

Fault-tolerant quantum circuits on connectivity-constrained hardware with SWAP gates

Shao-Hen Chiew^{1,2,*}, Ezequiel Ignacio Rodríguez Chiacchio¹, Vishal Sharma¹,
Jing Hao Chai¹, and Hui Khoo Ng^{1,3,4,†}

¹*Entropica Labs, 186b Telok Ayer Street, Singapore 068632, Singapore*

²*Institute of Physics, École Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland
and Center for Quantum Science and Engineering, EPFL, CH-1015 Lausanne, Switzerland*

³*Department of Physics, National University of Singapore, Singapore, Singapore*

⁴*Centre for Quantum Technologies, National University of Singapore, Singapore, Singapore*



(Received 10 October 2024; accepted 4 June 2026; published 20 July 2026)

In near-term quantum computing devices, connectivity between qubits remain limited by architectural constraints. A computational circuit with given connectivity requirements necessary for multiqubit gates has to be embedded within physical hardware with fixed connectivity. Long-distance gates have to be done by first routing the relevant qubits together. The simplest routing strategy involves the use of SWAP gates to swap the information carried by two unconnected qubits to connected ones. Ideal SWAP gates just permute the qubits; real SWAP gates, however, have the added possibilities of causing simultaneous errors on the qubits involved and spreading errors across the circuit. A general swap scheme thus changes the error-propagation properties of a circuit, including those necessary for fault-tolerant functioning of a circuit. Here, we present a simple strategy to design the swap scheme needed to embed an arbitrary circuit onto a physical hardware with constrained connectivity, in a manner that preserves the fault-tolerant properties of the abstract circuit. Our approach is generally applicable, and works on any circuit, including ones that carry out logical operations and error correction. The embedded circuit will, of course, be noisier, compared to a native implementation of the abstract circuit, but we show in the examples of embedding surface codes on heavy-hexagonal and hexagonal lattices that the deterioration is not severe. This then offers a straightforward solution to implementing circuits with fault-tolerance properties on current hardware.

DOI: [10.1103/crv2-99qd](https://doi.org/10.1103/crv2-99qd)

I. INTRODUCTION

Quantum computing offers the potential to solve problems intractable on classical computers. Full realization of that potential demands the use of large-scale fault-tolerant quantum hardware [1–4] capable of reliable operations between arbitrary subsets of qubits in the computer. In near-term devices, connectivity between qubits remains limited by architectural constraints, so operations between some qubits—often spatially distant ones—cannot be directly implemented [5–8]. Instead, one employs a procedure that routes the information carried by those qubits to ones that are connected, perform the necessary operation, and then route the information back.

Routing procedures, integral to circuit synthesis [9,10] and compilation [11], range from elaborate entanglement-based ones that create a “portal” between the original and target qubits [12], physical spatial shuttling of the qubits

[13–15], to strategies that rely on SWAP gates to effect the information transport along some “wire” of intervening qubits [16–20]. Ideal SWAP gates simply permute the qubits; real SWAP gates, however, can cause simultaneous errors on the qubits involved, and can spread errors across the circuit. A swap scheme thus generally changes the error-propagation properties of a circuit, and can destroy fault-tolerance properties—which control the spread of errors in the circuit due to imperfect components—carefully designed into the original circuit.

Nevertheless, we show here that, by a restriction of the allowed swap moves, swap-based routing schemes can allow us to embed and implement *arbitrary* circuits on physical hardware with constrained connectivity in a manner that preserves their fault-tolerance properties. Specifically, the error patterns from faults in the embedded circuit resemble those of the original circuit; the former thus inherits any fault-tolerant properties—determined by the set of error patterns that can be corrected—of the latter. The simplicity of our swap rules makes them easy to incorporate into existing circuit-synthesis algorithms. Of course, there will be increased noise in the embedded circuit, from the added SWAP gates. We show, however, that in surface-code circuits embedded on heavy-hexagonal and hexagonal (or honeycomb) lattices, our swap schemes are fairly low depth so that the noise deterioration is mild. Our routing strategy—applicable to *arbitrary* circuits

*Contact author: shao.chiew@epfl.ch

†Contact author: huikhoo.n@nus.edu.sg

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

including those that implement error correction, logical operations, algorithmic subroutines, etc.—thus offers an immediate path to implementing any circuit with fault-tolerant properties on current hardware.

Below, we begin in Sec. II with some basic concepts, and then we explain our main result in Sec. III: We propose a set of swap rules that a routing strategy must satisfy, and we prove that such routing strategies preserve the fault-tolerance properties of a given circuit. We illustrate our approach in Sec. IV for the case of implementing the error-correction circuits for the current popular surface codes on hardware with heavy-hexagonal- or hexagonal-lattice connectivity. We explicitly demonstrate the fault-tolerance-preserving nature of our routing strategy by reproducing standard fault-tolerance threshold graphs for the embedded circuits. Furthermore, we show that the noise deterioration in these examples, due to the additional SWAP gates needed to overcome the connectivity mismatch, is not severe. We conclude in Sec. V. For the ease of the reader, we have relegated much of the technical details to the Appendixes, with Appendix A explaining how our swap rules can be incorporated into routing search algorithms, while Appendix B provides full information on the examples discussed in Sec. IV.

II. PRELIMINARIES

Consider an *abstract circuit*, acting on *abstract qubits*, that carries out a specific function. An *interaction graph* describes the connectivity (with specified temporal order) needed for the multiqubit gates in the circuit. We want to implement this circuit on a given hardware, with its *physical qubits*, primitive gates, and connectivity, i.e., a *device graph* that describes the possible multiqubit gates between physical qubits. Each abstract qubit is assigned to exactly one physical qubit at any time. We refer to these physical qubits as *computational qubits*; leftover ones are *routing qubits*. We assume that the abstract circuit is compiled so that it employs only primitive gates available on the hardware, but it may contain multiqubit gates on abstract qubits assigned to computational qubits not connected on the device graph; we call these *long-distance gates*. The actual circuit carried out on the hardware, including SWAP gates and routing qubits, is referred to as the *physical circuit*. *Computational operations* are those in the physical circuit that correspond to abstract circuit operations.

In the physical circuit, a long-distance gate is realized as a sequence of swaps to bring the computational qubits to connected positions on the device graph so that the multiqubit gate can be implemented. We abstract this back to an effective multiqubit gate on the abstract qubits by tracing away the routing qubits. In the ideal case, this effective gate is equal to the abstract gate; in reality, the SWAP gates can introduce errors so that the effective gate has a different noise description than a native abstract gate. We refer to the abstraction of the physical circuit back into action purely on the abstract qubits, together with the associated noise, as the *embedded circuit*.

Central to our discussion is the SWAP gate, a two-qubit linear operation that acts as

$$\text{SWAP}(|\phi\rangle_1|\psi\rangle_2) = |\psi\rangle_1|\phi\rangle_2, \quad (1)$$

for states $|\phi\rangle$ and $|\psi\rangle$ on the two participating qubits; the operation extends to arbitrary (nonproduct) two-qubit states by linearity. An ideal SWAP gate interchanges the states carried by the two qubits. Errors in the input emerge unchanged in the output, following the respective qubit that brought in the error. A SWAP is thus like a spatial weaving of quantum wires, with the state and errors continuing to be carried by the same wire. A faulty SWAP gate, however, can introduce new errors on one or both qubits involved; it can also spread errors such that incoming errors on one qubit can contaminate the other qubit.

We employ the standard noise model used in fault-tolerance theory. When a circuit location—a spacetime location in the circuit at which a state preparation, a gate (including identity for a waiting step), or a measurement is performed—functions imperfectly, we say that a *fault* has occurred there. Faults can cause simultaneous errors on one or more of the qubits participating in that location; errors are allowed to be arbitrary operators (we expect our approach to generalize to biased-noise situations [21,22]). A *fault path* refers to a specification of faulty circuit locations; its weight is the number of faults. Each fault path results in a set of *error patterns* on the qubits, with each error pattern specifying the (normalized) error operators. The weight of an error pattern is the weight of the associated fault path. For concreteness, we assume that faults occur independently on individual circuit locations, each with probability p . This can be generalized to a local model, where we only require that ℓ faults occur with probability $\leq O(p^\ell)$.

We use this noise model for both physical and abstract circuits. For the latter, we imagine implementing the abstract circuit without SWAPs, disregarding hardware connectivity constraints. This gives the noisy abstract circuit, against which the performance of the embedded circuit is benchmarked.

III. ERROR-PATTERN-PRESERVING ROUTING

We want a swap-based routing schedule that implements the interaction graph on the given device graph, with fault-tolerance properties specified below. The abstract circuit is embedded as a sequence of *computational layers*, during which computational operations occur. Each adjacent pair of computational layers is separated by a sequence of SWAPs that route computational qubits around so that long-distance gates in the next computational layer can be implemented.

We allow for two types of SWAP gates: (1) type-1 SWAP—a SWAP gate between a computational qubit and a routing qubit; and (2) type-2 SWAP—a SWAP gate between two computational qubits that participate in the same operation (or circuit location) in the computational layer immediately before or after the swap sequence. We refer to that computational operation as one *associated with* the type-2 SWAP. Our routing schedules use only type-1 and type-2 SWAPs, but not swaps between two routing qubits—These add no useful moves to routing the information.

