

# Low-overhead fault-tolerant quantum computation by gauging logical operators

Received: 18 November 2024

Dominic J. Williamson<sup>1</sup>✉ & Theodore J. Yoder<sup>2</sup>

Accepted: 17 February 2026

Published online: 02 April 2026

 Check for updates

Quantum computation must be performed in a fault-tolerant manner to be useful in practice. Recent progress has established quantum error-correcting codes with sparse connectivity requirements and constant qubit overhead suitable for quantum memory. However, existing schemes that include fault-tolerant logical measurement on such quantum memories do not always achieve low qubit overhead. Here we present a low-overhead method to implement fault-tolerant logical measurement on a quantum error-correcting code by treating the logical operator as a physical symmetry and gauging it so that it is enforced by a product of local symmetries. The gauging measurement procedure introduces a high degree of flexibility that can be exploited to achieve a qubit overhead that is linear in the weight of the operator being measured up to a polylogarithmic factor. This flexibility also allows the procedure to be adapted to arbitrary quantum codes. Our results provide a more efficient approach to performing fault-tolerant quantum computation, making it more tractable for near-term implementation.

Quantum error-correcting codes are an essential ingredient to protect quantum information in a quantum computer from errors due to coupling to an external environment. Recent progress<sup>1–5</sup> has led to the discovery of good quantum low-density parity-check (qLDPC) codes<sup>6–8</sup>, which have a constant encoding rate and relative distance. Such codes are far more efficient at protecting large amounts of quantum information<sup>9,10</sup> than standard approaches based on the surface code. An important question is whether the advantages that good qLDPC codes offer for quantum information storage come at the cost of straightforward and efficient logical quantum information processing.

A key requirement for logical quantum gates on a quantum error-correcting code is fault tolerance—they must function in the presence of errors and protect the encoded quantum information throughout a computation. Existing approaches to perform fault-tolerant logical quantum gates can be roughly divided into code preserving and code deforming.

Code-preserving gates include transversal gates<sup>11</sup>, which act via the same operator on all physical and logical qubits as well as more general locality-preserving gates<sup>12</sup>. Non-trivial code-preserving gates

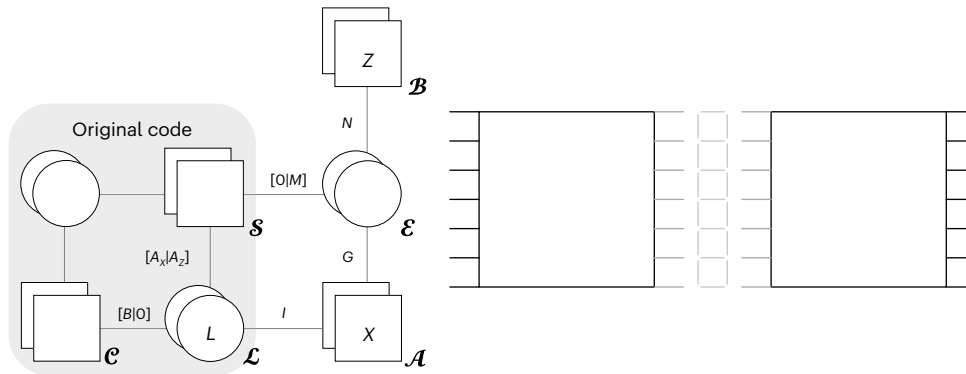
are synonymous with symmetries of an underlying code<sup>13,14</sup>. As a consequence, such gates necessitate a degree of structure to exist in the code.

Code-deforming gates are more general and allow a code to be deformed through a sequence of different codes to enact a logical gate<sup>15,16</sup>. Unlike code-preserving gates, code-deforming gates are generic as they do not necessitate the existence of non-trivial symmetries of the code beyond the logical operators themselves. In particular, code-deforming gates allow for the fault-tolerant measurement of logical operators<sup>16</sup>. This opens up the possibility of implementing measurement-only quantum computation, where Clifford gates and T-gate injections are implemented via measurement, or a Pauli-based computation<sup>17</sup> in which a quantum circuit is compiled into a sequence of generalized magic state injections<sup>18</sup>. Lattice surgery is a code-deforming approach to fault-tolerant computation for the surface code that has been studied extensively<sup>16,18–20</sup>.

There are a number of proposals for performing fault-tolerant logical gates on qLDPC codes in the literature<sup>21–27</sup>. These include code-preserving gates and code-deforming gates. The most direct generalization of surface code lattice surgery is the scheme proposed in ref. 23, and refined in ref. 26, which applies to general qLDPC codes

<sup>1</sup>IBM Quantum, IBM Almaden Research Center, San Jose, CA, USA. <sup>2</sup>IBM Quantum, IBM T.J. Watson Research Center, Yorktown Heights, NY, USA.

✉ e-mail: [dominic.williamson@sydney.edu.au](mailto:dominic.williamson@sydney.edu.au)



**Fig. 1 | Tanner graph of the gauging measurement and a surface code example.** Left: Tanner graph of the deformed code can be compactly represented by creating ordered subsets of qubits (circles) and checks (squares) of  $X$ ,  $Z$  or mixed (unlabelled) type. Each edge connecting an  $X$  or  $Z$  check set  $\mathcal{P}$  and qubit set  $\mathcal{Q}$  is labelled by a binary matrix  $H$  with  $|\mathcal{P}|$  rows and  $|\mathcal{Q}|$  columns, where  $H_{ij}=1$  if and only if the  $i$ th check in  $\mathcal{P}$  acts non-trivially on the  $j$ th qubit in  $\mathcal{Q}$ . Edges from the mixed-type check sets are instead labelled with a symplectic matrix of the form  $[H_x|H_z]$ , where  $H_x$  indicates qubits acted on by  $X$  and  $H_z$ , those by  $Z$ . The original code may not be CSS, but we assume (without loss of generality) that  $L$ —the operator being measured—is  $X$  type and its qubit support is  $\mathcal{L}$ . Set  $\mathcal{C}$  contains

checks from the original code that do not have  $Z$ -type support on  $\mathcal{L}$ , whereas set  $\mathcal{S}$  contains checks that do. Also,  $\mathcal{A}$  is the set of Gauss’s law operators  $A_v$ ,  $\mathcal{B}$  is the set of flux operators  $B_p$ ,  $\mathcal{E}$  is the set of edge qubits and we abuse the notation so that  $G$  also denotes the auxiliary graph’s incidence matrix. Matrix  $N$  specifies a cycle basis of  $G$ , and  $M$  indicates how original stabilizers are deformed by perfect matching in  $G$ . Right: applying the gauging measurement procedure to a product of  $X$ -type logicals on a pair of surface codes (dark grey edges) and choosing the graph  $G$  to be a ladder (light grey edges) results in a standard surface code lattice surgery procedure.

and makes use of an auxiliary system similar to a patch of surface code. The auxiliary system itself incurs a qubit overhead of  $\Theta(Wd)$ , where  $W$  is the weight of the logical being measured and  $d$  is the code distance. This auxiliary overhead can make fault-tolerant-measurement-based computation much more expensive than implementing a quantum memory. For instance, a good quantum code that uses  $n = \Theta(d)$  physical qubits to encode  $k = \Theta(d)$  qubits (ref. 6 provides examples) requires an auxiliary system at least  $\Omega(n^2)$  in size, substantially larger than the code itself. Even for other constant-rate  $k = \Theta(n)$  codes with polynomial distances  $d = \text{poly}(n)$ , such as hypergraph product codes<sup>2,28</sup>, measuring generic logical operators with weight  $W = \Theta(n)$ , such as in a Pauli-based computation, uses a superlinear number  $n^{\omega(1)}$  of qubits in the auxiliary system.

