_i, s_i), \qquad l'_{\mathrm{z},i+k} = (0, 0|v_i, u_i), \tag{G17}$$

for all $1 \leq i \leq k$.

Likewise, individually addressable permutation $\overline{\mathrm{CNOT}}$ gates on the original non-CSS code implies pairwise products of $\overline{\mathrm{CNOT}}$ gates implemented by qubit permutations on the resulting CSS code. Specifically, let $\pi$ denote a qubit permutation implementing $\overline{\mathrm{CNOT}}_{ij}$ on the original code. Then the qubit permutation $\pi \oplus \pi$—where the same permutation $\pi$ acts on first $n$ and last $n$ qubits—implements $\overline{\mathrm{CNOT}}_{ij}\overline{\mathrm{CNOT}}_{(j+k)(i+k)}$ on the resulting code.

## Appendix H: Gate search beyond phantom

### 1. Automorphism logical Clifford gates

Here we detail our procedure to exhaustively identify all automorphism logical Clifford gates on the phantom codes generated—i.e. logical gates implemented by qubit permutations and local (single-qubit) physical Clifford gates. Consider a stabilizer code on $n$ physical qubits defined by the stabilizer generator matrix $H \coloneqq [H^{(\mathrm{x})} \mid H^{(\mathrm{z})}]$ in symplectic form (see App. A 1), where each row encodes a stabilizer generator. We extend this stabilizer generator matrix to a three-block representation [22] by appending the entry-wise (modulo-two) sum $H^{(\mathrm{x})} \oplus H^{(\mathrm{z})}$, giving

$$H_{\mathcal{E}} \coloneqq \left[H^{(\mathrm{x})} \mid H^{(\mathrm{z})} \mid H^{(\mathrm{x})} \oplus H^{(\mathrm{z})}\right]. \tag{H1}$$

This form is convenient as both $H$ and $S$ physical gates act as coordinate permutations. The $H$ gate maps $X \leftrightarrow Z$ under conjugation and thus exchanges a column between $H^{(\mathrm{x})}$ and $H^{(\mathrm{z})}$, while the $S$ gate maps $X \mapsto Y$ and $Z \mapsto Z$, and therefore exchanges a column between $H^{(\mathrm{x})}$ and $H^{(\mathrm{x})} \oplus H^{(\mathrm{z})}$.

The row span $\langle H_{\mathcal{E}} \rangle$ defines a binary linear code. Its automorphism group $\mathrm{Aut}(\langle H_{\mathcal{E}} \rangle)$ consists of all column permutations that preserve this span. To ensure that these automorphisms correspond to valid physical Clifford circuits, we restrict attention to those that also lie in the automorphism group of

$$B \coloneqq [\mathbb{I} \mid \mathbb{I} \mid \mathbb{I}], \tag{H2}$$

which guarantees that each automorphism corresponds to a circuit composed only of $H$, $S$, and SWAP gates [22, Theorem 6]. The correspondence between column permutations and physical Clifford gates is straightforward:

- $(i,\, n+i)$, which affects qubit $i$ in blocks $H^{(\mathrm{x})}$ and $H^{(\mathrm{z})}$, implements $H_i$.
- $(n+i,\, 2n+i)$, which affects qubit $i$ in blocks $H^{(\mathrm{z})}$ and $H^{(\mathrm{x})} \oplus H^{(\mathrm{z})}$, implements $S_i$ up to a Pauli correction.
- $(i,j)(n+i,n+j)(2n+i,2n+j)$, which affects qubits $i$ and $j$ in all three blocks, implements $\mathrm{SWAP}_{ij}$.

Thus, by computing

$$\mathrm{Aut}(\langle H_{\mathcal{E}} \rangle) \cap \mathrm{Aut}(\langle B \rangle), \tag{H3}$$

we obtain the set of logical Clifford operators realizable as circuits built from $\{H, S, \mathrm{SWAP}\}$ that preserve the codespace. These circuits are precisely those generated by qubit permutations and local physical Clifford gates. We find these automorphism groups using the computational algebra system `MAGMA` [70].

A subtlety is that the logical gates found from $\mathrm{Aut}(\langle H_{\mathcal{E}} \rangle) \cap \mathrm{Aut}(\langle B \rangle)$ are exact only up to physical Pauli factors and global phase. Indeed, let $\overline{U}$ denote the Clifford unitary induced by a selected permutation in $\mathrm{Aut}(\langle H_{\mathcal{E}} \rangle) \cap \mathrm{Aut}(\langle B \rangle)$. Because binary symplectic data fixes a Clifford only up to Paulis and phases [71], $\overline{U}$ can send a stabilizer to a signed product of stabilizers, not necessarily positive. While stabilizer sign changes can in principle be frame-tracked in software, at least up till non-Clifford logical gates in the circuit that are sensitive to stabilizer signs, here we ensure an exact logical gate by tacking physical Pauli corrections that restores all stabilizers to be positive onto the gate implementation. These Pauli corrections are found through standard Clifford-tableau computation.

In general, the logical action associated with a physical circuit found in this procedure is dependent on the logical basis chosen on the code. On general stabilizer codes, this means that to answer the question of whether a desired logical gate $\overline{U}$ is available via qubit permutations and physical local Cliffords, one must enumerate over all possible logical bases of the code and list the automorphism gates in each—an expensive task. In the case of CSS logical bases, however, this issue does not arise as there is a notion of logical basis independence:

**Proposition 19** (CSS logical basis independence of automorphism logical gates)**.** *If a CSS phantom code admits any automorphism logical gate $\overline{O}$ in some CSS logical basis, then it admits that gate in every CSS logical basis.*

*Proof.* Changes between any pair of CSS logical bases on a code correspond to (i.e. can be effected by) a logical CNOT circuit. Suppose the code admits the automorphism logical gate $\overline{O}$ in a CSS logical basis $L_{\mathrm{x}}, L_{\mathrm{z}}$. Now consider an arbitrary CSS logical basis $L'_{\mathrm{x}}, L'_{\mathrm{z}}$, which is related to the first by a CNOT circuit $C$ such that $L'_{\mathrm{x}} = F_{\mathrm{xx}}(C) L_{\mathrm{x}}$ and $L'_{\mathrm{z}} = F_{\mathrm{zz}}(C) L_{\mathrm{z}} = F_{\mathrm{xx}}(C)^{-\mathsf{T}} L_{\mathrm{z}}$ (see App. A 1 for notation conventions). Then $\overline{O}$ can be performed in the $L'_{\mathrm{x}}, L'_{\mathrm{z}}$ logical basis as follows: (1) transform the logical basis to $L_{\mathrm{x}}, L_{\mathrm{z}}$ by performing $C^{\dagger}$, (2) perform $\overline{O}$ as known, and (3) transform the logical basis back by performing $C$. Because the code is phantom, the entire circuit $C$ can be implemented via qubit permutations. Therefore, overall, $\overline{O}$ is performed using only qubit permutations and single-qubit physical Clifford gates—an automorphism logical gate. □

Therefore, on the CSS phantom codes that we investigate, choosing an arbitrary CSS logical basis for analysis of their automorphism logical gate sets suffices. We use the standard form logical basis (see App. D 1) in our implementation. We also note that, on a phantom code, the existence of an automorphism gate $\overline{O}_{ij\ldots k}$ on some set of distinct logical qubits $\{i, j, \ldots, k\}$ immediately implies the existence of automorphism $\overline{O}_{i'j'\ldots k'}$ on all distinct $\{i', j', \ldots, k'\}$, as logical swaps between logical qubits can be implemented by qubit permutations. For example, automorphism gates $\overline{H}_i$, $\overline{S}_i$, $\overline{H}_i\overline{S}_i\overline{H}_i$ when present are always supported on all logical qubits $i$, and $\mathrm{CZ}_{ij}$, $\mathrm{CXX}_{ij} = \overline{H}_i\overline{H}_j\mathrm{CZ}_{ij}\overline{H}_i\overline{H}_j$ when present are always supported on all ordered pairs of distinct logical qubits $(i, j)$.