Our main result is to show that an error pattern in the embedded circuit arising from faulty SWAPs can be mimicked by an equivalent- or lower-weight fault path on the computational operations only. The set of error patterns for the embedded circuit then coincides with that for the noisy abstract circuit;

we refer to this as the *error-pattern-preserving* (EPP) property of a routing schedule.

To show the EPP property, we first consider routing schedules comprising only type-1 SWAPs. Faulty type-1 SWAPs can introduce spatiotemporally correlated errors in the embedded circuit absent in the abstract circuit. This can occur when multiple computational qubits are swapped with the same routing qubit. Nevertheless, the following lemma shows that such correlated errors occur with a probability resembling that of independent errors.

Lemma 1. Consider a weight- k error pattern in the noisy abstract circuit. If only type-1 SWAPs are allowed, this error pattern spreads to weight $k + \ell$ in the embedded circuit only if at least ℓ type-1 SWAPs are faulty, which occurs with probability $O(p^\ell)$.

Proof. If all SWAPs are ideal, $\ell = 0$, and the weight- k error pattern is inherited without change by the embedded circuit. A faulty type-1 SWAP, occurring with probability p , can introduce errors on the participating computational and routing qubits. The error on the computational qubit adds an error in the error pattern in the embedded circuit. The error on the routing qubit—either generated by the faulty SWAP or spread by the faulty SWAP from an existing error in the computational qubit—can spread to another computational qubit it is swapped with only if that subsequent SWAP is also faulty. Hence, each additional error on a computational qubit arises only from a faulty SWAP; ℓ additional errors require at least ℓ faulty SWAPs, which occur with probability $O(p^\ell)$. ■

Lemma 1 tells us that we can view any error pattern modified by faulty type-1 SWAPs as if they were generated by additional faults, one for each computational qubit involved in a faulty SWAP, occurring with probability p . With this, we can prove our main result that a routing schedule with only type-1 and type-2 SWAPs is error-pattern preserving.

Theorem 1 (EPP routing schedules). Consider an abstract circuit that employs a routing schedule with only type-1 and type-2 SWAPs. For every error pattern that arises in the embedded circuit from a fault path in the physical circuit, there exists an equal- or lower-weight fault path in the noisy abstract circuit that can give the same error pattern.

Proof. The proof is constructive. Consider an error pattern in the embedded circuit and its associated fault path in the physical circuit. We first regard, just for a moment, the type-2 SWAPs in the given routing schedule as computational gates; the remaining routing schedule only has type-1 SWAPs and Lemma 1 applies. Each fault on a type-1 SWAP can then be viewed as a single-qubit fault on the computational qubit involved. This single-qubit fault is allocated to the nearest computational operation involving that qubit; the type-1 SWAP is then regarded as fault-free. Faults allocated to multiple qubits that participate in the same computational operation are combined into a single fault on that operation; if that operation is already faulty, any additional faults are absorbed into the existing fault. Since we assume arbitrary errors for every fault, we need not worry about how specific error operators get modified in subsequent operations. After reallocating all faults in the type-1 SWAPs to computational operations, the physical circuit has become one with fault-free type-1-SWAPs-only routing schedule. We remove these type-1 SWAPs from the circuit, since they simply enact permutations of the

computational qubits and do not affect the error patterns. We are then left with faulty type-2 SWAPs (earlier regarded as computational gates) immediately adjacent to computational layers. The fault on each type-2 SWAP can then be absorbed into its associated computational operation; the type-2 SWAP is then regarded as fault-free and can again be removed. We are finally left with a circuit, now viewed as a noisy abstract circuit with no SWAPs, with a fault path as constructed above to give the same error pattern as the original embedded circuit. That this fault path has equal or lower weight than the original physical fault path is assured by Lemma 1 and the fact that our construction can merge, but not split, multiple faults into a single one. ■

Theorem 1 allows us to argue that our routing schedules preserve the fault-tolerance properties of an abstract circuit. A quantum circuit, built upon an error-correcting code, is fault tolerant if it does not convert correctable errors into uncorrectable ones, even if the circuit is faulty. Concrete conditions can be enforced to assure proper behavior when no more than t faults occur in the circuit, for a scheme built upon a code that corrects errors on t or fewer qubits (see, e.g., Refs. [4,23–25]). All such conditions involve checks on the error patterns arising from fault paths of weights t or smaller. Our EPP routing schedule preserves the set of such error patterns according to Theorem 1, and hence the embedded circuit automatically inherits the fault-tolerance properties of the abstract circuit. Put differently, if the abstract circuit is such that t or fewer faults anywhere lead only to correctable error patterns, the physical circuit with t or fewer faults also results in an embedded circuit with only correctable error patterns.

We have thus reduced the problem of fault-tolerantly embedding an abstract circuit on physical hardware to a search of an EPP routing schedule. As mentioned in the introduction, the general qubit routing problem is well studied. By adding the EPP constraints of limiting to type-1 and type-2 SWAPs, we can easily adapt existing qubit routing algorithms to give EPP solutions. In Appendix A, we explain how to do this for a simple greedy search algorithm, and use that in our examples below; the same ideas can be adapted for a more elaborate search.

Of course, the embedded circuit will be noisier than the noisy abstract circuit—The physical circuit, with the added SWAP gates, has many more ways in which faults can occur. The number of ways in which more than t faults occur in the physical circuit will hence generally be larger than that for the abstract circuit. These $>t$ -fault situations give the failure modes of the abstract or embedded circuit, and determine the resulting logical error rate. The noisier embedded circuit will thus have a larger logical error rate, which translates into a more stringent fault-tolerance noise threshold below which errors can be removed by noisy error correction. Nevertheless, we show in our surface-code examples below that the noise deterioration is mild.

IV. SURFACE-CODE EXAMPLES

As an example, we apply our results to embed the error-correction circuit for the rotated planar surface code [26,27]—a popular approach to error correction—on hardware with heavy-hexagonal (as used in IBM Quantum

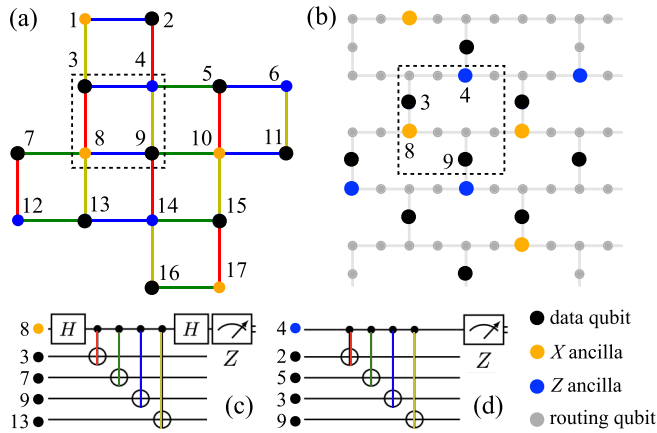


FIG. 1. (a) The interaction graph of the distance-3 rotated planar surface code, with each edge colored according to the temporal layer in which a CNOT occurs. The unit cell is enclosed in the dotted box. (b) The initial placement of the surface-code lattice onto the heavy-hexagonal device graph. The unit cell is enclosed in the dotted box. The hexagonal case is constructed in a similar manner; see Appendix B for full details. Panels (c) and (d) show the X and Z syndrome extraction circuits, respectively, with the qubit numbering and coloring of the CNOTs matching those in panel (a).

platforms) and hexagonal lattices as their device graphs. The rotated planar surface code has data qubits on the vertices of a square lattice, and X- and Z-type ancillary qubits at the center of each plaquette in a checkerboard arrangement [Fig. 1(a)]. The X and Z ancillas are used in the syndrome extraction (SE) circuits [Figs. 1(c) and 1(d)]. Syndrome extraction happens across the entire surface-code lattice simultaneously, with the X and Z SE circuits running in parallel. Each SE round comprises an ancilla-preparation layer, four CNOT layers, and a measurement layer. Our abstract circuit implements a full error-correction cycle for a distance- d surface code, comprising d rounds of syndrome extraction using the SE circuits, followed by syndrome decoding to infer and (attempt to) correct the errors. The CNOTs in the SE circuits give the interaction graph a (rotated) square-lattice structure, a mismatch with the heavy-hexagonal and hexagonal device graphs.

The periodic interaction and device graphs allow us to consider only unit cells of the lattices. This same unit cell works for all surface-code distances, as the code grows only in area while retaining the local structures; the same routing schedule thus works for all distances. We choose an initial placement of the abstract unit cell onto the device-graph unit cell, i.e., assign each abstract qubit to a physical qubit [see Fig. 1(b)]. With these initial layouts, we search for EPP routing schedules, looking for ones with minimal depth. The periodic surface-code structure further allows for a simple parametrization of routing schedules, which enables an easier search (see Appendix B 1).

For the heavy-hexagonal situation, our minimal-depth solution comprises five SWAP layers per SE round, with each physical qubit (excluding the altogether unused ones) participating in 3.5 SWAPs on average (see Appendix B 4 for further details, and video in the Supplemental Material [28] illustrating the action of the routing schedule). At the end

of one SE round, the computational qubits are displaced one unit cell diagonally from the initial layout positions; in the next round, the directions and ordering of the routing moves are reversed—allowed by the lattice symmetry—so that the computational qubits return to the initial positions after yet another cycle, to prevent walk-off of the embedded circuit. The minimal-depth EPP schedule for the hexagonal situation is much simpler: One needs only a single type-2 SWAP layer between the final two CNOT layers for each SE round.