In this work, we introduce a general procedure for the fault-tolerant measurement of a logical operator in a stabilizer code by treating it as a physical symmetry and gauging it via measurement<sup>29,30</sup>. The gauging procedure is common in the theory of condensed-matter and high-energy physics, it enforces a global symmetry via a product of local symmetries. Our measurement scheme is based on the fact that the gauging transformation makes it possible to infer the measurement of a logical operator via a product of local stabilizers in a deformed code. Gauging has been widely used in theoretical physics to construct new models and establish relationships between known models<sup>31–33</sup>. The connection of gauging to quantum codes has also been studied<sup>34–38</sup>. Previous work focused on gauging procedures that correspond to the simultaneous initialization and read-out of all logical qubits in a code and did not consider fault tolerance. The appearance of a similar procedure in recent work on weight reduction for quantum codes hints at wider applications of gauging in quantum error correction<sup>39–44</sup>.

Here we go beyond previous work by developing a fault-tolerant gauging measurement procedure that can precisely address arbitrary individual logical operators in a large code block. We prove that the worst-case qubit overhead of the gauging measurement procedure applied to an arbitrary Pauli operator of weight  $W$  is  $O(W \log^3 W)$ , a substantial improvement over existing results in the literature. We further demonstrate that the flexibility inherent to the gauging measurement procedure leads to better performance than existing schemes for logical measurement<sup>25,26</sup> even for small instances of bivariate bicycle (BB) codes<sup>10</sup> (see the BB code example section in the Supplementary Information).

## Results

### Gauging measurement

We now present the procedure to measure a logical operator via gauging. We focus on the task of measuring a logical representative  $L$  in an  $[n, k, d]$  qLDPC stabilizer code on qubits specified by checks  $\{s_i\}$  (ref. 11). Choosing an appropriate basis for each qubit, we ensure that  $L$  is a product of Pauli- $X$  matrices without loss of generality. We view  $L$  as a symmetry of the code by identifying the code space with the ground space of the Hamiltonian  $H = -\sum_i s_i$ , and noting that  $L$  commutes with  $H$ . From this point of view, it is natural to project into an eigenspace of  $L$  by gauging it.

To gauge  $L$ , we first pick a connected graph  $G$  whose vertices  $V$  are identified with the qubits in the support of  $L$ . We sometimes refer to this graph as the auxiliary graph. Next, we introduce auxiliary qubits on the edges  $E$  of  $G$ , one per edge, initialized in the  $|0\rangle$  state. In the physics literature, these edge qubits are referred to as gauge qubits. Using these edge qubits, we introduce a set of Gauss’s law operators,  $\mathcal{A} = \{A_v\}_{v \in V}$ , where  $A_v = X_v \prod_{e \ni v} X_e$ , and crucially  $L = \prod_v A_v$ , so that we can infer the eigenvalue of  $L$  from the eigenvalues of all  $A_v$ . To obtain these eigenvalues, we measure each  $A_v$  operator. This results in a deformed code, which is a subsystem code if the number of edge qubits exceeds the number of Gauss’s law operators. However, this subsystem code is gauge fixed by the choice of the initial state for the edge qubits. Specifically, in the stabilizer group, include a set of flux operators  $\mathcal{B} = \{B_p\}_{p \in \mathcal{C}}$ , where  $B_p = \prod_{e \in p} Z_e$ , and  $p$  labels a generating set of cycles for  $G$ . The flux operators commute with the  $A_v$  operators by construction and can be obtained by deforming the initial single-qubit  $Z_e$  operators on edge qubits following standard stabilizer update rules<sup>11</sup>.

The measurement of operators in  $\mathcal{A}$  also necessarily deforms the check operators of the original code. Specifically, those check operators  $s_i$  that have  $Z$ -type support (that is, the set of qubits acted on by a  $Y$  or  $Z$ ) intersecting with the support of  $L$  do not commute with operators in  $\mathcal{A}$ . Thus, in the deformed code,  $s_i$  is replaced by  $s_i$  times some set of single-qubit  $Z_e$  operators on edge qubits. This set of edge qubits has a conceptually simple form in relation to the graph  $G$ . Because  $L$  is logical in the original code,  $s_i$  commutes with it, and therefore, the size of the  $Z$ -type support of  $s_i$  on  $L$  is even. The corresponding even-sized set of vertices in  $G$  can be paired up via a perfect matching  $\mu_i$ , that is, a set of edges whose  $Z_2$  boundary is equal to the set of vertices. Then,  $\tilde{s}_i = s_i \prod_{e \in \mu_i} Z_e$  is an appropriately deformed version of  $s_i$ , which

commutes with  $\mathcal{A}$ . Because the flux operators  $\mathcal{B}$  defined on cycles of the graph have +1 eigenvalues, different choices of perfect matching result in the same code space.

Figure 1 depicts the Tanner graph of the deformed code including the auxiliary edge qubits, the Gauss's law operators  $\mathcal{A}$ , the flux operators  $\mathcal{B}$  and the deformed checks  $\{\tilde{s}_i\}$ .

The above gauging procedure maps the original code space into the code space of a deformed code (also known as a gauged code in physics literature<sup>37</sup>). The gauged code supports fully mobile charges that are created and moved around via strings of  $Z_e$  operators, and fluxes that are created via  $X_e$  operators. These operators can be viewed as generalized higher-form symmetries, and share a mutual braiding anomaly due to the commutation relations of Pauli matrices. To implement the eigenspace projections of the gauging procedure, we rely on measuring the Gauss's law operators<sup>29,30</sup>. Although the initial measurement of each Gauss's law operator may return a random eigenvalue outcome, the product of all outcomes yields the measured eigenvalue of the logical  $L$ . We can return to the original code space by simply projecting onto the simultaneous +1 eigenspace of the  $Z_e$  operators on all the edge qubits. To implement this step, we measure  $Z_e$  on each edge qubit. Although the outcome of each measurement may be random, it is possible to find a Pauli- $X$  byproduct operator on the original qubits that can be applied to achieve the same effect as projecting onto all the +1 measurement outcome. This step is commonly referred to as ungauging<sup>34</sup>. These steps are collected in Algorithm 1.

**Algorithm 1.** Gauging measurement procedure

**Require:** a Pauli stabilizer code specified by checks  $\{s_i\}$  initialized in the physical code state  $|\psi\rangle$ .

A Pauli logical operator representative  $L$ .

A connected graph  $G = (V, E)$  with an arbitrarily chosen vertex  $v_0$  and an isomorphism that identifies the qubits in the support of  $L$  with the vertices of  $G$ .

**Ensure:** the result  $\sigma = \pm 1$  of measuring  $L$  and the post-measurement code state  $|\Psi\rangle = \frac{1}{2}(\mathbb{1} + \sigma L)|\psi\rangle$ .

$\sigma \leftarrow 1$

$|\Psi\rangle \leftarrow |\psi\rangle$

$\{\omega_e\}_{e \in E} \leftarrow \{1\}_{e \in E}$

$\gamma \leftarrow \emptyset$

**for each** edge  $e$  in  $G$  **do**

$|\Psi\rangle \leftarrow |\Psi\rangle \otimes |0\rangle_e$       $\triangleright$  initialize a qubit on  $e$

**end for**

**for each** vertex  $v$  in  $G$  **do**

Measure  $A_v$  on  $|\Psi\rangle$       $\triangleright A_v = X_v \prod_{e \ni v} X_e$

$\varepsilon \leftarrow$  measurement result      $\triangleright$  measurement result is  $\pm 1$

$|\Psi\rangle \leftarrow \frac{1}{2}(\mathbb{1} + \varepsilon A_v)|\Psi\rangle$       $\triangleright$  post-measurement state

$\sigma \leftarrow \varepsilon \sigma$

**end for**

**for each** edge  $e$  in  $G$  **do**

Measure  $Z_e$  on  $|\Psi\rangle$

$\omega_e \leftarrow$  measurement result      $\triangleright$  measurement result is  $\pm 1$

$|\Psi\rangle \leftarrow \frac{1}{2}(\mathbb{1} + \omega_e Z_e)|\Psi\rangle$       $\triangleright$  post-measurement state

Discard auxiliary qubit on  $e$

**end for**

**for each** vertex  $v$  in  $G$  **do**

$\gamma \leftarrow$  an arbitrary edge path from  $v_0$  to  $v$

**if**  $\prod_{e \in \gamma} \omega_e = -1$  **then**

$|\Psi\rangle \leftarrow X_\gamma |\Psi\rangle$       $\triangleright$  apply byproduct operator

**end if**

**end for**

**Theorem 1.** The gauging procedure defined in Algorithm 1 is equivalent to performing a projective measurement of  $L$ .