### 2. Logical Clifford and magic diagonal gates

#### a. *Review of phase polynomials*

We first give a short review of the formalism of phase polynomials [72, 73], which is convenient for treating diagonal unitary gates and will be used repeatedly in the following subsections. To begin, we consider single-qubit gates. The following association between phase monomials in the formal binary variable $x \in \{0, 1\}$ and single-qubit diagonal gates in the Clifford hierarchy holds:

$$\begin{aligned} x \mod 2 &\to Z, \\ x \mod 4 &\to S, \\ x \mod 8 &\to T. \end{aligned} \tag{H4}$$

A gate on the $l^{\text{th}}$ level of the Clifford hierarchy is associated with a phase monomial taken modulo $N = 2^l$. Controlled gates acting on multiple qubits can be expressed as follows:

$$\begin{aligned} 2x_1x_2 \mod 4 &\to \mathrm{CS}_{12}, \\ 2x_1x_2 \mod 8 &\to \mathrm{CS}_{12}, \\ 4x_1x_2x_3 \mod 8 &\to \mathrm{CCZ}_{123}, \\ 8x_1x_2x_3x_4 \mod 8 &\to \mathrm{CCCZ}_{1234}, \end{aligned} \tag{H5}$$

where the subscripts on the formal binary variables $x_1, x_2, \ldots$ denote the qubits on which the gates act. Assembling phase monomials into polynomials corresponds to taking products of the diagonal gates, taken modulo $N = 2^l$ where $l$ is the highest level of the Clifford hierarchy the gates in the circuit belong to:

$$\begin{aligned} x_1 + x_2 + 2x_3 \mod 4 &\to S_1S_2Z_3, \\ x_1 + 3x_2 + 4x_4 \mod 8 &\to T_1T_2^3Z_4. \end{aligned} \tag{H6}$$

Diagonal gates commute with each other. To describe diagonal circuits at the third level of the Clifford hierarchy (e.g. $T$, CS, CCZ), taking modulus $N = 8$ is sufficient; for the fourth level (e.g. $T^{1/2}$, CCCZ), taking modulus $N = 16$ is sufficient. Lastly, we remark that the binary addition (i.e. XOR) of formal variables can be turned into products and vice versa, a common transformation used when manipulating phase polynomials:

$$\begin{aligned} 2xy \mod 8 &= x + y + 7(x \oplus y) \mod 8, \\ 4x_1x_2x_3 \mod 8 &= x_1 + x_2 + 7(x_1 \oplus x_2) + x_3 + 7(x_1 \oplus x_3) + 7(x_2 \oplus x_3) + (x_1 \oplus x_2 \oplus x_3) \mod 8. \end{aligned} \tag{H7}$$

#### b. *Logical diagonal gates via products of single-qubit $Z$-basis rotations*

A computational- ($Z$-) basis logical state of an $[\![n, k, d]\!]$ CSS code can be written as

$$|\overline{\mathbf{x}}\rangle = \sum_{\mathbf{y}\in\mathbb{F}_2^{r_\mathrm{x}}} |\mathbf{x}L_\mathrm{x} \oplus \mathbf{y}H_\mathrm{x}\rangle\,, \tag{H8}$$

where $\mathbf{x}$ is the length-$k$ binary row vector labelling the state and $H_\mathrm{x} \in \mathbb{F}_2^{r_\mathrm{x}\times n}$ and $L_\mathrm{x} \in \mathbb{F}_2^{k\times n}$ are the $X$-type stabilizer generator matrix and logical matrix of the code respectively (see App. A 3 for introduction to this convention). We assume without loss of generality that $H_\mathrm{x}$ has full row rank. Then, acting on the state with a physical operator

$$\mathbf{\Gamma} = \sum_{i=1}^{n} \Gamma_i x_i \mod N, \tag{H9}$$

which is a product of single-qubit $Z$-basis rotations, yields the state

$$\mathbf{\Gamma}\,|\overline{\mathbf{x}}\rangle = \sum_{\mathbf{y}\in\mathbb{F}_2^{r_\mathrm{x}}} \omega^{\sum_{i=1}^n \Gamma_i(\mathbf{x}L_\mathrm{x}\oplus\mathbf{y}H_\mathrm{x})_i \bmod N}\,|\mathbf{x}L_\mathrm{x}\oplus\mathbf{y}H_\mathrm{x}\rangle\,, \tag{H10}$$

where $\omega = e^{2i\pi/N}$ and as before $N = 2^l$ for probing the $l^{\text{th}}$ level of the Clifford hierarchy. Defining

$$\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{v}) \coloneqq \sum_{i=1}^n \Gamma_i v_i \mod N, \qquad \mathbf{v}\in\mathbb{F}_2^n, \tag{H11}$$

we write more succinctly

$$\mathbf{\Gamma}\,|\overline{\mathbf{x}}\rangle = \sum_{\mathbf{y}\in\mathbb{F}_2^{r_\mathrm{x}}} \omega^{\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{x}L_\mathrm{x}\oplus\mathbf{y}H_\mathrm{x})}\,|\mathbf{x}L_\mathrm{x}\oplus\mathbf{y}H_\mathrm{x}\rangle\,. \tag{H12}$$

For $\mathbf{\Gamma}$ to implement a valid logical diagonal gate, each logical state $|\overline{\mathbf{x}}\rangle$ should merely pick up a phase, which demands that all phase factors in the sum above coincide. Therefore, we require that $\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{x}L_\mathrm{x}\oplus\mathbf{y}H_\mathrm{x})$ is invariant for all $\mathbf{y}\in\mathbb{F}_2^{r_\mathrm{x}}$. We now follow the treatment described in Ref. [73, Appendix D]. To make use of this invariance constraint, we first rewrite $\mathbf{x}L_\mathrm{x}\oplus\mathbf{y}H_\mathrm{x}$ as

$$\mathbf{x}L_\mathrm{x}\oplus\mathbf{y}H_\mathrm{x} = x_1\mathbf{g}^1\oplus\cdots\oplus x_k\mathbf{g}^k\oplus y_1\mathbf{g}^{k+1}\oplus\cdots\oplus y_{m-k}\mathbf{g}^m \equiv \bigoplus_{j=1}^m z_j\mathbf{g}^j, \tag{H13}$$

where $\mathbf{g}^i$ are the rows of the stacked matrix $G = \binom{L_\mathrm{x}}{H_\mathrm{x}}$, and $\mathbf{z}\equiv(x_1,x_2,...,x_k,y_1,y_2,...,y_{r_\mathrm{x}})$ is a length-$m$ binary vector, $m\coloneqq k+r_\mathrm{x}$. Then

$$\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{x}L_\mathrm{x}\oplus\mathbf{y}H_\mathrm{x}) = \sum_{i=1}^n \Gamma_i\left(\bigoplus_{j=1}^m z_j g_i^j\right) \mod N. \tag{H14}$$