To illustrate the EPP nature of our routing schedules, we perform numerical simulations (using Stim [29]) of the physical circuits for different distances d under circuit-level depolarizing noise characterized by error probability p . To imitate the standard situation where one type of two-qubit gate is available, each SWAP is composed from three consecutive CNOTs, needed anyway in the SE circuits. Pauli errors are inserted at all layers of the circuit: initialization to $|0\rangle$ and Z-measurement suffer bit flip X with probability p ; nonidentity single-qubit gates have Pauli X , Y , or Z errors, each with probability $p/3$; CNOTs experience nonidentity two-qubit errors $\sigma_i \otimes \sigma_j$, for $\sigma_i, \sigma_j = \mathbb{1}, X, Y, Z$, each with probability $p/15$. Identity gates are treated as ideal. This circuit-level noise is applied to the physical circuits with our EPP routing for different d , and we benchmark the logical error probability p_L at the end of the error-correction cycle against that for an abstract circuit with the same circuit-level noise.

For decoding, we employ PyMatching [30,31] based on the minimum-weight-perfect-matching (MWPM) algorithm, which remains applicable here. Specifically, the errors that arise from the additional SWAP gates are incorporated into existing error mechanisms in the manner as explained in Theorem 1, by updating the edge weights in the matching graph; see Appendix B 3 for further details. One could, of course, improve decoding accuracy by taking into account the error patterns that arise from, say, swapping with the same faulty routing qubit. Dealing with such correlated patterns, however, requires the use of hypergraphs to indicate the multiple edges that are simultaneously erroneous. We opt instead for simpler standard graph-based decoding procedures, by decomposing the correlated error patterns into independent ones, which work adequately.

The results for the heavy-hexagonal case are given in Fig. 2, with p_L plotted against the effective error probability p_{eff} , equal to p for the abstract circuit, but rescaled to the best-fit value of $p_{\text{eff}} = 3.63p$ for the embedded circuit. This p_{eff} is the effective error probability for the embedded circuit, absorbing the noise introduced by the SWAP gates into an effective noise on the abstract qubits. The horizontal rescaling brings the curves for the abstract and embedded circuits on top of each other for easier comparison. We see that, for every d , the gradients of the curves for the abstract and embedded circuits match, even though the curve fitting for p_{eff} was performed only for $d = 35$. This demonstrates the EPP nature of our routing schedule: How the gradients of the p_L versus p_{eff} (or p) lines change as d grows tells us about the fault-tolerance properties of the circuits; a preservation of the fault-tolerant nature of a circuit demands an unchanged gradient behavior. Consequently, the p_L -versus- p_{eff} lines for different d also intersect at the same threshold point for both abstract and embedded circuits.

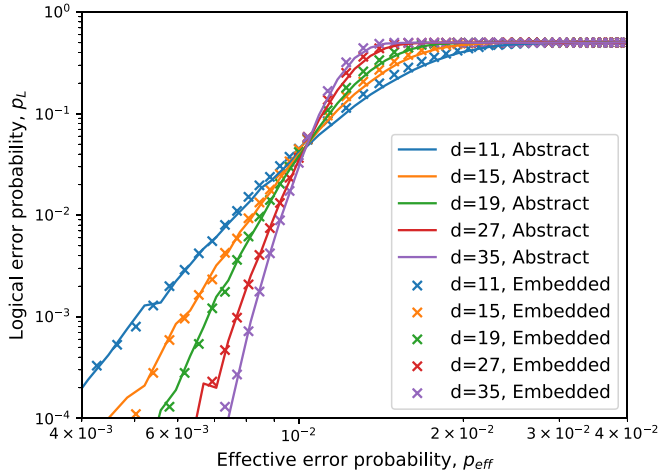


FIG. 2. The logical error probability p_L vs the effective error probability p_{eff} for the heavy-hexagonal lattice situation under depolarizing noise with error probability p . For the abstract circuit, $p_{\text{eff}} = p$; for the embedded circuit, $p_{\text{eff}} = 3.63p$, obtained by fitting both curves for $d = 35$ below the threshold value.

We thus conclude that our EPP routing schedule preserves the fault-tolerant nature of the surface-code error-correction circuit, with a worsening of the threshold by a factor of 3.63; this factor will be smaller if each SWAP is a single high-fidelity operation, rather than the three CNOTs assumed here. An analogous simulation of the hexagonal situation gives a nearly identical plot, but now with $p_{\text{eff}} = 1.25p$ for the embedded circuit; see Appendix B 5 for full details, including the choice of the initial placement of the abstract unit cell on the device graph.

The rescaling factors of 3.63 and 1.25 for the heavy-hexagonal and hexagonal cases can be derived, in part, from an analysis of the effective noise of a CNOT gate in the SE circuit, when implemented via SWAP gates in the embedded circuit. The physical CNOT itself is assumed to be subjected to two-qubit depolarizing noise with strength p ; a SWAP gate sees depolarizing noise with strength p_{SWAP} , possibly different from p , in anticipation of the fact that the SWAP may be composed from a few $[O(1)]$ more primitive physical gates. In our example, we assume a SWAP to be made up of three CNOTs so that $p_{\text{SWAP}} = 3p$ for the heavy-hexagonal case; for the hexagonal case, we found that we are always able to combine two of the three CNOTs in the SWAP with the computational CNOTs so that each SWAP requires only one additional CNOT and hence $p_{\text{SWAP}} = p$. For a CNOT implemented with $n_{1,1}$ and $n_{1,2}$ type-1 SWAPs involving, respectively, the first and second computational qubits of the CNOT, and n_2 type-2 SWAPs, we can write down the overall noise channel for that CNOT operation. In general, this channel is not depolarizing noise—there is asymmetry in the strengths of the different Pauli noise terms—but we take the effective depolarizing noise strength to be the deviation from 1 of the identity term. Furthermore, to obtain a single number for that noise strength for the different CNOTs in the SE abstract circuit, we average the swap parameters $n_{1,1}$, $n_{1,2}$, and n_2 over the routing solution for the whole SE circuit. This then gives us $p_{\text{eff}} \simeq 3.1p$ and $1.25p$, respectively, for the heavy-hexagonal and hexagonal

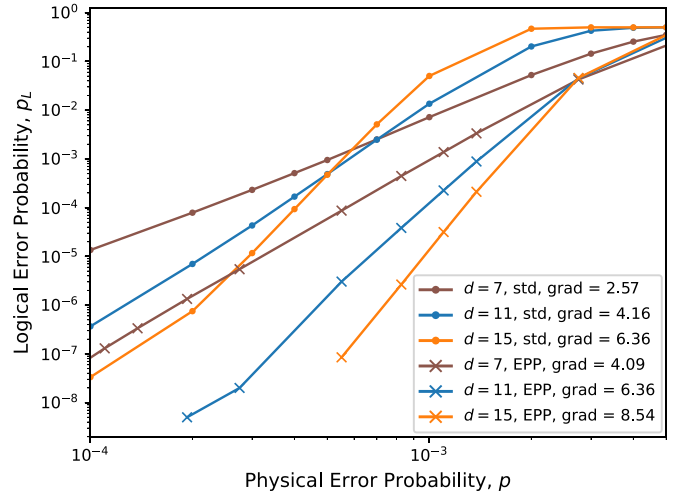


FIG. 3. Comparison of the surface-code circuit embedding from a standard (non-fault-tolerant) routing algorithm SABRE (lines with circular markers) vs our EPP protocol (lines with cross markers). The logical error probability p_L is plotted against the physical error probability p for different code distances $d = 7, 11$, and 15 . The lines from our EPP embedding have gradients consistent with the code distances [gradient = $\frac{1}{2}(d-1) + 1 = 4, 6$, and 8 for $d = 7, 11$, and 15]; in contrast, the best-fit gradients for the SABRE lines are lower by factors of 1.3–1.6, indicative of non-fault-tolerant embeddings that result in reductions in the effective code distance.

cases, with deviation from the best-fit numbers arising from the analytical approximations described above.

Compared with recent proposals for implementing the surface code on hexagonal [32] and heavy-hexagonal lattices [33,34], we see a threshold deterioration of the same order of magnitude with the same spatial footprint. Notably though, those schemes relied on first modifying the standard SE circuits to have a *hexagonal* interaction graph before embedding them onto the device graph. Our approach requires no such modification, and is applicable beyond this narrow case of surface codes on hexagonal and heavy-hexagonal lattices, to arbitrary circuits and device graphs for which alternative interaction graphs may not be available.