**Proof.** The proof is provided in the Methods.

We note that the auxiliary graph  $G$  can be chosen to have additional vertices beyond the qubits in support of a given logical operator. Conceptually, this is achieved by adding an extra qubit, initialized in the  $|+\rangle$  state, for each desired extra vertex and then gauging  $L$  multiplied by  $X$  on each extra vertex, instead of  $L$  itself. We refer to extra vertices as dummy vertices. In practice, the qubits on dummy vertices do not need to exist; as they are initialized in the  $|+\rangle$  state, their contribution to the Gauss's law operators is just the deterministic value +1.

**Graph desiderata and constructions**

Algorithm 1 is an extremely flexible recipe for logical measurement because the choice of a connected auxiliary graph  $G$  is arbitrary. However, the properties of the deformed code strongly depend on the choice of this graph. For instance, the gauging measurement procedure has a qubit overhead equal to the number of edges in the graph  $G$  (and involves an additional, proportional number of checks). Next, we list the desirable properties of  $G$  to achieve a fault-tolerant implementation of the gauging measurement with a low overhead.

**Definition 1.** (Suitable graph). Given a logical operator  $L$  in a qLDPC code, a suitable graph  $G$  for the gauging measurement procedure is defined to be a connected graph that results in a deformed code that is LDPC and has a code distance at least that of the original code.

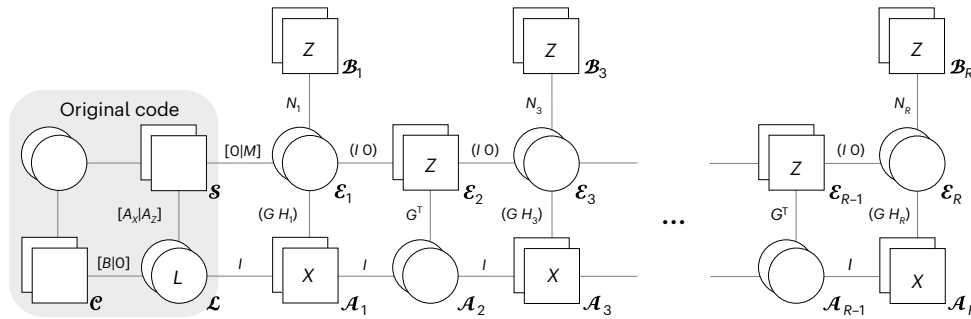
**Theorem 2.** (Graph desiderata). Consider the gauging measurement procedure for a logical operator  $L$  in a qLDPC code with a choice of connected graph  $G$ . If the following desiderata are satisfied,  $G$  is a suitable graph.

1. The graph  $G$  is sparse, that is, it has a constant degree.
2. The matchings  $\mu_i$  for all checks  $s_i$  are of a constant size, and any edge qubit is in no more than a constant number of these matchings.
3. There is a generating set of constant-weight cycles for  $G$  such that any edge is in a constant number of cycles from the generating set.
4. The graph  $G$  is sufficiently expanding in the sense that its Cheeger constant  $h(G)$  is at least 1.

**Proof.** The first three desiderata are shown to hold if and only if the deformed code is LDPC by explicitly defining a generating set of checks for the deformed code (Lemma 1 in the Methods), and inspecting that the generating set for the LDPC property. Lemma 2 in the Methods establishes that the expansion property in desideratum 4 implies a sufficient deformed code distance.

**Theorem 3.** (Construction of a suitable  $G$ ). Consider an arbitrary-weight- $W$  logical  $L$  in a qLDPC code; a suitable graph  $G$  can be constructed with qubit overhead  $O(W \log^3 W)$ .

**Proof.** We now outline a construction that produces a graph  $G$  with qubit overhead  $O(W \log^3 W)$  that satisfies the desiderata in Theorem 2. The graph starts with  $W$  vertices associated with the qubits in the support of  $L$ . First, to satisfy desideratum 2, for each check  $s_i$ , we pair up the qubits in the intersection of the  $Z$ -type support of  $s_i$  with the support of  $L$ , and add an edge to  $G$  for each pair of matched qubits. This step results in a constant-degree graph due to the LDPC property of the input code. Second, to satisfy desideratum 4, we add edges to  $G$  until  $h(G) \geq 1$ . This step can be performed by adding edges to  $G$  at random and preserving the constant degree, or by taking an existing constant-degree expander graph with  $h(G) \geq 1$  and adding its edges to  $G$ . Both these methods lead to a constant-degree graph. Third, we take the Cartesian graph product of  $G$  with a path graph on  $r$  vertices. In other words, this step takes  $r$  copies of  $G$  and for all  $i = 1, 2, \dots, r - 1$ , it connects copy  $i$  to copy  $i + 1$  by adding an edge between each vertex in copy  $i$  and the corresponding vertex in copy  $i + 1$ . Fourth, we add edges to the additional layers to sparsify the cycle basis of this product graph and achieve a desired constant-weight cycle basis. This results in a final



**Fig. 2 | Tanner graph of the decongested and cellulated gauging measurement.** Tanner graph of the complete construction including decongestion and cellulation to guarantee the deformed code is LDPC (Theorem 3). We use the same notation as that in Fig. 1. Here, as a result of the Cartesian product operation in step three of Theorem 3, the  $r = (R + 1)/2$  copies of a base expanding graph  $G$  have vertices and edges  $(A_j, E_j)$  for odd  $j$ . Notation  $(B_1, B_2)$

indicates a block matrix constructed from submatrices  $B_1$  and  $B_2$ . We use block matrices in which the qubits of the left block are those on the edges of the expanding graph  $G$  and qubits of the right block are those added to cellulate cycles in step four of Theorem 3. Depending on the layer, cellulation is done to different elements of the cycle basis of  $G$ . Using  $r = O(\log^3 W)$  layers ensures the deformed code is LDPC by the decongestion lemma<sup>45</sup>.

graph that satisfies the first three desiderata by construction (Lemma 3 in the Methods). The last two steps are based on the Freedman–Hastings decongestion lemma<sup>45</sup> and cellulation, and their purpose is to satisfy desideratum 3 by creating a graph with a sparse cycle basis without compromising the other desiderata that were established in steps one and two. Desideratum 4 may not be satisfied by the larger graph created during the decongestion step; however, in Corollary 1 (Methods), we show that as long as the original graph  $G$  satisfies desideratum 4, the deformed code distance is at least that of the original code. The Freedman–Hastings decongestion lemma establishes an  $O(\log^3 W)$  upper bound on the number of layers needed to sparsify the cycles of an arbitrary constant-degree graph  $G$  with  $W$  vertices when following the third and fourth steps; ref. 46 provides a detailed review of this point. The lemma comes with an efficient algorithm to construct a sparsified basis of cycles, given a constant-degree graph. Figure 2 shows a depiction of the deformed code after these steps.

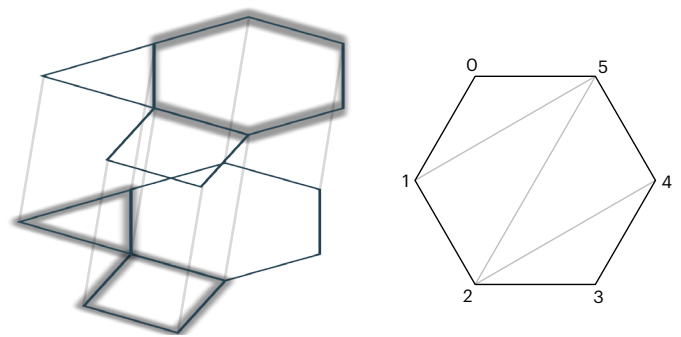
**Fault-tolerant implementation**

Even if the measurements are faulty, Algorithm 1 can be implemented in a fault-tolerant manner by measuring the stabilizer checks of the original code for  $d$  rounds, followed by measuring the checks of the deformed code for  $d$  rounds, and finally measuring the checks of the original code for a further  $d$  rounds.