Now we make use of the property [73, Eq. D7]

$$\bigoplus_j a_j = \sum_j a_j - 2\sum_{j<l} a_j a_l + 4\sum_{j<l<r} a_j a_l a_r - 8\sum\cdots, \tag{H15}$$

for binary variables $a_i$ (a proof can be found in Ref. [40]). This allows us to expand the $\bigoplus$ in Eq. (H14), after which terms with coefficients that are multiples of $N$ can be discarded as they are congruent to 0 mod $N$. For example, to explore the Clifford hierarchy up to the third level ($N=8$), only the first three terms of Eq. (H15) need to be retained,

$$\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{x}L_\mathrm{x}\oplus\mathbf{y}H_\mathrm{x}) = \sum_{i=1}^n \Gamma_i\left[\sum_{j=1}^m z_j\mathbf{g}_i^j - 2\sum_{1\le j<l\le m} z_j z_l \mathbf{g}_i^j\mathbf{g}_i^l + 4\sum_{1\le j<l<r\le m} z_j z_l z_r \mathbf{g}_i^j\mathbf{g}_i^l\mathbf{g}_i^r\right] \mod 8 \tag{H16}$$

and employing the definition of $\mathrm{wt}_{\mathbf{\Gamma}}(\cdot)$, we can write this as

$$\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{x}L_\mathrm{x}\oplus\mathbf{y}H_\mathrm{x}) = \sum_{i=1}^n \mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{g}^j) z_j - \sum_{1\le j<l\le m} 2\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{g}^j\wedge\mathbf{g}^l) z_j z_l + \sum_{1\le j<l<r\le m} 4\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{g}^j\wedge\mathbf{g}^l\wedge\mathbf{g}^r) z_j z_l z_r \mod 8, \tag{H17}$$

where the symbol $\wedge$ denotes element-wise product of binary vectors, $(\mathbf{u}\wedge\mathbf{v})_i = u_i v_i$. Since we require this expression to be invariant with respect to $\mathbf{y}$, the rows of $G$ must satisfy:

- $\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{g}^j)\equiv 0 \mod 8,\ \forall j\in\{k+1,...,m\}$;
- $2\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{g}^j\wedge\mathbf{g}^l)\equiv 0 \mod 8,\ \forall j,l\in\{1,...,m\}$, with $j\neq l$ and not both in $\{1,...,k\}$;
- $4\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{g}^j\wedge\mathbf{g}^l\wedge\mathbf{g}^r)\equiv 0 \mod 8,\ \forall j,l,r\in\{1,...,m\}$, with $j\neq l\neq r\neq j$ and not all three in $\{1,...,k\}$.

Finding solutions for $\mathbf{\Gamma}$ that satisfy this condition, and which therefore implement valid logical diagonal gates, can be reduced to finding the kernel of a matrix $M$ whose entries in $\mathbb{Z}_8$ comprising the rows:

- $\mathbf{g}^j$ for $j \in \{k+1, k+2, ..., m\}$;
- $2(\mathbf{g}^j \wedge \mathbf{g}^l)$ for $j, l \in \{1, ..., m\}$, $j \neq l$ not both in $\{1, ..., k\}$;
- $4(\mathbf{g}^j \wedge \mathbf{g}^l \wedge \mathbf{g}^r)$ for $j, l, r \in \{1, ..., m\}$, $j \neq l \neq r \neq j$ not all three in $\{1, ..., k\}$.

The generators of the kernel of $M$ (modulo 8) describe all possible ways of acting with a series of $T$ gates and their powers on every qubit such that the codespace of the code is preserved. Lastly, the logical action of any valid combination of $T$ gates amounts to a phase factor $\exp[i\pi f_T(x_1, x_2, ..., x_k)/4]$ on the logical states—in general $\exp[2i\pi f(\mathbf{x})/N]$ for an arbitrary level of the Clifford hierarchy—which can be recovered by evaluating the polynomial

$$f_T(\mathbf{x}) = \sum_{j=1}^{k} \mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{g}^j) x_j + \sum_{1 \leq j < l \leq k} 2\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{g}^j \wedge \mathbf{g}^l) x_j x_l + \sum_{1 \leq j < l < r \leq k} 4\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{g}^j \wedge \mathbf{g}^l \wedge \mathbf{g}^r) x_j x_l x_r \mod 8. \tag{H18}$$

This method can be used to exhaustively identify logical diagonal gates implemented by products of single-qubit $Z$-basis rotations at any level of the Clifford hierarchy. For example, at the second level of the Clifford hierarchy where the physical single-qubit rotations are powers of $S$, the matrix $M$ has entries in $\mathbb{Z}_4$ and comprises the rows:

- $\mathbf{g}^j$ for $j \in \{k+1, k+2, ..., m\}$;
- $2(\mathbf{g}^j \wedge \mathbf{g}^l)$ for $j, l \in \{1, ..., m\}$, $j \neq l$ not both in $\{1, ..., k\}$;

and the overall phase on logical states is now given by $\exp[i\pi f_S(x_1, x_2, ..., x_k)/2]$ where

$$f_S(\mathbf{x}) = \sum_{j=1}^{k} \mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{g}^j) x_j + \sum_{1 \leq j < l \leq k} 2\mathrm{wt}_{\mathbf{\Gamma}}(\mathbf{g}^j \wedge \mathbf{g}^l) x_j x_l \mod 4. \tag{H19}$$

### *c. Logical diagonal fold gates*

The above method of finding logical diagonal gates implemented by products of single-qubit $Z$-basis rotations (i.e. transversally) can be combined with the embedded code technique [22, 41], to more generally find logical diagonal gates implemented by products of physical single- and multi-qubit diagonal gates. In particular, this is useful for identifying gates that are implemented with physical $S$ and CZ in depth one. This type of logical gates is termed "fold" gates in the literature, and is distinguished from transversal gates as they involve physical multi-qubit operations within the codeblock and is therefore not a priori guaranteed to be distance-preserving.

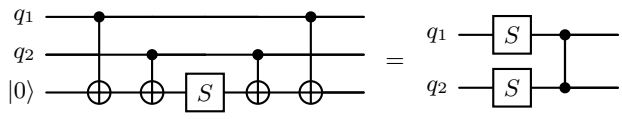



Figure 17. **Circuit identity underlying code embedding for CZ gates.** Here $\{q_1, q_2\}$ is an arbitrary pair of data qubits on the original code, and the third qubit initialized in $|0\rangle$ is an ancillary qubit that is introduced. Coupling the ancillary qubit to $\{q_1, q_2\}$ and performing an $S$ gate and a reset ($R$) on the ancilla is equivalent to performing $S$ and a CZ gate on $\{q_1, q_2\}$. This follows from the phase polynomial identity $q_1 \oplus q_2 = q_1 + q_2 + 2q_1q_2 \mod 4$.

To introduce the embedded code approach, we establish the circuit identity in Fig. 17. The idea is that by suitably introducing an ancillary qubit for every pair of data qubits on the original code, single-qubit $S$ gates on the ancillary qubit can be made equivalent to $S$ and CZ gates on the data qubits. These ancillary qubits are entangled with data qubits on the original code via CNOTs, effectively defining a larger "embedded" code—these CNOT operations extend the encoding circuit of the original code to produce the encoding circuit of the embedded code. The stabilizers of the embedded code are accordingly fixed by the stabilizers of the original code and the $Z$ stabilizers of the ancillary qubits, propagated through the CNOT operations. To be concrete, we outline explicitly the steps to construct the embedded code:

1. Add an ancillary qubit for each pair of data qubits $\{q_i, q_j\}$ on the original code.

2. Propagate each stabilizer of the original code through the CNOT gates with the ancilla qubits.

3. For each ancillary qubit $x$ associated with data qubits $\{q_i, q_j\}$ of the original code, add a stabilizer $Z_x Z_i Z_j$.