With this surface-code example, we can also illustrate the difference in performance between the embedded SE circuits obtained from our EPP routing scheme and those from standard protocols that do not impose fault-tolerance requirements. Specifically, we employ SABRE [17,35] (available through Qiskit), a highly optimized routine for circuit embedding and swap-gate routing, to embed the surface-code SE circuit onto the heavy-hexagonal lattice. The resulting SABRE-routed circuits are then used in place of our EPP circuit in the same error-correction simulations as before. The resulting logical error probabilities are plotted in Fig. 3, alongside those from our EPP circuit (the same data used in Fig. 2). Notably, even though the SABRE circuits require fewer physical qubits, our EPP-routed circuit has lower logical error probabilities. More importantly, the SABRE circuits show p_L -versus- p lines with gradients significantly smaller than those for the EPP circuit, corresponding to a reduction of code distance. This reduction can be attributed to the violation of fault-tolerance requirements, with SABRE employing

SWAP gates beyond the type-1 and type-2 SWAPs enforced in our EPP routing scheme. Further details of these numerical experiments are provided in Appendix B 6.

V. CONCLUSION AND DISCUSSION

We have shown that a restriction to type-1 and type-2 SWAPs results in EPP routing schedules, thus preserving the fault-tolerance properties of a circuit. Noisy type-1 SWAPs can be thought of as interactions between computational qubits and a spatiotemporally correlated environment formed by the routing qubits that never participate in computational operations. From this perspective, that type-1 SWAPs preserve fault-tolerance properties becomes obvious: It arises from the same reason that standard fault tolerance allows for certain weak interactions with an environment [4,36–38]. Together with the fact that type-2 SWAPs give errors resembling those from existing abstract operations, these observations provide the intuition behind the fault-tolerance-preserving nature of our approach.

The restriction to type-1 and type-2 SWAPs is easy to accommodate within existing routing algorithms. In our surface-code examples, the problem symmetries further enabled a concise search. Relaxing the swap-types restriction might be possible for surface codes, by including the expanded set of error patterns within the decoding algorithm, e.g., adding hypergraph edges in the syndrome lattice [39,40], but we leave this for future investigations.

Our surface-code examples show a mild noise increase from the added SWAPs. The swap scheme can possibly be further optimized by choosing a different initial embedding. Our procedure does not guarantee, in general, the existence of EPP schedules for given interaction and device graphs. A sparser embedding with more routing qubits would yield EPP schedules more readily, but at the cost of a larger spatial footprint.

We note that our current routing search algorithm, for simplicity, constrains SWAPs and computational gates into separate circuit layers. For our heavy-hexagonal example, we found an EPP routing schedule with six SWAP layers per SE round; upon inspection, that circuit can be compressed—by allowing SWAP-gate CNOTs and computational gates to occupy the same layers—to a depth equal to that of the minimal-depth solution discussed above. Interestingly, this six-SWAP-layer solution introduces only temporally, but not spatially, correlated errors, a property perhaps helpful in limiting the spread of errors. A more flexible circuit compilation program that handles simultaneous circuit compression and EPP routing search can potentially generate more solutions, including ones with desirable features.

More broadly, our routing approach enables a user to implement any circuit with fault-tolerant properties on a quantum computing device with a connectivity mismatch, provided an EPP routing solution can be found. Quantum computing hardware providers can also adopt our method as part of their compilation and transpilation protocol, and thus guarantee preservation of any fault-tolerant properties of a user-submitted circuit. Apart from devices with highly restricted size and connectivity, we expect that EPP routing strategies can be found for any circuit, simply by having a

sparser initial embedding and hence more available routing qubits. One can, of course, always employ the trivially fault-tolerant, but extremely wasteful, solution where there is a dedicated “wire” of routing qubits between every pair of computational qubits connected on the circuit’s interaction graph; such a solution would demand many SWAPs and incur high noise (not to mention, qubit and gate) overheads. Our result offers the possibility of reusing routing qubits and swapping past already-connected computational qubits (type-2 SWAPs), for a much more compact and resource-saving solution, while not spoiling the underlying fault-tolerant properties. On devices with highly restricted connectivities, e.g., qubits on a line with only nearest-neighbor gates, there may simply be no routing solutions, for a given circuit, that satisfy our type-1 and type-2 SWAP restrictions, or finding a valid routing solution may become more difficult with the SWAP limitations. But as we have seen in our comparison with the non-fault-tolerant SABRE routing, having routing solution that respects fault tolerance is not optional if we want to preserve the code distance and have error-correction function as originally designed.

In the long term, one would expect the choice of the error-correcting code and the associated fault-tolerant schemes for implementing a given algorithmic circuit to be solely the responsibility of the quantum computer service provider. In that case, fault-tolerant designs tailor-made for the specific device connectivity will be the norm, with no need for additional EPP guarantees. However, while the community is still grappling with different hardware connectivities and error-correction approaches, our solution offers a straightforward way to implement fault-tolerant circuits built upon different codes on available hardware in the near term. Even in the medium-to long-term periods, we expect our approach to serve as a standard tool to bridge any hardware connectivity shortfalls, in software-hardware codesign tasks, to manage temporarily faulty qubits or gates, etc.

Going beyond swap-based routing, one could also ask for EPP *quantum* routing, a notion explored in Ref. [41]. Quantum routing permits the use of arbitrary two-qubit unitary operators as the means for effecting qubit transport; routing using only SWAP gates is termed *classical* routing in their language. The arguments behind our Lemma 1 and Theorem 1 rely on the fact that the unitary gate that enacts the routing does not spread errors unless a fault occurs. This is satisfied by the SWAP gate, but can also hold for more general gates. While the bounds shown in Ref. [41] preclude a complexity separation between classical and quantum routing, allowing for other two-qubit gates in the routing procedure opens up the playing field for designing lower-noise routing operations with smaller noise deterioration in the embedded circuit.

All in all, our work relaxes hardware connectivity requirements for fault-tolerant quantum computing tasks. Its general applicability provides a concrete route to implementing arbitrary circuits on hardware in a fault-tolerant manner, without having to redesign the abstract circuit. Our work also motivates the development of native, high-quality SWAP gates, so that the information transport adds only minimal noise. Perhaps, we can eventually achieve with SWAP gates, the high-fidelity transport possible today in neutral-atom experiments via optical tweezers [14].

ACKNOWLEDGMENT

H.K.N. acknowledges partial support from the National Research Foundation, Singapore and A*STAR under its CQT Bridging Grant.

DATA AVAILABILITY

Data on the numerical experiments are available upon reasonable request from the authors.

APPENDIX A: SEARCH ALGORITHMS FOR EPP ROUTING SCHEDULES

Here, we discuss search algorithms for routing schedules and explain how to incorporate our EPP constraints in a straightforward manner. We focus on a simple search procedure based on a greedy, distance-minimizing search [18], but the same principles apply to other more elaborate methods [18,42,43].

We begin with an abstract quantum circuit given as an ordered product $U = U_L U_{L-1}, \dots, U_1$ of L layers, where each U_i contains two-qubit gates with no overlapping supports. We initialize $E_I^1, \dots, E_I^L$ to be the edges in the interaction graph G_I corresponding to the two-qubit gates in $U_1, \dots, U_L$, respectively. The interaction graph is denoted as $G_I \equiv (V_I, E_I)$, where $V_I(E_I)$ is the set of nodes (edges) of the graph; similarly, the device graph is $G_D \equiv (V_D, E_D)$. For the embedding to be possible, we must have $|V_D| \geq |V_I|$. A qubit mapping $\hat{p}_t$ denotes the collective assignment of abstract qubits to physical qubits at layer t .

Our goal here is to iteratively execute all two-qubit gates in $E_I^1, \dots, E_I^L$, and we keep track of the executed gates by removing them from the set. This is achieved by using SWAP gates to bring abstract qubits next to one another in the device graph $G_D \equiv (V_D, E_D)$, so that the remaining gates can be executed. In other words, defining $d : V_D \times V_D$ as the geodesic or shortest-distance function between nodes in the device graph G_D , for all edges $(q_1, q_2) \in E_I^1$, SWAP gates are used to update the mapping $\hat{p}$ so that $d(q_1, q_2) = 1$, in which case the edge (q_1, q_2) can be removed from E_I^1 . To quantify the collective distance between qubits of all edges in an edgeset E , define the objective function:

$$R(E) \equiv \sum_{(q_1, q_2) \in E} d(\hat{p}(q_1), \hat{p}(q_2)). \quad (\text{A1})$$

First, an initial embedding $\hat{p}_0$ is chosen, possibly in a manner that maximizes the number of executable edges in the first layer E_I^1 . These edges are subsequently executed, i.e., removed from E_I^1 . Next, consider the set of edges in G_D , where executing the corresponding SWAP gates at those edges maximally decreases $R(E_I^1)$. Here, we further impose the EPP constraints that allow only type-1 or type-2 SWAPs. SWAPs are executed at these edges, the embedding is updated accordingly (to $\hat{p}_1$), and the set of remaining edges E_I^1 are checked to see if any satisfies $d = 1$, in which case they are executed (removed from E_I^1). If no SWAPs or gates in E_I^1 can be executed, one of the remaining edges in E_I^1 is selected arbitrarily, and a type-1 or type-2 SWAP that decreases d is executed. The above procedure is repeated until E_I^1 is empty, at which point we move on to edges in the second layer E_I^2 ,

and so on until $E_I^1, \dots, E_I^L$ are all empty. If, at any step of the way, a type-1 or type-2 SWAP move cannot be found, the algorithm fails and terminates without a solution.