**Theorem 4.** (Fault tolerance). *The fault-tolerant implementation of Algorithm 1 with a suitable graph has spacetime fault distance  $d$ .*

**Proof.** The proof follows several lemmas that are stated and proved in the ‘Fault-tolerant implementation’ section in the Methods and the spacetime code and spacetime fault-distance section in the Supplementary Information. The argument is structured as follows: the fault distance is shown to be lower bounded by the minimum of the space fault distance and the time fault distance. The time fault distance is lower bounded by the number of rounds between the start and end of the code deformation, which is chosen to be  $d$ . The space fault distance is lower bounded by the distance of the original code multiplied by  $\min(1, h(G))$ . This is because any logical in the deformed code can be cleaned such that it defines a logical of the original code. The distance reduction of the cleaning is determined by the connectivity of  $G$ .

The  $d$  rounds of quantum error correction in the original code before and after the gauging measurement are for the purposes of establishing a proof and may be overkill in practice. In a full fault-tolerant computation, the number of rounds required before and after gauging measurement depends on the surrounding operations. If the gauging measurement occurs in the middle of a large computation, it could be that even a constant number of rounds before and



**Fig. 3 | Cycle sparsification and cellulation.** Left: cycle sparsification with two layers. Thin, black edges are copies of the original graph and grey edges connect them. A complete cycle basis consists of the highlighted cycles and all length-four cycles between layers that traverse exactly two black and two grey edges. The constant  $c$  in Definition 3 is 1 after this sparsification, whereas it was 2 for the original graph, and therefore,  $R_G^1 = 2$ . Right: cellulating a weight-six cycle (black) into triangles by adding additional edges (grey).

after are sufficient to ensure fault tolerance. It is an open question to fully characterize when fault tolerance is achieved with fewer than  $d$  rounds between gauging measurements and how this choice affects the threshold depending on the surrounding operations.

**Discussion**

In this work, we have introduced a method to implement low-overhead fault-tolerant quantum computation with high-distance qLDPC codes. Our method is based on viewing a logical operator as a symmetry and gauging it via local measurements. The gauging measurement procedure allows a high degree of flexibility, making it a promising method that can be optimized for even small code instances with near-term applications. We have taken advantage of this flexibility to devise specific gauging measurements (for example, BB codes) that are more efficient than all existing measurement schemes in the literature<sup>23,26</sup>.

The gauging measurement goes beyond previous approaches to qLDPC code surgery, which was initiated in ref. 23 and developed in subsequent works<sup>25–27,47</sup>. Our key innovation is the introduction of a flexible ancilla system that is not wholly defined by the structure of the logical in the original code. This allowed us to find a fault-tolerant qLDPC code surgery procedure to measure arbitrary Pauli logicals with a worst-case scaling that is linear in the weight of the target logical, up to polylogarithmic factors. It also allowed us to find more efficient logical measurements on instances with a small block size. The flexibility of

the gauging measurement procedure can be used to recover a number of well-known existing schemes for logical measurement including surface code lattice surgery<sup>16</sup>, the measurement scheme in ref. 23 and the modified version in ref. 26. The recovering existing protocols from the gauging measurement framework section in the Supplementary Information provides more details on how these and other existing protocols are special cases of gauging measurement.

Some aspects of our approach to design high-performance gauging measurements extend previous work on qLDPC code surgery. In particular, ref. 26 points out the importance of sufficient hypergraph expansion to ensure a large fault distance. As a result of our work, we are able to guarantee appropriate expansion in the gauging measurement of an arbitrary logical operator by adding edges to the auxiliary graph. The gauging measurement also implements gauge fixing of the deformed code, as in ref. 26, by measuring the checks associated with cycles in the auxiliary graph. Importantly, the graph-based approach introduced in the gauging measurement allows us to construct a sparse cycle check basis for any code by using cellulation and decongestion<sup>41,45</sup> to ensure that the deformed code remains LDPC (Fig. 3). This goes beyond previous ad hoc approaches to gauge fixing on specific instances without a worst-case guarantee, as that in ref. 26.

Concurrent with our work, ref. 48 introduced a similar approach to qLDPC code surgery that involves gauge fixing with cellulation, and boosting expansion by adding edges to an auxiliary graph. There, the focus was on logicals of a fixed Pauli type in the Calderbank–Shor–Steane (CSS) codes, whereas our focus is on general Pauli logicals in stabilizer codes, which can be non-CSS. Furthermore, ref. 48 does not consider decongestion, and, as such, does not provide a worst-case guarantee on the scaling of their logical measurement procedure. In a follow-up work, ref. 49 builds directly on the gauging measurement framework by introducing a graph construction algorithm that is used to find an efficient procedure to perform a joint logical measurement of disjoint logical operators on one or more qLDPC code blocks. This operation is a gauging measurement with a different choice of auxiliary graph to the worst-case construction outlined above.

The progress reported in this work raises a number of directions that invite further investigation. For what families of good qLDPC codes can the gauging measurement be performed with a linear qubit overhead? Can the generalization of the gauging measurement to a hypergraph be used to measure a large number of commuting but overlapping logical operators simultaneously? Can we introduce meta-checks to the generalized gauging measurement procedure to perform a single-shot measurement with a constant time overhead? How should the fault-tolerant gauging measurement be decoded? We expect that a general-purpose decoder based on belief propagation with ordered statistics post-processing can be used; however, it should be possible to take advantage of the extra structure inherent to the procedure to find a decoder with better performance. Such a decoder could incorporate matching on the  $A_i$  syndromes similar to the approach in ref. 26.

The directions listed here can be interpreted as a list of desiderata for an ideal fault-tolerant logical measurement procedure: it should implement Pauli-based computation, be fully parallelized and have a constant relative overhead in space and time. It remains an open question whether any fault-tolerant logical measurement can satisfy the above desiderata, or how close one can get. Even the answer to the classical analogue of this question is not known.

## Online content

Any methods, additional references, Nature Portfolio reporting summaries, source data, extended data, supplementary information, acknowledgements, peer review information; details of author contributions and competing interests; and statements of data and code availability are available at <https://doi.org/10.1038/s41567-026-03220-8>.