This procedure produces an $[\![n + \binom{n}{2}, k, d']\!]$ embedded code for an original $[\![n, k, d]\!]$ code. A logical basis on the embedded code can likewise be found by propagating a logical basis on the original code through the CNOT operations. Fold-diagonal logical Clifford gates on the original code can then be found by identifying diagonal logical Clifford gates on the embedded code implemented by products of single-qubit $Z$-basis rotations, through the method described in App. H 2 b, and mapping the single-qubit operations on ancillary qubits where present back onto data qubits of the original code via the circuit identity of Fig. 17.

This Z-basis rotations on embedded code approach is proposed in [41], and gives a fold-type logical gate [36, 39] when restricted to a depth-one physical implementation. A later work [22] also allows one to do permutations on the embedded code whose action on the original code translates to CNOT gates. However, we only use the former approach because a logical phase gate with some $S$ action is enough to enable the full Clifford group for phantom codes, see Fig. 13. The advantage of restricting to depth-one physical implementation is that the fault-distance of this gadget is at least half of the code distance (since two-qubit gates are allowed but with disjoint support).

However, although any element in the kernel of $M$ (see App. H 2 b) applied to the embedded code gives a valid diagonal logical operation, their physical implementation may not be depth-one when translated back to the original code. Due to this enormous search space, it is almost intractable to identify all the fold diagonal gates; it is also inefficient to even find one for larger codes (unless one constrains which pairs to add the gadget). Therefore, only partial search results for those $n \lesssim 20$ phantom codes are shown in Table I. The fold diagonal gates for the larger phantom codes are algebraically constructed (or unknown), see App. E 3 and App. F 3 b.

## Appendix I: Numerical benchmarking of logical performance

### 1. Noise model

We employ a circuit-level noise model with relative error rates calibrated to recent neutral-atom array experiments [3], detailed in Table VIII.

| Error mechanism | Model | Relative rates |
|---|---|---|
| Idle error | 1-qubit depolarizing | $p/300$ |
| 1-qubit gate error | 1-qubit depolarizing | $p/15$ |
| 2-qubit gate error | 2-qubit depolarizing | $p$ |
| Measurement error | Classical bit-flip | $5p/3$ |
| Reset error | Qubit bit-/phase-flip | $p$ |

Table VIII. **Noise model used in benchmarking simulations.** Gate errors are applied after the physical gates, measurement errors are applied on the classical measurement outcomes, and reset errors are applied on physical qubits after reset and initialization.

Depolarizing error channels of strength $p$ entail a probability $1 - p$ of leaving the input state unperturbed, and bit- and phase-flip error channels of strength $p$ impose $X$ and $Z$ errors, respectively, with probability $p$.

We take $p = 3 \times 10^{-3}$ for error rates on current-generation hardware based on recent experiments (Ref. [3], and e.g. Refs.[46–48]), and $p = 1 \times 10^{-3}$ and $p = 5 \times 10^{-4}$ for near-term (within the next 1–2 years) and future error rates, respectively, informed by ongoing and projected progress on neutral-atom platforms. Many alternative effective and microscopic error models are possible [74, 75]; our choice here is a pragmatic choice given the clear compatibility of neutral atoms with phantom codes.

### 2. Surface code implementation details

#### a. *Circuit structure and matching-based correlated decoding*

We employ a version of the recent matching-based correlated decoder [44] for the surface code, which follows from a line of work developing correlated decoding and algorithmic fault tolerance for logical circuits containing transversal

logical gates [42, 43]. Here, we use the term "correlated decoding" to mean decoding techniques that take into account error propagation through the physical circuit underlying the computation. For example, through a transversal $\overline{\text{CNOT}}$ gate, an $X_i$ error on the $i^{\text{th}}$ data qubit of the control codeblock would propagate into $X_i$ errors on *both* the control and target codeblocks. Correlated decoding enables drastic improvements in logical performance over simpler propagation-blind independent decoding of codeblocks (e.g. a $\sim 2\times$ enhancement in threshold of a $\overline{\text{CNOT}}$ gate [42]). To suitably discuss decoding, we use the language of detectors [64, 76], which is a space-time generalization of stabilizers and refer to products of Pauli operators that detect Pauli errors in a region of a circuit.

For our purpose, the relevant advance of Ref. [44] is a framework to perform correlated decoding on surface codes through minimum-weight perfect matching (MWPM), which is fast in practice and highly accurate, in place of slower integer programming or less accurate clustering decoders used in earlier works [42, 43]. While surface codes are matchable in memory experiments with code-capacity or circuit-level noise, where decoding can be performed through standard MWPM, the same is not true for logical circuits with transversal gates performed between codeblocks. The difficulty is that logical gates between codeblocks in general introduce hyperedges in the decoding graph, where a single error mechanism (e.g. a measurement error) flips more than two detectors (i.e. checks). The decoding hypergraph, unlike simple graphs, cannot be treated efficiently with standard MWPM.

The method proposed in Ref. [44] is to split the decoding hypergraph into multiple graphs, each deriving from the lightcone of a reliable logical Pauli product on the logical circuit, such that each decoding subproblem can be treated via MWPM. The overall decoding predictions of logical observable (i.e. $\overline{X}_1$, $\overline{X}_2$, $\overline{Z}_1\overline{Z}_2$, etc.) flips are then assembled from the outcomes of these subproblems. There are different ways of performing this decomposition—for example, one can start from the logical observables measured at the end of the circuit and examine their backward lightcones, or the logical stabilizers defining the initial logical state and examine their forward lightcones [44, Sec. 2.2 and App. B]. We employ the latter strategy. As is consistent with Ref. [44], we adopt the convention that a single syndrome extraction (SE) round is performed on the two codeblocks involved in every transversal $\overline{\text{CNOT}}$ before and after the gate. An overview of the decoding procedure is as follows:

1. On a logical circuit over $K$ logical qubits on $K$ codeblocks of the surface code, we denote by $\mathcal{S}^0 = \{S_i^0\}_{i=1}^K$ a basis of logical Pauli stabilizers of the initial logical state. For example, the initial state $\left|\overline{+}\overline{0}\overline{0}\right\rangle$ is stabilized by $\{\overline{X}_1, \overline{Z}_2, \overline{Z}_3\}$, where the subscripts label the codeblocks.

2. The SE rounds on the logical circuit can be organized into layers. The $t = 0$ layer is the first round of SE, which initializes the surface code codeblocks. For example, to prepare a codeblock in the $|\overline{0}\rangle$ (resp. $|\overline{+}\rangle$) state, the data qubits are initialized in $|0\rangle^{\otimes n}$ (resp. $|+\rangle^{\otimes n}$) and the first SE round performed. Subsequent $t \geq 1$ SE rounds are performed during the logical computation. The logical circuit terminates with transversal measurement of all codeblocks, which readouts the logical Pauli observables of the codes and also serves as the last SE round.