If successful, the above algorithm returns an EPP routing schedule. If it terminates without a solution, either an EPP schedule simply does not exist—this may be very likely for highly constrained device graphs—or the above greedy search must be improved to search through more routing schedules, typically at the expense of higher computational costs. One such improvement is to increase the depth of the search in each iteration, i.e., allowing the algorithm to look more than one move ahead to avoid getting stuck at a local minima. We refer the reader to Ref. [18] for further discussion of this aspect.

We elaborate on how this search algorithm was modified to produce EPP routing schedules for topological codes such as the surface code below.

APPENDIX B: DETAILS ON SURFACE-CODE EXAMPLES

Here, we provide details specific to the surface-code examples discussed in the main text.

1. Search algorithm for surface-code routing schedules

Topological codes, including surface codes, possess a large degree of locality and translational symmetry in the defining code stabilizers, directly reflected in their SE circuits and their associated interaction graphs. In cases where the quantum device has similar symmetries (common in current quantum hardware architecture), these properties can be leveraged to reduce the size of the routing search problem. Here, we describe a search algorithm that yields constant-depth (i.e., does not scale with code distance d) routing schedules for the surface = code error-correction circuit, with a computational cost that is independent of code distance. It can be straightforwardly generalized to other topological codes. This algorithm yields routing schedules for the examples discussed in the main text.

Since the interaction graph of the surface-code SE circuits and the device graphs are both periodic, with invariant local structures as the code distance increases, the core idea behind our approach is to embed a unit cell of the interaction graph onto a unit cell of the device graph, and subsequently search for the routing schedule for this unit. This avoids the need to search across the entire interaction- and device-graph lattices, which increase in size with code distance. The resulting schedule can then be tiled in a repeated manner up to the desired code distance, with tiles at the code boundary appropriately truncated to ensure that lower-weight stabilizers at the boundaries are measured correctly.

We begin by identifying the units cells of the interaction and device graphs. A unit cell in the interaction graph of the surface code is illustrated in Fig. 4(a). The computational qubits of the surface code can be classified into four species: the Z ancillas, the X ancillas, the X -data qubits (defined as a data qubit connected horizontally to two X ancillas in that layer of the the interaction graph), and the Z -data qubits (similarly defined as a data qubit connected horizontally to two Z ancillas). Each species of qubits can be distinguished by the ordering in which they interact with other qubits throughout

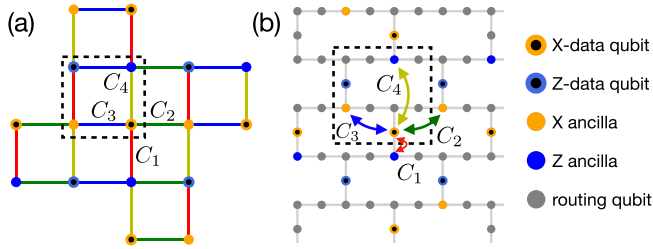


FIG. 4. The $d = 3$ surface-code example. The interaction graph is given in panel (a), and the heavy-hexagonal device graph in panel (b). Unit cells are marked by the dotted boxes. The four distinct types of qubits of the surface code are labeled on the right. Also shown are the four CNOTs experienced by an X-data qubit; gates C_1 , C_2 , C_3 , C_4 , and correspond to red, green, blue, and yellow colored lines, respectively, in the interaction graph (a), while the correspondingly colored arrows in panel (b) mark the pairs of computational qubits in the device involved in these operations.

the computation, which can be observed from the interaction graph and the ordering of CNOT gates of the surface code in Figs. 1(a) and 1(b). For instance, while data qubits interact with X and Z ancillas in an alternating fashion, X- (Z-) data qubits interact with Z (X) ancillas first. In the device graph, a unit cell that satisfies the same translational symmetry as the surface code is identified, marked by the dotted box in Fig. 4(b).

Next, a parametrization of the set of all routing schedules is required. Our approach consists of two parts, parametrizing the (1) initial embeddings and (2) swap layers separately:

(1) *Parametrization of initial embedding.* The four qubit species are assigned onto the unit cell of the device graph. If there are N lattice sites on the unit cell, there are a total of $N P_4$ possible assignments. For the heavy-hexagonal lattice, $N = 10$, giving 5040 possible initial assignments.

(2) *Parametrization of swap layer.* A swap layer can be defined by specifying the targets of the SWAP gates applied to each of the four species of qubits. For the heavy-hexagonal lattice, each qubit can be either idling or swapped with one of its two or three neighbors, so there are $\leq 4^4$ possible swap layers, which can be further reduced by enforcing the constraint on type-2 SWAPs. For convenience, we arrange the heavy-hexagonal and hexagonal lattices in a gridlike manner on a two-dimensional plane as in Figs. 1(b) and 4(b); this allows the action of each SWAP gate to be associated with a permutation with a qubit to its up (u), down (d), left (l), or right (r) directions, or idling (i).

In summary, the number of possible L -swap-layer routing schedules for a quantum error-correcting code with n species of qubits embedded onto a device graph of maximum degree Δ with N qubits per unit cell is upper bounded by $(N P_n \times n^{\Delta+1})^L$.

The general search algorithm described earlier can then be adapted to obtain EPP routing schedules in a distance-independent manner:

(1) Initialize an embedding of the four species of qubits in one of the $N P_4$ possible assignments in the unit cell in the device graph. For each of the four qubits, track the targets of the embedded CNOT gates (possibly located in adjacent unit cells),

which defines the distance/objective function in Eq. (A1). For example, for a particular X-data qubit, the targets of the effective gates are represented as curved colored lines labeled C_1 , C_2 , C_3 , and C_4 in Fig. 4(b), each corresponding to its abstract counterparts in the interaction graph of Fig. 4(a).

(2) Apply a swap layer (consisting of one of the directions u, d, l, r, i on each species of qubit) that minimizes the objective function in Eq. (A1) for the CNOT layer C_1 in a greedy manner. The embedding is updated.

(3) Check for any executable gates in C_1 , i.e., if the targets of the effective gates in layer C_1 are next to one another in the device graph in the current embedding.

(4) Repeat steps 2 and 3 until all gates in C_1 are executed, after which steps 2 and 3 are repeated sequentially with C_2 , C_3 , and C_4 .

If successful, the result is a set of swap layers $S_1, S_2, \dots$ interspersed between the CNOT layers C_1, C_2, C_3 , and C_4 that implements an EPP routing schedule.

2. Effective noise model

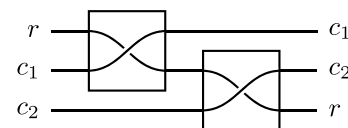
Here, we describe how to derive an effective noise model on the embedded circuit for our surface-code examples. Apart from explaining how we arrive at the scaled error probability parameter p_{eff} used in our plots in the main text, this serves also as an explicit illustration of Lemma 1 and Theorem 1 in the main text for the specific example of Pauli noise.

The physical noise model used in our simulations is that of depolarizing noise: A single-qubit gate sees a Pauli X, Y , or Z error with probability $p/3$ each; a two-qubit gate sees a two-qubit Pauli error with probability $p/15$ each. Idling qubits are assumed to be noiseless. Our discussion here, however, applies to a general Pauli noise model, where the single- and two-qubit gates see the following noise channels:

$$\begin{aligned} \mathcal{E}_q(\cdot) &\equiv \sum_{\alpha} p_{\alpha} \sigma_{\alpha}(\cdot) \sigma_{\alpha}, \\ \mathcal{E}_{q_1, q_2}(\cdot) &\equiv \sum_{\alpha\beta} p_{\alpha\beta} \sigma_{\alpha} \otimes \sigma_{\beta}(\cdot) \sigma_{\alpha} \otimes \sigma_{\beta}, \end{aligned} \quad (\text{B1})$$

where $\sigma_{\alpha} = I, X, Y, Z$, for $\alpha = 0, 1, 2, 3$. Note the trace-preservation requirements: $p_0 = 1 - \sum_{\alpha \neq 0} p_{\alpha}$ and $p_{00} = 1 - \sum_{\alpha\beta \neq 00} p_{\alpha\beta}$. The depolarizing channel has $p_{\alpha \neq 0} = p/3$ and $p_{\alpha\beta \neq 00} = p/15$, so that $p_0 = 1 - p = p_{00}$. The probability of an error in this case, whether it is a single- or two-qubit channel, is then just p .

The situation of Pauli noise gives particularly simple noise behavior, with faulty SWAP gates effectively introducing Pauli errors into the embedded circuit in an independent manner. To see this, we consider the following situation of two computational qubits c_1 and c_2 sequentially coupled to a routing qubit r by type-1 SWAPs:



If both SWAPs are faulty, each subjected to the two-qubit noise channel $\mathcal{E}_{q_1, q_2}$, straightforward calculation gives a product channel $\mathcal{E}'_{c_1} \otimes \mathcal{E}'_{c_2}$, where $\mathcal{E}'_q$ is the single-qubit

TABLE I. The effective noise channel on a CNOT in the embedded circuit. The probability expressions are accurate to linear order in p .