## References

- Gottesman, D. Fault-tolerant quantum computation with constant overhead. *Quantum Inf. Comput.* **14**, 1338–1372 (2014).
- Tillich, J.-P. & Zémor, G. Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Trans. Inf. Theory* **60**, 1193–1202 (2014).
- Hastings, M. B., Haah, J. & O’Donnell, R. Fiber bundle codes: breaking the  $N^{1/2}$  poly  $\log(N)$  barrier for quantum LDPC codes. In *Proc. 53rd Annual ACM SIGACT Symposium on Theory of Computing* 1276–1288 (Association for Computing Machinery, 2021).
- Panteleev, P. & Kalachev, G. Quantum LDPC codes with almost linear minimum distance. *IEEE Trans. Inf. Theory* **68**, 213–229 (2021).
- Breuckmann, N. P. & Eberhardt, J. N. Balanced product quantum codes. *IEEE Trans. Inf. Theory* **67**, 6653–6674 (2021).
- Panteleev, P. & Kalachev, G. Asymptotically good quantum and locally testable classical LDPC codes. In *Proc. 54th Annual ACM SIGACT Symposium on Theory of Computing* 375–388 (Association for Computing Machinery, 2022).
- Leverrier, A. & Zémor, G. Quantum Tanner codes. In *Proc. 63rd Annual IEEE Symposium on Foundations of Computer Science* 872–883 (IEEE, 2022).
- Dinur, I., Hsieh, M.-H., Lin, T.-C. & Vidick, T. Good quantum LDPC codes with linear time decoders. In *Proc. 55th Annual ACM Symposium on Theory of Computing* 905–918 (Association for Computing Machinery, 2023).
- Panteleev, P. & Kalachev, G. Degenerate quantum LDPC codes with good finite length performance. *Quantum* **5**, 585 (2021).
- Bravyi, S. et al. High-threshold and low-overhead fault-tolerant quantum memory. *Nature* **627**, 778–782 (2024).
- Gottesman, D. *Stabilizer Codes and Quantum Error Correction*. PhD thesis, California Institute of Technology (1997).
- Beverland, M. E. et al. Protected gates for topological quantum field theories. *J. Math. Phys.* **57**, 022201 (2016).
- Bombin, H. & Martin-Delgado, M. A. Topological quantum distillation. *Phys. Rev. Lett.* **97**, 180501 (2006).
- Yoshida, B. Gapped boundaries, group cohomology and fault-tolerant logical gates. *Ann. Phys.* **377**, 387–413 (2017).
- Raussendorf, R. & Harrington, J. Fault-tolerant quantum computation with high threshold in two dimensions. *Phys. Rev. Lett.* **98**, 190504 (2007).
- Horsman, D., Fowler, A. G., Devitt, S. & Van Meter, R. Surface code quantum computing by lattice surgery. *New J. Phys.* **14**, 123011 (2012).
- Bravyi, S., Smith, G. & Smolin, J. A. Trading classical and quantum computational resources. *Phys. Rev. X* **6**, 021043 (2016).
- Litinski, D. A game of surface codes: large-scale quantum computing with lattice surgery. *Quantum* **3**, 128 (2019).
- Fowler, A. G. & Gidney, C. Low overhead quantum computation using lattice surgery. Preprint at <http://arxiv.org/abs/1808.06709> (2018).
- Chamberland, C. & Campbell, E. T. Universal quantum computing with twist-free and temporally encoded lattice surgery. *PRX Quantum* **3**, 010331 (2022).
- Zhu, G., Sikander, S., Portnoy, E., Cross, A. W. & Brown, B. J. Non-Clifford and parallelizable fault-tolerant logical gates on constant and almost-constant rate homological quantum low-density parity-check codes via higher symmetries. *PRX Quantum* **6**, 040361 (2025).
- Scruby, T. R., Pesah, A. & Webster, M. Quantum rainbow codes. Preprint at <http://arxiv.org/abs/2408.13130> (2024).
- Cohen, L. Z., Kim, I. H., Bartlett, S. D. & Brown, B. J. Low-overhead fault-tolerant quantum computing using long-range connectivity. *Sci. Adv.* **8**, eabn1717 (2022).

24. Cowtan, A. & Burton, S. CSS code surgery as a universal construction. *Quantum* **8**, 1344 (2024).
25. Cowtan, A. SSIP: automated surgery with quantum LDPC codes. Preprint at <https://arxiv.org/abs/2407.09423> (2024).
26. Cross, A. W., He, Z., Rall, P. J. & Yoder, T. J. Improved QLDPC surgery: logical measurements and bridging codes. Preprint at <https://arxiv.org/abs/2407.18393> (2024).
27. Zhang, G. & Li, Y. Time-efficient logical operations on quantum low-density parity check codes. *Phys. Rev. Lett.* **134**, 070602 (2025).
28. Leverrier, A., Tillich, J.-P. & Zémor, G. Quantum expander codes. In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science* 810–824 (IEEE, 2015).
29. Williamson, D. J. & Devakul, T. Type-II fractons from coupled spin chains and layers. *Phys. Rev. B* **103**, 155140 (2021).
30. Tantivasadakarn, N., Thorngren, R., Vishwanath, A. & Verresen, R. Long-range entanglement from measuring symmetry-protected topological phases. *Phys. Rev. X* **14**, 021040 (2024).
31. Kramers, H. A. & Wannier, G. H. Statistics of the two-dimensional ferromagnet. Part I. *Phys. Rev.* **60**, 252 (1941).
32. Wegner, F. J. Duality in generalized Ising models and phase transitions without local order parameters. *J. Math. Phys.* **12**, 2259–2272 (1971).
33. Kogut, J. & Susskind, L. Hamiltonian formulation of Wilson's lattice gauge theories. *Phys. Rev. D* **11**, 395 (1975).
34. Williamson, D. J. Fractal symmetries: ungauging the cubic code. *Phys. Rev. B* **94**, 155128 (2016).
35. Vijay, S., Haah, J. & Fu, L. Fracton topological order, generalized lattice gauge theory and duality. *Phys. Rev. B* **94**, 235157 (2016).
36. Kubica, A. & Yoshida, B. Ungauging quantum error-correcting codes. Preprint at <https://arxiv.org/abs/1805.01836> (2018).
37. Dolev, K. et al. Gauging the bulk: generalized gauging maps and holographic codes. *J. High Energy Phys.* **2022**, 158 (2022).
38. Rakovszky, T. & Khemani, V. The physics of (good) LDPC codes I. Gauging and dualities. Preprint at <https://arxiv.org/abs/2310.16032> (2023).
39. Bacon, D., Flammia, S. T., Harrow, A. W. & Shi, J. Sparse quantum codes from quantum circuits. In *Proc. 47th Annual ACM Symposium on Theory of Computing* 327–334 (Association for Computing Machinery, 2015).
40. Hastings, M. B. Weight reduction for quantum codes. Preprint at <https://arxiv.org/abs/1611.03790> (2016).
41. Hastings, M. B. On quantum weight reduction. Preprint at <https://arxiv.org/abs/2102.10030> (2021).
42. Wills, A., Lin, T.-C. & Hsieh, M.-H. Tradeoff constructions for quantum locally testable codes. *IEEE Trans. Inf. Theory* **71**, 426–458 (2025).
43. Sabo, E., Gunderman, L. G., Ide, B., Vasmer, M. & Dauphinais, G. Weight-reduced stabilizer codes with lower overhead. *PRX Quantum* **5**, 040302 (2024).
44. Baspin, N. & Williamson, D. J. Wire codes. Preprint at <https://arxiv.org/abs/2410.10194> (2024).
45. Freedman, M. & Hastings, M. Building manifolds from quantum codes. *Geom. Funct. Anal.* **31**, 855–894 (2021).
46. He, Z., Cowtan, A., Williamson, D. J. & Yoder, T. J. Extractors: QLDPC architectures for efficient Pauli-based computation. Preprint at <https://arxiv.org/abs/2503.10390> (2025).
47. Xu, Q. et al. Fast and parallelizable logical computation with homological product codes. *Phys. Rev. X* **15**, 021065 (2025).
48. Ide, B., Gowda, M. G., Nadkarni, P. J. & Dauphinais, G. Fault-tolerant logical measurements via homological measurement. *Phys. Rev. X* **15**, 021088 (2025).
49. Swaroop, E., Jochym-O'Connor, T. & Yoder, T. J. Universal adapters between quantum low-density parity check codes. *PRX Quantum* **7**, 010324 (2026).

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

**Open Access** This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

© The Author(s) 2026

## Methods

### Gauging measurement

Here we prove Theorem 1, showing that the gauging measurement correctly performs a logical measurement. We first introduce definitions of boundary and coboundary maps on a graph  $G$ .

**Definition 2.** ( $\mathbb{Z}_2$  boundary and coboundary maps). *In this work, we use binary vectors to indicate collections of vertices, edges and cycles of the graph  $G$ . The boundary map  $\partial$  on an edge vector is a  $\mathbb{Z}_2$ -linear map that is defined by its action on a single edge  $\partial e = v + v'$ , where  $v$  and  $v'$  are the adjacent edges of  $e$ . The coboundary map is given by the transpose of the boundary map  $\delta = \partial^T$  and satisfies  $\delta v = \sum_{e \ni v} e$ . Given a choice of a collection of cycles in  $G$ , we also define a second boundary map  $\partial_2$  whose action on a cycle is  $\partial_2 c = \sum_{e \in c} e$ . Similarly, we define a second coboundary map  $\delta_2 = \partial_2^T$ , which acts on a single edge as  $\delta_2 e = \sum_{c \ni e} c$ .*

**Remark 1.** The maps  $\partial_2, \partial$  form an exact sequence if a generating set of cycles is chosen, and similarly for  $\delta, \delta_2$ . These sequences are not short exact as  $\delta$  has a non-trivial kernel given by all vertices.

Throughout this work, we abuse a notation by identifying the binary vector associated with a set of vertices, edges or cycles with the set itself. Where this is done, the meaning should be clear from context.