3. The decoding hypergraph comprises all error mechanisms that can occur on the circuit as defined by the noise model (e.g. depolarizing noise, measurement and reset errors) as hyperedges and detectors as vertices. We breakdown the detectors into two types:

   - *Initialization.* The $t = 0$ SE rounds themselves present as detectors. On a codeblock initialized in $|\overline{0}\rangle$ (resp. $|\overline{+}\rangle$), all $Z$-type (resp. $X$-type) stabilizers measured at $t = 0$ are detectors.
   - *Computation.* For all $t \geq 1$, detectors are declared by propagating each stabilizer $s_i^t$ of each codeblock measured in the SE round backward through the previous logical gate, which results in a product of stabilizers $\{s_j^{t-1}\}$ generically on multiple codeblocks measured in the previous SE round; the product of the set of measured stabilizers $\{s_j^{t-1}\} + s_i^t$ is declared as a detector.

   Error mechanisms (hyperedges) are incident on detectors (vertices) that they flip. The logical gates present in our context are mostly transversal $\overline{\text{CNOT}}$ gates between codeblocks.

4. We produce $K$ decoding graphs from the hypergraph. For each $S_l^0 \in \mathcal{S}^0$, we filter the decoding hypergraph into a corresponding graph by keeping only detectors in the forward lightcone of $S_l^0$. Specifically, the initialization detectors on codeblocks in $\text{supp}(S_l^0)$ are included. Moreover, propagating $S_l^0$ forward through logical gates to each SE layer $t \geq 1$ to obtain $S_l^t = \mu_{i_1}\mu_{i_2}\cdots$ where $\mu_{i_\ell} \in \{X, Z\}$, all detectors of the form $\{s_j^{t-1}\} + s_{i_\ell}^t$ where $i_\ell \in \text{supp}(S_l^t)$ and $s_{i_\ell}^t$ is the same Pauli type as $\mu_{i_\ell}$ are included. All other detectors are deleted from the hypergraph. This reduces all hyperedges into edges [44].

5. Each simulated noisy shot of the circuit produces a vector of binary measurement outcomes and thereby binary detector outcomes (whether or not each is flipped). Each of the $K$ decoding graphs are decoded through standard MWPM with this same vector of detector outcomes. The decoding result of the decoding graph built from the

forward lightcone of $S_l^0 \in \mathcal{S}$ reports on whether the logical Pauli product $S_l^{\rm f}$ is flipped at the end of the circuit, where $S_l^{\rm f}$ is $S_l^0$ forward-propagated through the entire logical circuit.

Note that $\mathcal{S}^{\rm f} = \{S_i^{\rm f}\}_{i=1}^K$ is a complete basis of logical stabilizers for the terminal logical state. Therefore, whether or not any deterministic Pauli logical observable (e.g. $\overline{X}_1\overline{Z}_2$) is flipped at the end of the circuit can be predicted from the decoding outcomes of the $K$ decoding graphs.

We use the sparse blossom MWPM algorithm in `pymatching` [77] to decode the decoding graphs.

### 3. Phantom code implementation details

#### a. *Fault-tolerant state preparation protocol*

Our state preparation protocols for the phantom $[\![64, 4, 8]\!]$ code builds on Ref. [19], which is originally proposed for the Golay code in Ref. [78]. The main idea is to employ a four-to-one logical state certification protocol for fault tolerance, which can be understood as a concatenated two-to-one (classical) state certification circuit with the two levels certifying the $X$ and $Z$ bases. There are two differences to Ref. [19]. First, we abstract away AOD movements as considered in Ref. [19] as we assume qubit permutations are to be compiled away as far as possible. Second, as our present context concerns $k > 1$ codes, we can take advantage of state automorphisms—i.e. qubit permutations that preserve the logical $|\overline{0}^k\rangle$ or $|\overline{+}^k\rangle$ state—which are larger than the code automorphism group.

We illustrate an example for the $[\![64, 4, 8]\!]$ phantom qRM code defined in Theorem 5, where all extra logicals are set to $Z$-type stabilizers. This code is not the one we use for benchmarking (it has 22 $X$-type and 38 $Z$-type stabilizer generators), but it allows us to explain the concept of the $|\overline{0000}\rangle$ state automorphism more easily. Whereas we use the top left $4 \times 4$ block of $A \in \mathrm{GL}(6, \mathbb{F}_2)$ manipulating $\{x_6, x_5, x_4, x_3\}$ [see Eq. (E1)] for permutation $\overline{\mathrm{CNOT}}$s, in the $|\overline{0000}\rangle$ state, the monomials $\{x_1x_2x_3, x_1x_2x_4, x_1x_2x_5, x_1x_2x_6\}$ take definite $+1$ values in the $Z$ basis and can therefore be treated as additional $Z$-type stabilizers. Therefore, all the degree $\leq 3$ (resp. $\leq 2$) monomials are set to $Z$-type (resp. $X$-type) stabilizers for this *state* on the code. We can then use the full GA automorphism on the six variables $x_6, \ldots, x_1$. In particular, $A$ does not have to be block-diagonal; any $A \in \mathrm{GL}(6, \mathbb{F}_2)$ is a $|\overline{0000}\rangle$ state automorphism. We state without proof that $\left(\begin{smallmatrix} B & C \\ 0_{2\times 4} & D \end{smallmatrix}\right) \in \mathrm{GL}(6, \mathbb{F}_2)$, where $B \in \mathbb{F}_2^{4\times 4}$, $C \in \mathbb{F}_2^{4\times 2}$, $D \in \mathbb{F}_2^{2\times 2}$, serves as state automorphisms for $|\overline{++++}\rangle$.

For the balanced $[\![64, 4, 8]\!]$ phantom qRM code (see Definition 8) that we perform numerical benchmarking on, we search over the following state automorphisms for the $|\overline{0000}\rangle$ and $|\overline{++++}\rangle$ state, respectively ($B$ is a $4 \times 4$ matrix):

$$\begin{pmatrix} & & & & * & 0 \\ & B & & & * & 0 \\ & & & & * & 0 \\ & & & & * & 0 \\ * & * & * & * & * & 0 \\ * & * & * & * & * & * \end{pmatrix} \in \mathrm{GL}(6, \mathbb{F}_2), \qquad \begin{pmatrix} & & & & * & 0 \\ & B & & & * & 0 \\ & & & & * & 0 \\ & & & & * & 0 \\ 0 & 0 & 0 & 0 & * & 0 \\ * & * & * & * & * & * \end{pmatrix} \in \mathrm{GL}(6, \mathbb{F}_2), \tag{I1}$$

where $B \in \mathbb{F}_2^{4\times 4}$ and $*$ indicating free entries are both searched over.

For a concrete implementation of the state-preparation protocol, we choose four permutations for each to minimize the number of malignant faults. We call an order-$s$ fault (configuration) *malignant* if it results in a strict preselection acceptance (i.e. no syndromes measured on the ancillary codeblock), but yet leaves a weight $>s$ residual error on the output codeblock (here the weight of the error is the minimum allowing multiplication by any stabilizer of the code *state*). For our distance-8 code, full fault-tolerance requires that there are no order-$<4$ malignant faults, so that the logical failure rate is guaranteed to scale at least as well as $p^4$ when varying the physical error rate $p$.