Pauli error	Probability
No error $\mathbb{1} \otimes \mathbb{1}$	$1 - p - p_{\text{swap}}[\frac{4}{5}(n_{1,1} + n_{1,2}) + n_2]$
$\sigma_i \otimes \mathbb{1}$	$\frac{1}{15} [p + p_{\text{swap}}(4n_{1,1} + n_2)], \quad i = 1, 2, 3$
$\mathbb{1} \otimes \sigma_j$	$\frac{1}{15} [p + p_{\text{swap}}(4n_{1,2} + n_2)], \quad j = 1, 2, 3$
$\sigma_i \otimes \sigma_j$	$\frac{1}{15} (p + p_{\text{swap}}n_2), \quad i, j = 1, 2, 3$

channel,

$$\mathcal{E}'_q(\cdot) \equiv \sum_{\alpha} \left(\sum_{\beta} p_{\alpha\beta} \right) \sigma_{\alpha}(\cdot) \sigma_{\alpha} \quad (\text{B2})$$

as the noise on the output computational qubits (tracing away the routing qubit r). Faulty type-1 SWAPs thus only introduce Pauli errors into the embedded circuit in an independent manner. In particular, the errors on a computational qubit due to its participation in a faulty type-1 SWAP can equivalently be thought of as arising from a fault on that computational qubit location in the embedded circuit, with an added probability $p_{1\text{-SWAP},\alpha} \equiv \sum_{\beta} p_{\alpha\beta}$ of σ_{α} error, in accordance with Eq. (B2). For depolarizing noise on the SWAP gate, $p_{1\text{-SWAP},\alpha} = 4p/15$, and the total additional probability of error is $\sum_{\alpha=1,2,3} p_{1\text{-SWAP},\alpha} = 4p/5$.

We thus have a more concrete picture in this Pauli-noise situation, compared with the general scenario of Theorem 1 in the main text, of how the errors due to SWAP gates can be attributed to faults in the computational qubits. Consider a swap sequence between two consecutive computational layers. Every time a computational qubit participates in a type-1 SWAP in that swap sequence, it acquires an additional $p_{1\text{-SWAP},\alpha}$ for the probability of a σ_{α} error. The errors from faulty type-2 SWAPs are lumped together with the associated (two-qubit) computational gate at the end of the swap sequence. Note that, since Pauli errors only either commute or anticommute, the order in which the Pauli errors occur during the swap sequence does not matter; we need only keep track of the accumulated probability of each kind of Pauli error. The exact effective noise model thus depends on the details of the full routing schedule, and can be very complicated in general.

For our surface-code examples under depolarizing noise, however, we can derive a simpler, albeit, approximate, description. We first observe that, for the SE circuits, all multiqubit computational gates are two-qubit CNOTs. The CNOTs in each SE circuit occur in four computational layers, and every (abstract or computational) qubit, apart from the boundary ones, participates in exactly one CNOT in each layer. Between each pair of CNOT layers, we insert swap sequences to route together any pair of unconnected computational qubits that participate in the same CNOT. Type-2 SWAPs are hence allowed only between each such pair of computational qubits. For a particular long-distance CNOT, suppose the swap sequence needed to move a pair of computational qubits q_1, q_2 together has $n_{1,1}$ and $n_{1,2}$ type-1 SWAPs on q_1 and q_2 , respectively, and n_2 type-2 SWAPs. Then, the effective noise of that CNOT in the embedded circuit can be described as given in Table I. Here, we have taken into account the

added noise from the swap sequence, as well as the original noise on the computational CNOT itself. The SWAP gates are assumed to be subjected to depolarizing noise with parameter $p_{\text{swap}}[\sim O(p)]$, anticipating that the SWAPs are possibly not native, but are made up of $O(1)$ primitive gates.

From Table I, we see that the effective noise channel for each CNOT (or, more generally, a two-qubit gate) in the embedded circuit is an asymmetric Pauli channel, with coefficients for the different Pauli errors that depend on the details of the routing schedule; the asymmetry arises from type-1 SWAPs. Nevertheless, one might hope for an approximate summary of the behavior by a single effective noise parameter p'_{eff} (where the prime on p'_{eff} distinguishes it from the best-fit p_{eff} used in the main text using spacetime averages of the routing schedule parameters and ignoring the asymmetry). Specifically, we regard the probability of no error (the $\mathbb{1} \otimes \mathbb{1}$ term in Table I) as the $p_0 \equiv 1 - p'_{\text{eff}}$ parameter for a depolarizing channel, replacing n_2 and the sum of $n_{1,1}$ and $n_{1,2}$ by their averages (indicated by a bar) over all qubits and layers for the entire routing schedule:

$$p'_{\text{eff}} \equiv p + p_{\text{swap}}[\frac{4}{5}\bar{n}_1 + \bar{n}_2], \quad (\text{B3})$$

where $\bar{n}_1 \equiv \bar{n}_{1,1} + \bar{n}_{1,2}$ counts the average number of type-1 SWAPs experienced by a computational qubit per layer. While the single parameter p'_{eff} cannot capture the asymmetry present in the actual effective noise channel, for our examples of heavy-hexagonal and hexagonal lattices, it nevertheless provides a good summary—as confirmed by our simulations—of the noise deterioration introduced by our routing schedule, as we explain below.

In our simulations, each SWAP is composed from three CNOTs, with a two-qubit depolarizing noise (with strength p) on each CNOT. The noise channel associated with each SWAP can be easily shown to still be depolarizing noise, to linear order in p , but with the noise strength $p_{\text{swap}} = 3p$. Furthermore, for the hexagonal-lattice example, where the optimal swap schedule employs only a single type-2 SWAP, that SWAP is always adjacent to a computational CNOT. Under the three-CNOT decomposition of the type-2 SWAP, this gives four CNOTs in a row such that a pair of CNOTs always cancels out, leaving a remaining pair of CNOTs to implement both the type-2 SWAP and the computational CNOT (this cancellation argument applies generally for all routing schedules, when type-2 SWAPs are decomposed in terms of CNOTs). Allocating the noise from the computational gate to one of the two CNOTs, and the noise for the SWAP gate to the other, we can set $p_{\text{swap}} = p$ in this case.

Now, for the hexagonal case, we have $\bar{n}_1 = 0$ and $\bar{n}_2 = 1/4$, with the division by 4 accounting for the average over the four CNOT computational layers. For the heavy-hexagonal case, we have $\bar{n}_1 = 3.5/4$ and $\bar{n}_2 = 0$, with 3.5 being the average number of type-1 SWAPs per SE round (with its four computational CNOTs) as stated in the main text (see Appendix B 4). Then, we have, for our two examples, the following p'_{eff} values:

$$\begin{aligned} \text{heavy-hex.: } p'_{\text{eff}} &= p + (3p)[\frac{4}{5} \times 3.5/4 + 0] = 3.1p, \\ \text{hex.: } p'_{\text{eff}} &= p + p[0 + 1/4] = 1.25p. \end{aligned} \quad (\text{B4})$$

These come close to the best-fit numerical p_{eff} values given in the main text, and Eq. (B3) can be used as a first estimate of how the noise strength gets rescaled by the EPP routing schedule, and the corresponding threshold deterioration.

3. Syndrome decoding for embedded circuits

Here, we describe how the error mechanisms of embedded circuits can be incorporated into the surface-code syndrome decoding. As discussed in the main text, the SWAP gates in the physical circuit can introduce correlated errors in addition to existing errors in the abstract circuit. These correlated errors, while not altering the fault-tolerance properties of the code, can affect the accuracy of the decoder. This is particularly so since different decoders for surface codes compete largely in the space where there are more errors than guaranteed correctable by the code distance, and hence beyond the fault-tolerance (and EPP) considerations. One expects to be able to reduce the logical error rates by making use of error-correlation information in the decoder. We explain below how to incorporate that into the matching graph of an MWPM surface-code decoder; the ideas can be extended to other codes decoded in a similar manner.

We consider a distance- d surface code that corrects up to $t = \lfloor d/2 \rfloor$ errors. We assume that it has been embedded into the hardware using an EPP routing schedule. The decoder sees only the embedded circuit, not the physical circuit; in particular, the matching graph—which carries the information about where single errors can occur in the circuit and with what weight (probability), and is used by the MWPM algorithm to decide on the allowed matchings and corresponding weights—contains only nodes and edges that represent data and ancilla qubits of the embedded circuit. The matching graph is constructed as follows:

(1) Begin with the standard matching graph [26,44] for the abstract circuit, to capture the basic error mechanisms present even without routing.

(2) Type-1 and type-2 SWAPs that fail independently mimic existing error mechanisms of the abstract circuit, and thus already exist as edges in the matching graph. Incorporate these additional sources of errors by locally updating their corresponding edge weights, which yields noise strength that is additive to the leading order.

(3) Type-1 SWAPs that connect multiple computational qubits to the same routing qubit can generally lead to correlated errors. These can be incorporated by decomposing them in terms of their constituent single-qubit error mechanisms. Explicitly, all correlated error mechanisms of weight $s \leq t$ are identified, which arise whenever the same routing qubit is involved in s type-1 SWAPs. For each of them, trace out the routing qubits to yield a spatiotemporally correlated error mechanism involving s locations. The noise strength of this error mechanism is simultaneously added to its constituent single-qubit error mechanisms, each of which already exists as an edge in the matching graph.

This procedure yields a matching graph with the same connectivity as that of the surface code, with updated edge weights, which can be used for syndrome decoding via the MWPM algorithm.