**Proof.** (Proof of Theorem 1). Applying the gauging measurement to an initial state  $|\Psi\rangle$  results in

$$\prod_v \frac{1}{2} (\mathbb{1} + \varepsilon_v A_v) |\Psi\rangle |0\rangle_E \tag{1}$$

up to an overall normalization. Here  $\varepsilon_v$  is the observed result of measuring the operator  $A_v = X_v \prod_{e \ni v} X_e$  and  $|0\rangle_E = |0\rangle^{\otimes E}$ . Ungauging by measuring out the edge degrees of freedom in the  $Z$  basis and discarding them results in the state

$$\langle z_e |_E \prod_v \frac{1}{2} (\mathbb{1} + \varepsilon_v A_v) |\Psi\rangle |0\rangle_E, \tag{2}$$

up to an overall normalization. Here  $\langle z_e |_E = \otimes_e \langle z_e |_e$ , where  $z_e$  is the observed result of measuring the operator  $Z_e$ . Next, we expand the product of projection operators into a sum

$$\langle z_e |_E \frac{1}{2^V} \sum_{c \in C^0(G, \mathbb{Z}_2)} \varepsilon(c) X_V(c) X_E(\delta c) |\Psi\rangle |0\rangle_E, \tag{3}$$

where the sum is over  $\mathbb{Z}_2$ -valued 0-cochains on  $G$ :

$$\varepsilon(c) = \prod_v \varepsilon_v^{c_v}, \tag{4}$$

$$X_V(c) = \prod_v X_v^{c_v}, \tag{5}$$

$$X_E(\delta c) = \prod_e X_e^{\{\delta c\}_e}. \tag{6}$$

Since  $\langle z_e |_E X_E(\delta c) |0\rangle_E$  is zero unless  $z_e = \{\delta c\}_e$  on all edges, we can rewrite the ungauged state as

$$\begin{aligned} & \frac{1}{2^V} \sum_{c \text{ s.t. } \delta c = z} \varepsilon(c) X_V(c) |\Psi\rangle \\ &= \frac{1}{2^V} X_V(c') \sum_{c \in Z^0(G, \mathbb{Z}_2)} \varepsilon(c) X_V(c) |\Psi\rangle \end{aligned} \tag{7}$$

for a fixed  $c'$  that satisfies  $\delta c' = z$  and  $Z^0(G, \mathbb{Z}_2)$  is the group of 0-cocycles on  $G$ . For a connected graph  $G$ , there are only two elements  $c \in Z^0(G, \mathbb{Z}_2)$ , either  $c_v = 1$  for all vertices or  $c_v = 0$  for all vertices. Hence, the ungauged state is

$$X_V(c') \frac{1}{2} (\mathbb{1} + \sigma L) |\Psi\rangle, \tag{8}$$

where  $L = \prod_v X_v$  is the logical operator being measured and  $\sigma = \prod_v \varepsilon_v$  is the observed outcome of the logical measurement. Here  $X_V(c')$  is a Pauli byproduct operator determined by the observed measurement outcomes.

This byproduct operator is the same as the one defined in the final step of Algorithm 1.

### Graph desiderata and constructions

In this section, we analyse the deformed code that is created after measuring the  $A_v$  terms in the gauging measurement procedure. The auxiliary graph  $G$  contains vertices  $V$  and edges  $E$ .

**Lemma 1.** (Deformed code). *The following form a generating set of checks for the deformed code:*

- $A_v = X_v \prod_{e \ni v} X_e$  for all  $v \in V$  that are not dummy vertices.
- $A_v = \prod_{e \ni v} X_e$  for all dummy vertices  $v \in V$ .
- $B_\gamma = \prod_{e \in \gamma} Z_e$  for a generating set of cycles  $\gamma \subseteq E$ .
- $\tilde{s}_i = s_i \prod_{e \in \mu_i} Z_e$  for all checks in the input code  $s_i$  and appropriate perfect matchings  $\mu_i \subseteq E$  for each of those checks.

**Proof.** The  $A_v$  operators become checks since they are measured during the gauging process. Both  $B_\gamma$  and  $\tilde{s}_i$  operators can be thought of as arising from applying stabilizer update rules<sup>11</sup> to the initial stabilizer group generated by checks of the original code  $\{s_i\}$  and the single-qubit  $Z_e$  stabilizers on the edge qubits  $e$ . Specifically, the  $B_\gamma$  operators originate from the  $Z_e$  stabilizers because for a product of these edge stabilizers to remain a check after measuring the  $A_v = \prod_{e \ni v} X_e$  operators, it must commute with them all. This is equivalent to a product  $\prod_{e \in \gamma} Z_e$  being over edges  $\gamma$  that satisfies  $\partial \gamma = 0$ , that is,  $\gamma$  represents a graph cycle. Similarly, the  $\tilde{s}_i$  operators arise from products of  $s_i$  and  $Z_e$  stabilizers from the initialization step that commute with all  $A_v$  operators. The specific  $A_v$  operators that  $s_i$  anticommutes with are those associated with vertices in the set  $S_Z := \text{supp}_Z(s_i) \cap \text{supp}(L)$ , where  $\text{supp}_Z(s_i)$  is the  $Z$ -type support of  $s_i$ . Thus, commuting with all  $A_v$  requires a set of edges  $\mu_i$  such that  $\partial \mu_i = \text{supp}_Z(s_i) \cap \text{supp}(L)$ , or in other words,  $\mu_i$  is a perfect matching of the vertices in  $S_Z$ .

By counting the qubits and checks, it is evident that the dimension of the code space of the deformed code is only reduced by a single qubit compared with the original code, corresponding to the logical  $L$  that is measured by the gauging deformation. A total of  $|E|$  new qubits are introduced along with  $[|V|]$  new independent  $X$ -type checks and a number of new independent  $Z$ -type checks that is equal to  $C$ , the number of generating cycles of  $G$ . We then have the well-known relation  $|E| - C - [|V|] = -1$  for a connected graph. This provides an alternative method to Theorem 1 to see that no logical information is lost beyond the measurement of  $L$ .

There is a large degree of freedom when specifying a generating set of checks in the deformed code. If we fix the choice of  $A_v$  and  $s_i$  checks, then the freedom can be associated with choosing cycles and matchings  $\gamma$  and  $\mu_i$ . These choices do not affect the code space, as the  $B_\gamma$  operators for any generating set of cycles in  $G$  generate the same algebra. Furthermore, the  $\tilde{s}_j$  operator for two different choices of path  $\gamma$  are related by multiplication with  $B_\gamma$  operators. In practice, we aim to choose a set of paths  $\gamma$  and  $\mu_i$  that result in a set of checks with a small weight and that result in a small qubit degree (the number of checks a qubit is involved in). Indeed, this is the point of the first three graph desiderata in Theorem 2, which ensure the deformed code is LDPC. The necessity and sufficiency of the desiderata for that purpose should now be clear from inspecting the full set of deformed code checks in Lemma 1.

To complete the proof of Theorem 2, it remains to show that graph expansion implies the deformed code has code distance no smaller than the original code. We do that in the following lemma.

**Lemma 2.** (Space fault distance). *The distance of the deformed code satisfies  $d^* \geq \min(h(G), 1)d$ , where  $h(G)$  is the Cheeger constant of  $G$  and  $d$  is the distance of the original code.*

**Proof.** A logical operator on the deformed code can be written as  $L' = i^0 L_X^V L_Z^V L_X^E L_Z^E \bar{L}$ . Here  $\bar{L}$  captures the support of  $L'$  that does not intersect the gauged logical,  $s_X$  denotes the  $X$  support of  $L'$ ,  $L_X^V = \prod_{v \in s_X \cap V_G} X_v$  captures the intersection of the  $X$  support of  $L'$  with the gauged logical,  $L_X^E = \prod_{e \in s_X \cap E_G} X_e$  captures the  $X$  support of  $L'$  on the edges introduced by the gauging procedure, and similarly for  $Z$ .