We were unable to exactly satisfy this strict demand through our numerical search. Our state preparation protocol for $|\overline{0000}\rangle$ has no order-two and $<100$ order-three malignant faults, and that for $|\overline{++++}\rangle$ has one order-two and $\sim 200$ order-three malignant faults. This difference in the number of malignant faults is expected, as the state automorphism space is larger for $|\overline{0000}\rangle$ than $|\overline{++++}\rangle$. Moreover, we did not find the $+\mathbf{b}$ in Eq. (E1), where $\mathbf{b} \in \mathbb{F}_2^6$, to help with eliminating the order-two malignant faults. Nevertheless, since the number of order-two/three malignant faults is small, we were able to observe $p^4$ scaling in logical failure rate in the $p = 5 \times 10^{-4}$ to $p = 3 \times 10^{-3}$ regime in our numerical simulations.

#### b. *Interfacing fault-tolerant state-preparation with logical circuit simulations*

To ensure an accurate simulation of noise on prepared codeblocks and their subsequent propagation into the logical circuit being executed, we performed (Monte Carlo) simulations of the state-preparation protocol described above (App. I 3 a) for $|\overline{0000}\rangle$ and $|\overline{++++}\rangle$ logical states and recorded large number of samples of the residual error on the

output codeblocks under strict and relaxed preselection. Then, when running the numerical simulation of the logical circuit, we inject these errors on each (otherwise perfectly initialized) ancillary codeblock employed.

### c. *Spatiotemporal sliding-window list and most-likely-error decoding*

Steane EC (with fault-tolerant state preparation) is known to be single-shot as the Steane syndrome extraction gadget involves only transversal operations. However, there are error correlations between Steane rounds that can be exploited in decoding. This notion is similar to *correlated decoding* as introduced on surface codes (see App. I 2 a), with a structural difference being that transversal $\overline{\text{CNOT}}$ gates are present in the Steane gadgets themselves.

We give a simple example to motivate this decoding consideration. Consider two consecutive rounds of bit-flip (i.e. $X$-error-detecting) Steane EC on a single data codeblock. Before these two rounds, there is the residual $X$-type error $x \in \mathbb{F}_2^n$ on the codeblock. For simplicity, assume that only the transversal measurement in the Steane gadget is noisy (i.e. the transversal $\overline{\text{CNOT}}$s and ancillary codeblocks are perfectly noiseless), and errors $m_1, m_2 \in \mathbb{F}_2^n$ occurred on the measurements. Then we observe $x \oplus m_1 \oplus c_1$ and $x \oplus m_2 \oplus c_2$ in the two transversal measurements, where $c_1, c_2 \in \mathbb{F}_2^n$ are random codewords. A naïve decoding of the first Steane round independent of the second would return the $X$-type correction $x \oplus m_1$ to be applied to the data codeblock; however, it is clearly more optimal to avoid performing the $m_1$ correction, as $m_1$ is a measurement error on the ancillary codeblock and does not correspond to qubit errors on the data codeblock. A better strategy is therefore to decode $x \oplus m_1 \oplus c_1$ and $x \oplus m_2 \oplus c_2$ jointly, to produce only $x$ as the correction. This simple observation—to decode across two Steane rounds—yielded two-orders-of-magnitude improvement in logical failure rate in our numerical tests.

This explains the use of sliding-window decoding where each window spans two consecutive Steane EC rounds. To also take into account error propagation across transversal $\overline{\text{CNOT}}$ gates on the logical circuit (à la correlated decoding), we set each window to span the control and target codeblock of each transversal $\overline{\text{CNOT}}$ gate and their Steane EC rounds before ("leading") and after ("trailing") the gate. Each window is decoded by either list or most-likely-error (MLE) decoding:

- *MLE decoding.* We construct the circuit-level detector error model (i.e. parity check matrix) for each window, and commit to faults that are inferred to have happened before the transversal $\overline{\text{CNOT}}$. The corresponding corrections are propagated through the circuit (i.e physical Pauli-frame tracking) to update measurement results in later Steane rounds. In this way the decoding for later windows depend on results of earlier windows.

  We implement the MLE decoding per window by integer programming [5]. In short, we construct a mixed-integer linear program that maximizes the probability (more specifically log-likelihood) of the inferred physical error given the noise model, subject to the constraint that the error generates syndromes consistent with those observed. These problems are solved with the high-performance integer programming solver `Gurobi`.

  To simplify the detector error model so that decoding is faster, we chose to model the ancillary codeblocks for Steane EC as noiseless logical states (e.g. through the hypercube encoding circuit) subject to single-qubit depolarizing noise on each qubit. The strength of this noise is set to $0.8p$ for strict preselection and $2.5p$ for relaxed preselection; these scales were chosen by probing the residual error weight distributions from the state preparation protocol. Moreover, we modelled the residual error on the data codeblock prior to the leading Steane EC round as single-qubit depolarizing noise of strength $2p$; this heuristic value was not specially tuned. We emphasize that the simplified noise modelling here is only for the purpose of decoding; the circuit simulation injects errors sampled from simulations of the state-preparation protocol (see App. I 3 b).

- *List decoding.* We adapt the list decoder developed in Ref. [45] for single-round code-capacity depolarizing-noise decoding to the setting of correlated decoding across two Steane EC rounds. We illustrate the central idea using the example described previously. The decoder takes as input $y_1 = x \oplus m_1 \oplus c_1$ and $y_2 = x \oplus m_2 \oplus c_2$. Let us only focus on the error on one qubit, so that $x, m_1, m_2 \in \mathbb{F}_2$, and ignore the random codewords. Assuming that the errors are well-modelled by the binary symmetric channel, $x \sim \text{BSC}(p)$ and $m_1, m_2 \sim \text{BSC}(p)$, the correlation between $y_1 = x \oplus m_1$ and $y_2 = x \oplus m_2$ is, in fact, structurally similar to $X$ and $Z$ error correlations in depolarizing noise. This similarity allows the code-capacity depolarizing-noise list decoder to be repurposed.

  In particular, we associate depolarizing noise with a quaternary alphabet: $\mathbb{I} = (0,0)$, $X = (1,0)$, $Z = (0,1)$ and $Y = (1,1)$. In our present task of decoding across two Steane rounds, we associate $(1,0)$ with $m_1 = 1,\ m_2 = 0,\ x = 0$; $(0,1)$ with $m_1 = 0,\ m_2 = 1,\ x = 0$; and, most importantly, $(1,1)$ with $x = 1$ and $m_1 = m_2 = 0$. In the last case, it is also possible that $(1,1)$ arises from $x = 0$ and $m_1 = m_2 = 1$, but this arises from a higher-order fault process occurring with probability $p^2$ (compared to $p$) and so can be disregarded to good approximation. That is, only when the list decoder infers the fault $(1,1)$ given the inputs $y_1$ and $y_2$ (likewise interpreted in quaternary) do we attribute a correction to the residual error $x$.

This correlated list decoder takes $\mathcal{O}(Ln\log n)$ time per window, where $L$ is the list size. We set the list size heuristically to 16 for the $[\![64,4,8]\!]$ code. We used the `aff3ct` [79] toolbox in the implementation of list decoding.

In practice, we observe that the list decoder is >100× faster, though ∼half as accurate, per window than MLE decoding in our benchmarking. The difference in decoding accuracy is expected, because MLE is able to draw on qualitatively more correlation information: MLE accesses both bit-flip and phase-flip Steane EC syndromes, on both the control and target codeblocks, together for decoding. The correlated list decoder, on the other hand, separates the decoding for $X$ and $Z$ errors. For $X$ errors, it (1) decodes across the two bit-flip EC rounds (before and after the transversal $\overline{\text{CNOT}}$) of the control codeblock, (2) commits corrections to the leading EC round, (3) propagates the correction to update the trailing bit-flip syndromes of the target codeblock, and finally (4) decodes across the two bit-flip Steane EC rounds of the target codeblock; and similarly for $Z$ errors, where it decodes across the phase-flip EC rounds of the target codeblock first. In principle, with a larger alphabet (than quaternary), it is possible for the list decoder to be generalized to take these correlations—between $X$ and $Z$ error types and between control and target codeblocks) into account. We leave this possible improvement for future work.