TABLE II. Five-SWAP-layer routing schedule for the heavy-hexagonal lattice example. The C_i columns are the computational CNOT layers, while the S_i columns are the swap layers. For the S_i columns, the symbols u , d , l , r , and i indicate swaps of the qubit (row) with its neighboring qubit to move up, down, left, right, or idle, respectively. For the C_i columns, the same symbols denote the direction of the qubit interacting with it through a CNOT gate (i.e., the orientations of the colored edges in Fig. 5).

	C_1	S_1	C_2	S_2	S_3	C_3	S_4	S_5	C_4
X data	d	u	r	l	l	l	u	i	u
Z data	d	u	l	i	i	r	l	l	u
X ancilla	u	l	l	i	i	l	l	d	d
Z ancilla	u	r	r	l	l	r	i	l	d

We note that the correlated errors can be directly included as hyperedges (i.e., edges joining more than two nodes) in the matching graph, instead of decomposing them in terms of independent error mechanisms. This improves the performance of decoding and thus lowers logical error probabilities and threshold requirements, at the expense of transforming the matching graph into a hypergraph that requires a hypergraph matching algorithm [39,40]. Specifically, correlated errors involving s qubits or fault locations appear as order- $2s$ hyperedges (i.e., edges joining $2s$ nodes) involving existing nodes in the matching graph that can always be formed by the union of nodes involved in s existing edges (each corresponding to an existing single-qubit error mechanism that a faulty type-1 SWAP mimics). In our simulations, we adopt the simpler approach of working only with MWPM and a matching graph consisting of graphlike edges (i.e., edges only join two nodes).

4. Further details of the heavy-hexagonal example

Here, we describe the routing schedule used in our heavy-hexagonal lattice example, as found by the greedy distance-minimizing routing algorithm described earlier. The precise moves of the EPP routing schedule are given in Table II, with intermediate embeddings during CNOT layers illustrated in Figs. 5(a)–5(d). It consists of five SWAP layers interspersed between the four computational CNOT layers of

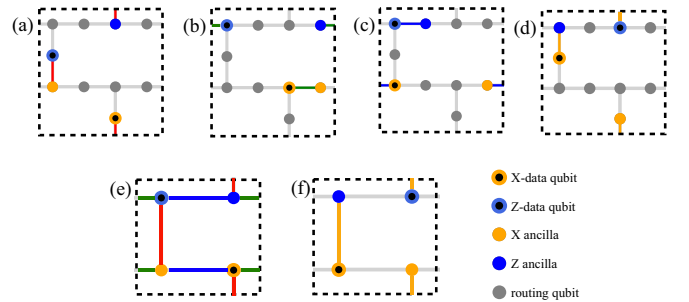


FIG. 5. Intermediate embeddings for the heavy-hexagonal [panels (a)–(d)] and hexagonal [panels (e) and (f)] lattice EPP schedules. Red, green, blue, and yellow colored edges denote computational operations corresponding to the CNOT layers C_1 , C_2 , C_3 , and C_4 , respectively.

TABLE III. One-SWAP-layer routing schedule for the hexagonal lattice. Notations follow those of Table II.

	C_1	C_2	C_3	S_1	C_4
X data	d	r	l	r	u
Z data	d	l	r	r	u
X ancilla	u	l	r	r	d
Z ancilla	u	r	l	r	d

one round of syndrome extraction. At the end of one full round, the computational qubits are displaced one unit cell diagonally from their initial positions, which are reversed in the next round by reversing the ordering and directions of the swap moves.

Each data or ancillary qubit experiences, on average, $\bar{n}_1 = 3.5/4$ and $\bar{n}_2 = 0$ type-1 and type-2 SWAPs, respectively, per layer. Decomposing each SWAP gate as three CNOT gates, we find that it can be executed within 19 time steps. Notably, certain routing qubits in this schedule are involved in type-1 SWAPs with computational qubits that are not connected in the interaction graph, giving rise to the possibility of spatiotemporally correlated errors as mentioned in the main text.

5. Further details of the hexagonal example

Here, we describe the routing schedule used in our hexagonal (or honeycomb) lattice example, as found by the greedy distance-minimizing routing algorithm described earlier. The precise moves of the EPP routing schedule are given in Table III, with intermediate embeddings during CNOT layers illustrated in Figs. 5(e) and 5(f). It consists of a single SWAP layer between the final pair of interaction layers, consisting exclusively of type-2 SWAP gates. There are also no routing qubits involved—Every physical qubit is mapped to a data or ancilla qubit of the surface code. At the end of one full round, the computational qubits are displaced one unit cell horizontally from their initial positions, which are reversed in the next round by reversing the ordering and directions of the swap moves.

Numerical simulations of the logical error probabilities are given in Fig. 6. Each data or ancillary qubit experiences $\bar{n}_2 = 1/4$ SWAP gates on average, per layer. Decomposing each SWAP gate as three CNOT gates and simplifying the circuit via cancellations of two CNOT gates as described earlier, we find that it can be executed within five time steps. Notably, this schedule saturates the naive lower bound of one SWAP layer, necessary when embedding a degree-4 interaction graph onto a degree-3 device graph, and is therefore optimal in the number of swap layers. In addition, since no routing qubits are involved, it is also optimal in terms of the number of physical qubits required to embed the surface code onto the hexagonal lattice.

6. Details on the simulations for SABRE protocol

Here, we provide more details on the numerical simulations for the standard circuit layout and swap-based routing protocol SABRE [17,35] implemented through Qiskit. Similar to the basic algorithm outlined in Appendix A, the SABRE algorithm selects the subset of qubits that is most densely

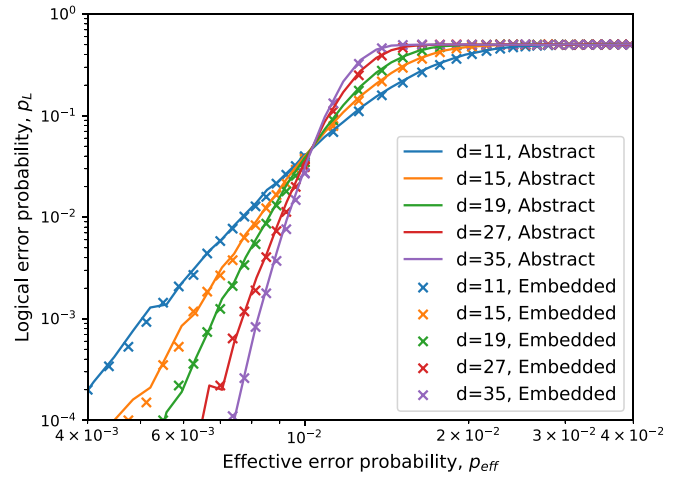


FIG. 6. The logical error probability p_L vs the effective error probability p_{eff} for the example of the surface code embedded onto the hexagonal lattice under depolarizing noise with error probability p . For the benchmark noisy abstract circuit, $p_{\text{eff}} = p$; for the physical (embedded) circuit, $p_{\text{eff}} = 1.25p$ obtained by fitting both curves for $d = 35$ below the threshold value.

connected within the connectivity graph of a device, and subsequently inserts SWAP gates to match the interaction graph of the abstract circuit in a heuristic, depth-minimizing manner. For convenience, we fix the number of qubits selected to be the same as the number of qubits in the abstract circuit, for qubit minimization, a standard compilation choice.

Unlike in our EPP routing scheme, the routed circuits from SABRE generically contain (noisy) SWAP gates between abstract qubits that are unconnected in the interaction graph of the abstract circuit. These noisy SWAP gates are taken into account in the matching graph for MWPM, leading to the inclusion of edges not originally present in the matching graph of the standard (abstract) surface code. Subsequently, the

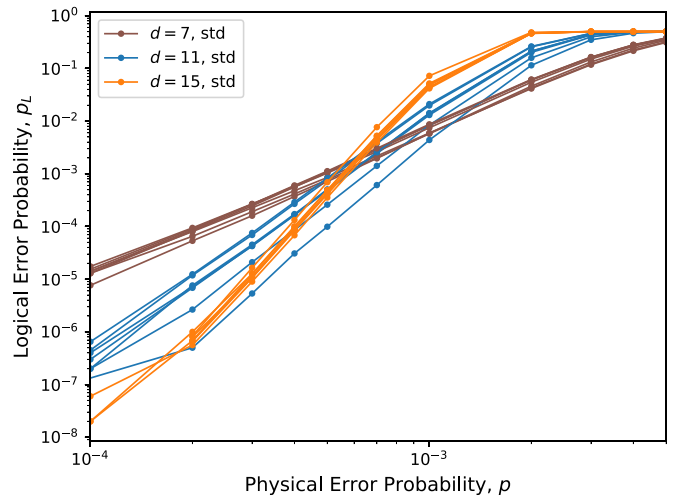


FIG. 7. Logical error probability p_L vs physical error probability p for different realizations of the SABRE-routed surface-code circuits, for code distances $d = 7, 11$, and 15 . Each curve gives the p_L vs p line for a single realization (the best out of many trials) of the surface code routed by SABRE.

resulting syndrome measurements are decoded, and the logical error probabilities p_L are obtained for a range of physical error probabilities p .