First, we consider the  $X$ -type component of the logical operator  $L'$ . The logical operator must commute with the  $B_p$  checks by definition; hence,  $s_X^E = s_X \cap E_G$  is a 1-cocycle on the graph  $G$  as it satisfies  $\delta_2 s_X^E = 0$ . Since the  $B_p$  terms are defined on a generating set of cycles, we have that the sequence formed by  $\delta, \delta_2$  is exact (Remark 1). Hence,  $s_X^E = \delta \bar{s}_X^V$  for some set of vertices  $\bar{s}_X^V \subset V_G$ . From this, we have  $L_X^E = \bar{L}_X^V \prod_{v \in \bar{s}_X^V} A_v$ , where  $\bar{L}_X^V = \prod_{v \in \bar{s}_X^V} X_v$ . We now have another logical representative that is equivalent to  $L'$ , given by  $\bar{L} = i^0 L_X^V \bar{L}_X^V L_Z^V L_Z^E \bar{L} = L' \prod_{v \in \bar{s}_X^V} A_v$ .

Next, we consider the  $Z$ -type component of the logical operator  $L'$ . We first point out that the deformed checks  $\bar{s}_i$  are given by the original checks, potentially multiplied by some  $Z_e$  operators. Similarly, the equivalent logical operator  $\bar{L}$  is some operator on the original qubits potentially multiplied by  $Z_e$  operators.

From this, we can see that the equivalent logical operator restricted to the qubits of the original code  $\bar{L}|_V = i^0 L_X^V \bar{L}_X^V L_Z^V \bar{L}$  must be a logical operator of the original code. This is because it must commute with the deformed checks; since the additional operators on the edge qubits in the deformed checks are all of the form  $Z_e$ , they play no role in the commutation relations. From this, we see that the full equivalent logical operator  $\bar{L}$  is obtained from the restricted logical  $\bar{L}|_V$  via the gauging code deformation.

Hence, any logical in the deformed code is equivalent to a logical on the original code  $\bar{L}|_V$  potentially multiplied by some  $Z_e$  operators. The weight of any non-trivial logical on the original code, such as  $\bar{L}|_V$ , is lower bounded by the distance  $d$ . Hence, the weight of the unrestricted logical  $\bar{L}$  is also lower bounded by  $d$ . Furthermore, we can construct  $\bar{L}$  to have support on no more than half the vertex qubits  $\leq \frac{|V|}{2}$  by optionally multiplying  $L'$  with the stabilizer  $\prod_{v \in V_G} A_v$ .

We now lower bound the relative change in operator weight, induced by the equivalence under multiplication with vertex stabilizers that convert the deformed logical  $\bar{L}$  back to the logical  $L'$ , by the Cheeger constant  $h(G)$  of a single layer of graph  $G$ . In the worst case, multiplication with  $\prod_{v \in \bar{s}_X^V} A_v$  to convert  $\bar{L}$  to  $L'$  removes the support on the vertex set  $\bar{s}_X^V$  and adds support on the edge coboundary set  $\delta \bar{s}_X^V$ . Then, we apply  $|\delta \bar{s}_X^V| \geq h(G)|\bar{s}_X^V|$  to achieve the distance bound on the logical  $L'$ .

In the rest of this subsection, we discuss the cycle sparsification step in more detail and ensure it preserves the first three desiderata and the deformed code distance, as claimed in Theorem 3.

**Definition 3.** (Cycle-sparsified graph  $\bar{G}$ ). *Given a graph  $G$  and a constant  $c$ , we call the cycle degree, a cycle sparsification of  $G$  is a new graph that is built by adding edges to copies of  $G$  numbered  $1, 2, \dots, r$ , and alternatively referred to as layers (as shown in Fig. 2; also see Fig. 3). One type of additional edge connects each vertex to its layer in the subsequent layer and serves to implement the Cartesian product operation of graph  $G$  with a path graph on  $r$  vertices (Fig. 3). The other type of additional edges cellulate a cycle into triangles by connecting vertices as follows:  $\{(1, N-1), (N-1, 2), (2, N-2), (N-2, 3), \dots\}$  following an ordering of the vertices as they are visited when the cycle is traversed (Fig. 3). If the original graph had cycle basis  $\{y_i\}$ , the cycle-sparsified graph cellulates exactly one copy of each  $y_i$ , and, given a sufficient number of layers  $r$ , the number of these cellulated cycles containing any particular edge can be chosen to be at most  $c$  for any choice of  $c \geq 1$ . We use  $R_c^G$  to denote the minimal number of layers to achieve a cycle sparsification of  $G$  with constant  $c$ . Besides  $|L|$ , the number of specified vertices in the first layer; all vertices in the cycle-sparsified graph are dummy vertices.*

**Remark 2.** *The cycle-sparsified graph  $\bar{G}$  may fail to satisfy desideratum 4. This is because the cycle sparsification step does not preserve  $h(G) \geq 1$ . In particular, once there are more than four copies of  $G$  in the cycle-sparsified graph  $\bar{G}$ , the set  $V_{0,1} = V_{G_0} \cup V_{G_1}$  of all vertices on the first two copies of  $G$  satisfies  $|\delta V_{0,1}| = |V_{G_1}| = \frac{1}{2}|V_{0,1}|$ , and hence, the Cheeger constant is upper bounded by a half. Below, we prove that  $\bar{G}$  nonetheless results in a deformed code with the same distance bound as for  $G$ . The reason behind this is that sufficient expansion in the first layer of  $G$  that is attached directly to the logical operator suffices to guarantee that the distance is preserved during code deformation.*

**Corollary 1.** (Cycle-sparsified space fault distance). *The distance of the deformed code based on the cycle-sparsified graph  $\bar{G}$  (Definition 3) satisfies  $d^* \geq (h(G), 1)d$ .*

**Proof.** Following the proof of Lemma 2, we have  $L_X^E = \bar{L}_X^V \prod_{v \in \bar{s}_X^V} A_v$ , where  $\bar{L}_X^V = \prod_{v \in \bar{s}_X^V \cap V_{G_0}} X_v$ . In the definition of  $\bar{L}_X^V$ , the product of the operators is only over vertices in  $G_0$  since the vertices in the other layers are dummy vertices, which do not support qubits. We now have that  $\bar{L} = i^0 L_X^V \bar{L}_X^V L_Z^V L_Z^E \bar{L} = L' \prod_{v \in \bar{s}_X^V} A_v$  is a logical representative equivalent to  $L'$ . The restriction of the logical  $\bar{L}$  to the original qubits produces a logical in the original code; hence, the weight of  $\bar{L}$  is at least the distance of the original code.

To bound the distance of the general logical  $L'$  in the deformed code, we now focus on the change in weight caused by the term  $\prod_{v \in \bar{s}_X^V} A_v$ . The size of the vertex set  $\bar{s}_X^V \cap V_{G_0}$  can be taken as  $\leq \frac{|V_{G_0}|}{2}$ , up to multiplying  $L'$  with the measured logical, which is now the stabilizer  $\prod_{v \in G} A_v$ . The operator  $\bar{L} \prod_{v \in \bar{s}_X^V} A_v$  has  $X$  support on at least all edges in  $(\delta(\bar{s}_X^V \cap V_{G_0})) \cap E_{G_0}$ . This is the set of edges inside  $G_0$  that are in the coboundary of the relevant vertices in  $G_0$ . The size of this set satisfies  $|\delta(\bar{s}_X^V \cap V_{G_0}) \cap E_{G_0}| \geq h(G)|\bar{s}_X^V \cap V_{G_0}|$ . Hence, the relative weight of  $L'$  and  $\bar{L}$  is lower bounded by  $h(G)$ .

**Lemma 3.** *Given a graph  $G$  that satisfies the first two desiderata in Theorem 2, its cycle sparsification  $\bar{G}$  satisfies the first three desiderata in Theorem 2.*

**Proof.** The degree of the cycle-sparsified graph  $\bar{G}$  is upper bounded by the degree of  $G$  plus the constant  $(\deg(G)c + 2)$ , where  $c$  is the congestion number of cycles in the chosen basis that are assigned to a single layer of  $\bar{G}$ . Hence, if the graph  $G$  satisfies desideratum 1, so too does  $\bar{G}$ . The cycle-sparsified graph  $\bar{G}$  supports matchings  $\mu_i$  that can be routed entirely through the first layer since that layer is already a connected graph containing all the non-dummy vertices. Hence, assuming the graph  $G$  satisfies desideratum 2, so too does  $\bar{G}$ . The cycle-sparsified graph  $\bar{G}$  has a cycle basis consisting of length-three and length-four cycles by construction. Hence,  $\bar{G}$  satisfies desideratum 3.