### 4. $\overline{\text{CNOT}}$ circuit benchmark

In Sec. V B and Fig. 5a we benchmarked a $\overline{\text{CNOT}}$ circuit of varying depth on four logical qubits, hosted on either four surface code codeblocks or a single $[\![64,4,8]\!]$ phantom qRM codeblock. These circuits take the following form: (1) data codeblock(s) initialization in $|\overline{0000}\rangle$ or $|\overline{++++}\rangle$, (2) repeated $\overline{\text{CNOT}}$s in the arrangement shown in Fig. 5a, (3) transversal measurement of deterministic $\{\overline{Z}_1,\overline{Z}_2,\overline{Z}_3,\overline{Z}_4\}$ for the $|\overline{0000}\rangle$ initial state, or $\{\overline{X}_1,\overline{X}_2,\overline{X}_3,\overline{X}_4\}$ for the $|\overline{++++}\rangle$ initial state, where the subscripts denote logical qubits.

For the surface code, codeblock initializations are performed in the standard fashion with physical Pauli-frame tracking (i.e. à la algorithmic fault tolerance)—see App. I 2 a—and all $\overline{\text{CNOT}}$s are transversal between codeblocks. For the phantom code, codeblock initialization is via our fault-tolerant state preparation protocol (see App. I 3 a), and all $\overline{\text{CNOT}}$s are via in-block qubit permutations that are compiled away by relabelling qubits. For both codes, transversal measurement of $Z$-type (resp. $X$-type) logical operators entail measuring all qubits of the data codeblock(s) in the $Z$ (resp. $X$) basis. The transversal measurement data also provides the last round of syndrome information.

We reported the logical failure rate per logical qubit ($\overline{\text{LFR}}$) in Fig. 5a. This $\overline{\text{LFR}}$ was computed as the average LFR across all logical qubits in both Pauli bases. On each circuit simulation shot starting from the $|\overline{0000}\rangle$ (resp. $|\overline{++++}\rangle$) initial state, the $i^{\text{th}}$ logical qubit is deemed to have suffered a logical failure when decoding of its $\overline{Z}_i$ (resp. $\overline{X}_i$) logical observable predicts an incorrect flip when compared to ground-truth (i.e. the actual logical qubit state). On $M$ shots for each Pauli basis $\mu\in\{X,Z\}$, let $m_i^{(\mu)}$ denote the number of logical failures for the $i^{\text{th}}$ logical qubit. Then

$$\overline{\text{LFR}}=\frac{1}{8}\sum_{i=1}^{4}\sum_{\mu\in\{X,Z\}}\frac{m_i^{(\mu)}}{M}. \tag{I2}$$

### 5. Logical GHZ state preparation benchmark

In Sec. V C and Fig. 5b we investigated logical GHZ state preparation on up to $K=64$ logical qubits. The logical circuit begins with the $|\overline{+00\cdots0}\rangle$ initial state and unitarily prepares the GHZ state through a log-depth $\overline{\text{CNOT}}$ circuit (which is, in fact, structurally identical to the encoding circuit for hypercube codes). All $\overline{\text{CNOT}}$s on the surface code are transversal, while the first three are in-block and all remaining are transversal for the phantom code.

Preparation of mixed-basis logical states, in particular the $|\overline{+000}\rangle$ state needed here, on $k>1$ codes is generally nontrivial. Our approach is to fault-tolerantly prepare $|\overline{0000}\rangle$ or $|\overline{++++}\rangle$ on two codeblocks and teleport one logical qubit, as shown in Fig. 13c. The single $\overline{\text{CNOT}}$ for teleportation requires two transversal $\overline{\text{CNOT}}$s to implement (see Lemma 2), and we insert a round of Steane EC in between. The window for decoding encompasses this gadget and involves two $\overline{\text{CNOT}}$s, but is only slightly larger than the usual single-$\overline{\text{CNOT}}$ window (as described in App. I 3 c). This is because there is no leading EC before the first $\overline{\text{CNOT}}$, since both codeblocks are freshly prepared; also, there is no trailing EC after the second $\overline{\text{CNOT}}$ on the $|\overline{++++}\rangle$ codeblock as it is transversally measured. We additionally preselect the $|\overline{+000}\rangle$ preparation conditioned on the last three logical qubits of the $|\overline{++++}\rangle$ codeblock being decoded correctly to $|\overline{+++}\rangle$; the rejection rate arising from this check is vastly subdominant at <0.05%.

We use direct fidelity estimation, as proposed in e.g. Ref. [80] and used in e.g. Refs. [81, 82], applied at the logical level to measure the logical fidelity of the prepared GHZ state. The logical fidelity of the prepared state $\rho$ is by

definition

$$\overline{\mathcal{F}} = \left\langle \overline{\mathrm{GHZ}}_K \right| \rho \left| \overline{\mathrm{GHZ}}_K \right\rangle, \qquad \left| \overline{\mathrm{GHZ}}_K \right\rangle = \frac{\left|\overline{0}^K\right\rangle + \left|\overline{1}^K\right\rangle}{\sqrt{2}}, \tag{I3}$$

where states and inner products between states are understood to be with respect to the logical Hilbert space. Suppose in each circuit shot, a basis $\mathcal{B}$ of logical stabilizer generators (e.g. $\{\overline{Z}_1\overline{Z}_2, \overline{Z}_2\overline{Z}_3, \ldots, \overline{Z}_{K-1}\overline{Z}_K, \overline{X}_1\overline{X}_2\cdots\overline{X}_k\}$ of the logical GHZ state is measured on $\rho$; for noiseless $\rho$, all such stabilizers should yield outcomes $+1$, but in the presence of noise some will yield $-1$ from shot to shot. Across $M$ shots of the circuit, let the number of $-1$ logical stabilizer observations be $m$. Then the logical infidelity can be shown to be

$$1 - \overline{\mathcal{F}} = \frac{m}{KM}. \tag{I4}$$

Note that, unlike e.g. Refs. [80, 82], we measure a complete basis $\mathcal{B}$ of stabilizer generators of the state rather than sampling a subset of stabilizers. This removes the consideration of sampling uncertainty.

To measure $\mathcal{B}$, which involves products of logical operators across codeblocks, on the surface and phantom codes, we (1) add a noiseless round of syndrome extraction (i.e. bare-ancilla for surface and Steane for phantom code) to enable decoding and (2) perform measurements of all the logical Pauli products in $\mathcal{B}$. This structure is a subcircuit proxy—in actual applications the prepared logical GHZ state will be consumed in a larger circuit, presumably terminating in logical measurements, and QEC cycles and logical measurement data of the broader circuit will be decoded together with those of the GHZ state preparation. Algorithmic fault tolerance on surface codes does not generally work with open time-boundaries, hence necessitating the closure of our benchmark circuit with (1). We further comment that (2) is equivalent to noiselessly executing the inverse of the logical GHZ state preparation circuit followed by the unencoding circuit of the codeblocks, at which point the logical state can be read off from physical qubits. We expect a deterministic $|\overline{+00\cdots0}\rangle$ in the absence of noise.