Due to the inherently stochastic nature of the SABRE algorithm, the output embedded circuit is not always the same. We thus repeat the SABRE algorithm protocol with seven realizations for each code distance d ; each realization has minimal depth among a total of 200 trials of initial subset selection and 500 subsequent routing trials. The results for all realizations

are given in Fig. 7; the lines in Fig. 3 are the averages of the seven realizations for each d . Fitting the (averaged) log-log curve at low error rates and computing the ratio r_d between the gradients for the SABRE circuits versus the EPP circuits (which have gradients identical to the abstract surface code due to Theorem 1, and as demonstrated in Fig. 2), we obtain $(r_7, r_{11}, r_{15}) = (0.63, 0.65, 0.74)$. The smaller-than-1 ratios confirm the reduction in the effective code distances due to the non-fault-tolerant SABRE embedding.

-
- [1] E. Knill, R. Laflamme, and W. H. Zurek, Resilient quantum computation, *Science* **279**, 342 (1998).
 - [2] J. Preskill, Reliable quantum computers, *Proc. R. Soc. Lond. Ser. A* **454**, 385 (1998).
 - [3] D. Aharonov and M. Ben-Or, Fault-tolerant quantum computation with constant error rate, *SIAM J. Comput.* **38**, 1207 (2008).
 - [4] P. Aliferis, D. Gottesman, and J. Preskill, Quantum accuracy threshold for concatenated distance-3 codes, *Quant. Inf. Comput.* **6**, 97 (2006).
 - [5] J. Koch, M. Y. Terri, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, Charge-insensitive qubit design derived from the Cooper pair box, *Phys. Rev. A* **76**, 042319 (2007).
 - [6] S. Bravyi, O. Dial, J. M. Gambetta, D. Gil, and Z. Nazario, The future of quantum computing with superconducting qubits, *J. Appl. Phys.* **132**, 160902 (2022).
 - [7] M. Dupont, B. Evert, M. J. Hodson, B. Sundar, S. Jeffrey, Y. Yamaguchi, D. Feng, F. B. Maciejewski, S. Hadfield, M. S. Alam, Z. Wang, S. Grabbe, P. A. Lott, E. G. Rieffel, D. Venturelli, and M. J. Reagor, Quantum-enhanced greedy combinatorial optimization solver, *Sci. Adv.* **9**, eadi0487 (2023).
 - [8] B. Hetényi and J. R. Wootton, Tailoring quantum error correction to spin qubits, *Phys. Rev. A* **109**, 032433 (2024).
 - [9] A. Wu, G. Li, H. Zhang, G. G. Guerreschi, Y. Ding, and Y. Xie, Mapping surface code to superconducting quantum processors, [arXiv:2111.13729](https://arxiv.org/abs/2111.13729).
 - [10] D. Kremer, V. Villar, H. Paik, I. Duran, I. Faro, and J. Cruz-Benito, Practical and efficient quantum circuit synthesis and transpiling with reinforcement learning, [arXiv:2405.13196](https://arxiv.org/abs/2405.13196).
 - [11] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, Qubit allocation, in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, CGO 2018 (Association for Computing Machinery, New York, NY, 2018), pp. 113–125.
 - [12] S. H. Choe and R. Koenig, How to fault-tolerantly realize any quantum circuit with local operations, *PRX Quantum* **6**, 010357 (2025).
 - [13] J. M. Pino, J. M. Dreiling, C. Figgatt, J. P. Gaebler, S. A. Moses, M. S. Allman, C. H. Baldwin, M. Foss-Feig, D. Hayes, K. Mayer, C. Ryan-Anderson, and B. Neyenhuis, Demonstration of the trapped-ion quantum CCD computer architecture, *Nature (London)* **592**, 209 (2021).
 - [14] D. Bluvstein *et al.*, Logical quantum processor based on reconfigurable atom arrays, *Nature (London)* **626**, 58 (2024).
 - [15] Y. Stade, L. Schmid, L. Burgholzer, and R. Wille, An abstract model and efficient routing for logical entangling gates on zoned neutral atom architectures, in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)* (Montreal, QC, Canada, 2024), pp. 784–795.
 - [16] W. Finigan, M. Cubeddu, T. Lively, J. Flick, and P. Narang, Qubit allocation for noisy intermediate-scale quantum computers, [arXiv:1810.08291](https://arxiv.org/abs/1810.08291).
 - [17] G. Li, Y. Ding, and Y. Xie, Tackling the qubit mapping problem for NISQ-era quantum devices, in *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, ASPLOS '19 (Association for Computing Machinery, New York, NY, 2019), pp. 1001–1014.
 - [18] A. M. Childs, E. Schoute, and C. M. Unsal, Circuit transformations for quantum architectures, in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 135 (TQC, 2019), pp. 3:1–3:24.
 - [19] G. Nannicini, L. S. Bishop, O. Günlük, and P. Jurcevic, Optimal qubit assignment and routing via integer programming, *ACM Trans. Quantum Comput.* **4**, 1 (2023).
 - [20] T. Ito, N. Kakimura, N. Kamiyama, Y. Kobayashi, and Y. Okamoto, Algorithmic theory of qubit routing, [arXiv:2305.02059](https://arxiv.org/abs/2305.02059).
 - [21] P. Aliferis and J. Preskill, Fault-tolerant quantum computation against biased noise, *Phys. Rev. A* **78**, 052331 (2008).
 - [22] D. K. Tuckett, S. D. Bartlett, S. T. Flammia, and B. J. Brown, Fault-tolerant thresholds for the surface code in excess of 5% under biased noise, *Phys. Rev. Lett.* **124**, 130501 (2020).
 - [23] D. Gottesman, Fault-tolerant quantum computation with constant overhead, [arXiv:1310.2984](https://arxiv.org/abs/1310.2984).
 - [24] N. Delfosse and B. W. Reichardt, Short Shor-style syndrome sequences, [arXiv:2008.05051](https://arxiv.org/abs/2008.05051).
 - [25] T. Tansuwannont, B. Pato, and K. R. Brown, Adaptive syndrome measurements for Shor-style error correction, *Quantum* **7**, 1075 (2023).
 - [26] 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).
 - [27] Y. Tomita and K. M. Svore, Low-distance surface codes under realistic quantum noise, *Phys. Rev. A* **90**, 062320 (2014).
 - [28] See Supplemental Material at <http://link.aps.org/supplemental/10.1103/crv2-99qd> for animation illustrating action of swap schedules.
 - [29] C. Gidney, Stim: A fast stabilizer circuit simulator, *Quantum* **5**, 497 (2021).
 - [30] O. Higgott, PyMatching: A Python package for decoding quantum codes with minimum-weight perfect matching, *ACM Trans. Quantum Comput.* **3**, 1 (2022).

- [31] O. Higgott and C. Gidney, Sparse Blossom: Correcting a million errors per core second with minimum-weight matching, [Quantum](#) **9**, 1600 (2023).
- [32] M. McEwen, D. Bacon, and C. Gidney, Relaxing hardware requirements for surface code circuits using time-dynamics, [Quantum](#) **7**, 1172 (2023).
- [33] C. Benito, E. López, B. Peropadre, and A. Bermudez, Comparative study of quantum error correction strategies for the heavy-hexagonal lattice, [Quantum](#) **9**, 1623 (2024).
- [34] B. Hetényi and J. R. Wootton, Creating entangled logical qubits in the heavy-hex lattice with topological codes, [PRX Quantum](#) **5**, 040334 (2024).
- [35] H. Zou, M. Treinish, K. Hartman, A. Ivrii, and J. Lishman, Lightsabre: A lightweight and enhanced Sabre algorithm, [arXiv:2409.08368](#).
- [36] B. M. Terhal and G. Burkard, Fault-tolerant quantum computation for local non-Markovian noise, [Phys. Rev. A](#) **71**, 012336 (2005).
- [37] D. Aharonov, A. Kitaev, and J. Preskill, Fault-tolerant quantum computation with long-range correlated noise, [Phys. Rev. Lett.](#) **96**, 050504 (2006).
- [38] H. K. Ng and J. Preskill, Fault-tolerant quantum computation versus Gaussian noise, [Phys. Rev. A](#) **79**, 032318 (2009).
- [39] O. Higgott, T. C. Bohdanowicz, A. Kubica, S. T. Flammia, and E. T. Campbell, Improved decoding of circuit noise and fragile boundaries of tailored surface codes, [Phys. Rev. X](#) **13**, 031007 (2023).
- [40] E. H. Chen, T. J. Yoder, Y. Kim, N. Sundaresan, S. Srinivasan, M. Li, A. D. Córcoles, A. W. Cross, and M. Takita, Calibrated decoders for experimental quantum error correction, [Phys. Rev. Lett.](#) **128**, 110504 (2022).
- [41] A. Bapat, A. M. Childs, A. V. Gorshkov, and E. Schoute, Advantages and limitations of quantum routing, [PRX Quantum](#) **4**, 010313 (2023).
- [42] C.-C. Lin, S. Sur-Kolay, and N. K. Jha, Paqcs: Physical design-aware fault-tolerant quantum circuit synthesis, in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (IEEE, 2015), Vol. 23, pp. 1221–1234.
- [43] L. Lao, B. van Wee, I. Ashraf, J. van Someren, N. Khammassi, K. Bertels, and C. G. Almudever, Mapping of lattice surgery-based quantum circuits on surface code architectures, [Quantum Sci. Technol.](#) **4**, 015005 (2018).
- [44] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Topological quantum memory, [J. Math. Phys.](#) **43**, 4452 (2002).