We use the freedom in choosing checks of the deformed code per Lemma 1 for the cycle-sparsified graph to take advantage of the layered structure. This leads to a very natural cycle basis constructed from cycles with a length of  $\leq 4$  coming in two types. First, for each layer  $i = 1, 2, \dots, r-1$  and each edge  $e \in E$  of the original graph, there is a length-four cycle  $y_{i,e}$ , which consists of the copy of edge  $e$  in layer  $i$ , the copy of the edge  $e$  in layer  $i+1$ , and the two edges connecting the layers together and adjacent to those copies of  $e$ . Second, there are length-three cycles resulting from the triangular cellulations.

In Definition 3, we have chosen to cellulate the cycles into triangles as they have minimal weight. A similar procedure applies for squares, or even arbitrary polygons, which need not have a uniform number of edges. We note that using squares is also natural in the sense that square cycles already appear between layers in the cycle-sparsified graph.

For a constant-degree graph, there are  $\Theta(|V|)$  cycles in a minimal generating set. For a random expander graph, making an appropriate choice, almost all generating cycles are expected to be of length

$O(\log |V|)$ . In this case, we expect a cycle degree of  $O(\log |V|)$  and a number of layers  $R_G^c = O(\log |V|)$  required for cycle sparsification. We remark that the decongestion lemma<sup>45</sup> establishes a worst-case bound  $R_G^c = O(\log^3 |V|)$  for cycle sparsification of a constant-degree graph. This leads to the overhead scaling bound in Theorem 3; ref. 46 provides a detailed review of this point. On the other hand, for some cases of the gauging measurement, no sparsification is required, such as the BB code example presented later. It would be interesting to understand conditions under which this occurs generally.

### Fault-tolerant implementation

In this section, we provide an overview of the fault-tolerant operation of the gauging measurement procedure. Detailed definitions and proofs of fault tolerance are provided in the spacetime code and fault distance section in the Supplementary information.

To prove Theorem 4, we analyse the gauging measurement in a standard phenomenological noise model<sup>50</sup> in which qubits and measurements of Pauli operators can suffer errors, but we ignore circuit-level details that may spread errors or introduce other sources of correlated errors. Our main strategy to ensure fault tolerance is to repeat the measurement of the checks of the deformed code, that is,  $A_\nu$ ,  $B_\nu$  and  $\bar{s}_\nu$ , at least  $d$  times, where  $d$  is the code distance of the original code. Intuitively, this provides redundant, and hopefully independent, measurements of logical  $L$  since the set of  $A_\nu$  measurements at any time  $t = 1, 2, \dots, d$ , denoted as  $\{A_\nu^{(t)}\}$ , would ideally satisfy  $L = L^{(t)} := \prod_\nu A_\nu^{(t)}$ . If the  $L^{(t)}$  values are not constant in time, however, we know an error occurred and can, in some cases, correct it.

The complete proof of Theorem 4 is involved; we provide details in the spacetime code and fault-distance section in the Supplementary Information and only summarize the main ideas here. This starts with some basic definitions (similar to ref. 51).

**Definition 4.** (Space and time faults). *A space fault is a Pauli error operator that occurs on some qubit during the implementation of the gauging measurement. A time fault is a measurement error in which the result of a measurement is incorrectly reported during the implementation of the gauging measurement. A general spacetime fault is a collection of space and time faults.*

For the purposes of our discussion, state initializations can be implemented via single-qubit  $Z$  measurement followed by a conditional application of  $X$  to prepare the  $|0\rangle$  state. State initialization faults (occurring, for example, when an edge qubit is initialized in  $|1\rangle$  instead of  $|0\rangle$  due to an error) can be considered space faults by decomposing them into a perfect initialization followed by a Pauli error.

**Definition 5.** (Detectors). *A detector is a collection of state initializations and measurements that ideally yield a deterministic result, in the sense that the product of the observed measurement results must be  $+1$  independent of the individual measurement outcomes, if there are no faults in the procedure.*

The set of detectors defines a spacetime code. We enumerate the detectors in gauging measurement in lemma 1 in the Supplementary Information. Generally, detectors are formed from subsequent measurements of the same check operator. This changes slightly at the first round of measurement in the deformed code, where the measurements of  $s_i$  and  $\bar{s}_i$  together form a detector, and on the return to the original code when the edge qubits are measured out, where the measurements of  $s_i$ ,  $\bar{s}_i$ , and of the edge qubits in  $\mu_i$  altogether form a detector.

**Definition 6.** (Syndrome). *The syndrome caused by a spacetime fault is defined to be the set of detectors that are violated in the presence of the fault. That is, the set of detectors that do not satisfy the constraint that the observed measurement results multiply to  $+1$  in the presence of the fault.*

With these definitions in hand, it is natural to define a spacetime logical fault to be a collection of space and time faults that violates

no detectors, or in other words, has a trivial syndrome. Similarly, by propagating the errors through the procedure, we can determine if a spacetime logical fault affects the result of the gauging measurement procedure, either by flipping the logical measurement result or by causing a logical operator other than  $L$  to be applied. If the spacetime logical fault does not affect the logical outcome of the procedure, we say it is a spacetime stabilizer; otherwise, it is a non-trivial spacetime logical fault. The minimum number of faults in a non-trivial spacetime logical fault is called the spacetime fault distance.

In the spacetime code and fault-distance section in the Supplementary Information, theorem 1 shows that the spacetime fault distance of gauging measurement is  $d$  provided that a graph  $G$  satisfying desideratum 3 in theorem 2 is used, and that at least  $d$  rounds of syndrome measurement in the deformed code are performed. Interestingly, we can show this by separately considering (1) that it takes at least  $d$  measurement faults to cause a non-trivial spacetime logical fault (that is, the time fault distance is at least  $d$ ; lemma 3 in the Supplementary Information) and (2) that the distance of the deformed code, or what we called the space fault distance in lemma 2, is also at least  $d$ . Effectively, these two failure mechanisms, time-like or space-like faults, can be cleanly separated (lemma 4 in the Supplementary Information) by multiplying an arbitrary spacetime logical fault by spacetime stabilizers, the structure of which we also explicitly describe in lemma 2 in the Supplementary Information.

### References

50. Dennis, E., Kitaev, A., Landahl, A. & Preskill, J. Topological quantum memory. *J. Math. Phys.* **43**, 4452–4505 (2002).
51. McEwen, M., Bacon, D. & Gidney, C. Relaxing hardware requirements for surface code circuits using time-dynamics. *Quantum* **7**, 1172 (2023).

### Acknowledgements

We thank T. Jochym-O'Connor and G. Zhu for useful discussions, A. Cowtan for useful comments on a draft, E. Swaroop for bringing our attention to the Hastings–Freedman decongestion lemma and S. He for explaining how the  $\log^3$  factor results implicitly from that lemma's proof. T.J.Y. thanks A. Cross for sharing his method of using CPLEX to calculate the code distances. D.J.W. thanks L. Cohen for an inspiring discussion. This work was initiated when D.J.W. was visiting the Simons Institute for the Theory of Computing.

### Author contributions

D.J.W. and T.J.Y. contributed extensively to the paper.

### Competing interests

US patent application 18/897427 (filed on 26 September 2024 and naming D.J.W. and T.J.Y. as co-inventors) contains technical aspects from this paper.

### Additional information

**Supplementary information** The online version contains supplementary material available at <https://doi.org/10.1038/s41567-026-03220-8>.

**Correspondence and requests for materials** should be addressed to Dominic J. Williamson.

**Peer review information** *Nature Physics* thanks the anonymous reviewers for their contribution to the peer review of this work.

**Reprints and permissions information** is available at [www.nature.com/reprints](http://www.nature.com/reprints).