The closure of the decoding time-boundary with (1) in principle provides the decoder with more information than would have been available in reality on a larger circuit. To prevent this boundary effect from causing a drastic underestimation of logical infidelity on the phantom code, we exclude windows across the noiseless SE round and the preceding round when performing sliding-window decoding—i.e. the noiseless SE round is decoded separately with the *uncorrelated* list decoder. However, fault-tolerant correlated decoding on the surface code requires time-boundaries closed with transversal measurements, and we cannot separate the layers. Therefore, in actuality, the surface code is given an unfair decoding advantage relative to the phantom code in our benchmark.

### 6. Trotterized quantum simulation circuit benchmark

In Sec. V D and Fig. 6 we benchmarked a Trotterized quantum simulation circuit for a many-body Hamiltonian on up to $K = 64$ logical qubits. As described in the main text, for both the surface and phantom codes, we selected rotation angles $\theta = \phi = \pi/2$ in each Trotter step so that noisy circuit simulation is tractable. In an actual quantum simulation experiment, small angles would be used so that the Trotter error is controlled (or alternatively large angles tuned away from $\pi/2$ to access classically intractable Floquet dynamics). Small-angle rotations can be performed in a resource-efficient way on both the surface and phantom codes through STAR injection—see e.g. Ref. [49] for the surface code and App. E 6 for the phantom code.

Each Trotter block implementing $\exp\left(-i\theta Z^{\otimes 8}\right)$ comprises a forward and inverse $\overline{\mathrm{CNOT}}$ ladder on eight logical qubits, sandwiching a $\overline{Z}$ rotation. Similar to logical GHZ state preparation, these $\overline{\mathrm{CNOT}}$ ladders are performed in log-depth. On the surface code all $\overline{\mathrm{CNOT}}$s are transversal; on the phantom code only a single transversal $\overline{\mathrm{CNOT}}$ is needed, with all other $\overline{\mathrm{CNOT}}$s being in-block. The sliding-window decoding discussed in App. I 3 c directly applies for the phantom code.

We start from the $\left|\overline{++++0000\cdots++++0000}\right\rangle$ logical state, and transversally measure a complete basis of logical stabilizers $\{\overline{X}_1, \overline{X}_2, \overline{X}_3, \overline{X}_4, \overline{Z}_5, \overline{Z}_6, \overline{Z}_7, \overline{Z}_8, \ldots, \overline{Z}_{K-3}, \overline{Z}_{K-2}, \overline{Z}_{K-1}, \overline{Z}_K\}$ at the end of the circuit. We likewise use direct fidelity estimation (see App. I 5) to obtain the logical state infidelity.

### 7. Benchmarking results at $p = 3 \times 10^{-3}$ physical error rate

We benchmark the phantom code against surface codes matched either by spatial footprint or by code distance; the matching considerations are detailed in Sec. V, and not repeated here. An important difference here is that the preselection rate for fault-tolerant preparation of codeblocks of the phantom code is 1.3% (resp. 5.1%) for the strict (resp. relaxed) case.

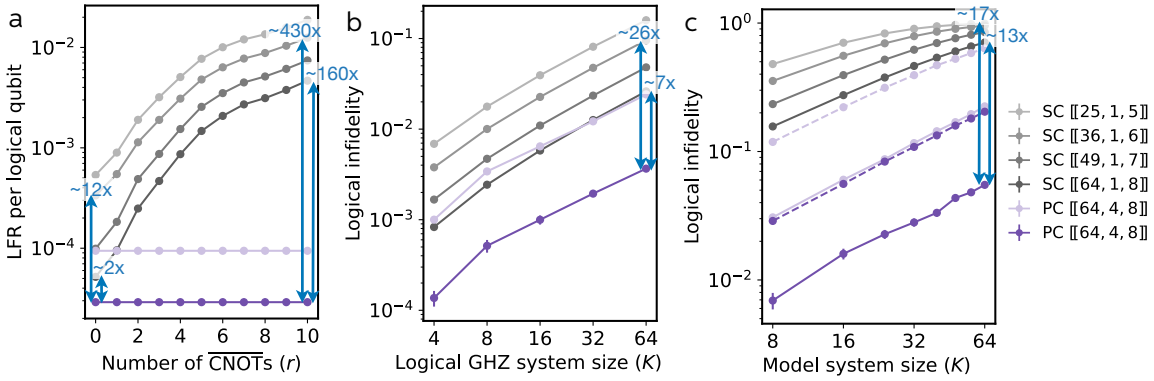



Figure 18. **Additional numerical benchmarking results of phantom and surface codes at $p = 3 \times 10^{-3}$ physical error rates.** **(a)** Repeated in-block $\overline{\text{CNOT}}$ circuits on four logical qubits hosted on a single $[\![64, 4, 8]\!]$ phantom codeblock or four surface codeblocks ($d = 5$–$8$). **(b)** Logical GHZ state preparation on $K = 4$–$64$ logical qubits, hosted on multiple $[\![64, 4, 8]\!]$ phantom or surface codeblocks. **(c)** Trotterized many-body quantum simulation with eight Trotter steps. In (a)–(c), dark and pale purple denote strict and relaxed preselection, respectively; in (c), solid and dashed lines denote sliding-window MLE and correlated list decoding, respectively. Error bars are 98% confidence intervals.

We begin with logical state preparation alone, corresponding to the $r = 0$ limit of Fig. 18a. In this setting, the phantom code achieves a $\sim$12$\times$ lower logical failure rate than the $d = 6$ surface code with comparable footprint. Even relative to the larger $d = 8$ surface code—which requires roughly twice as many physical qubits—the phantom code retains a $\sim$2$\times$ advantage.

As in-block $\overline{\text{CNOT}}$s are added, the contrast between the codes sharpens. For the surface code, the logical failure rate grows linearly with circuit depth, reflecting the accumulation of physical operations. In contrast, the phantom code's logical failure rate remains essentially constant, since its $\overline{\text{CNOT}}$s require no physical gates. This leads to improvements of up to $\sim$430$\times$ over the $d = 6$ surface code and $\sim$160$\times$ over the $d = 8$ surface code for the deepest (depth-10) circuits studied (Fig. 18a). The logical gate time of the phantom code remains effectively zero after state preparation, while it increases linearly with circuit depth for the surface code.

We observe the same behaviour in our more structured circuits. For logical GHZ state preparation at $K = 64$, the phantom code achieves a $\sim$26$\times$ reduction in logical infidelity relative to the $d = 6$ surface code (Fig. 18b). Even when compared to the larger $d = 8$ surface code, the phantom code maintains a $\sim$7$\times$ infidelity advantage while using approximately 1.8$\times$ fewer physical qubits. Notably, this advantage persists across all values of $K$, despite the increasing fraction of interblock $\overline{\text{CNOT}}$s.

Finally, for Trotterized dynamics, spatiotemporal sliding-window MLE decoding yields a $\sim$17$\times$ (resp. $\sim$13$\times$) reduction in logical infidelity over the $d = 6$ (resp. $d = 8$) surface code (Fig. 18c). The phantom code continues to outperform the $d = 8$ surface code even under relaxed preselection. Although the faster sliding-window correlated list decoder trades decoding accuracy for speed, the phantom code still exceeds the $d = 8$ surface code under strict preselection and the $d = 7$ surface code under relaxed preselection in logical fidelity.