PRISM: A Logical–Physical Boundary Compiler for Dual-Rail Qubits

Evan McKinney [*†‡], Kun Liu [*†], Harshvardhan K. Babla [†‡], Robert J. Schoelkopf [†‡], Yongshan Ding [*†‡]
[*]Department of Computer Science, Yale University, New Haven, CT, USA
[†]Yale Quantum Institute, Yale University, New Haven, CT, USA
[‡]Department of Applied Physics, Yale University, New Haven, CT, USA
evan.mckinney@yale.edu, yongshan.ding@yale.edu

***Abstract*—Quantum computers can encode each logical qubit across multiple physical degrees of freedom so that errors on the physical qubits can be detected or corrected. A compiler must then translate each logical gate into native instructions, and one gate can have several equivalent realizations that interact through different wires. Routing moves those wires, so a realization chosen before routing can force SWAPs that another would avoid, making code expansion and physical routing a phase-ordering problem. PRISM therefore keeps each gate's realizations open across the logical–physical boundary and commits to one only when routing reaches a layout that can execute it. We formalize which physical circuits implement a gate, which respect the fault model, and which the compiler keeps, and define when a gate can execute under a layout. For dual-rail cavity qubits, logical CR_z has four loss-admissible realizations, one per rail pairing, and its logical availability graph is exactly the simple quotient of the occupied device graph by the rail pairs. Across 25 workloads with 7 to 25 logical qubits on three device graphs, PRISM reduces routing SWAPs by 20.4–30.8% and physical depth by 8.8–10.1% relative to a greedy search that fixes each realization before routing. On the grid and hexagonal lattice, the SWAP reductions hold through 32 logical qubits. By shortening routes, PRISM also reduces photon loss on device models and therefore needs 44.3% fewer attempts per retained shot at 16 logical qubits, as a geometric mean over workloads and device graphs, and it lowers decoded logical error in a three-round Steane memory under skip-gate leakage.**

***Index Terms*—quantum circuit, quantum compilation, dual-rail qubits**

I. Introduction

Choosing a physical circuit for an encoded quantum gate before routing can force SWAPs that an equivalent circuit would avoid. An encoding spreads a logical qubit across several physical degrees of freedom [1], and one logical gate can have equivalent physical realizations that interact through different wires. Lowering translates logical operations into physical instructions through a hierarchy of compiler representations [2], and physical routing moves wires so that each interaction lands on a device edge [3]. Expanding each gate into one circuit discards the alternatives before routing can evaluate them, and routing first on a fixed logical coupling graph hides the wires that distinguish them. Code expansion and routing therefore form a phase-ordering problem, like those studied among quantum optimization passes [4].

We introduce PRISM (Physical-Realization Instruction-Set Mapping), which keeps each gate's realization set open across the logical–physical boundary and commits to one realization only when the router can execute it. Its boundary intermediate representation (IR) exposes the physical wires while it retains the alternative circuits (Fig. 1). During SABRE-style routing [5], PRISM scores the alternatives under the current layout and uses their costs to guide SWAP insertion. Routing changes which realization is available, so commitment should wait for the layout that routing reaches.

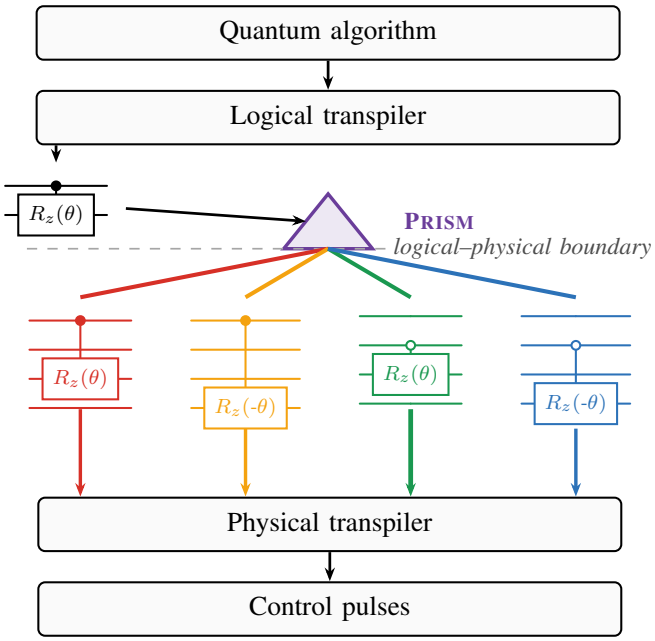


Fig. 1: PRISM carries the four realizations of dual-rail $\mathrm{CR}_z(\theta)$, one per rail pairing, across the logical–physical boundary for layout-aware selection during physical routing. Fig. 5 shows the emitted circuits.

A logical qubit occupies the single-excitation subspace $\mathrm{span}\{|01\rangle, |10\rangle\}$ of two cavity modes, or rails, and photon loss leaves a detectable vacuum state [6]–[8]. With local Z corrections, a controlled phase between either rail of one operand and either rail of the other implements the same logical CR_z [9], [10]. Because the four realizations need different device edges, the layout determines which of them can execute without routing. In Fig. 2, only the green pairing

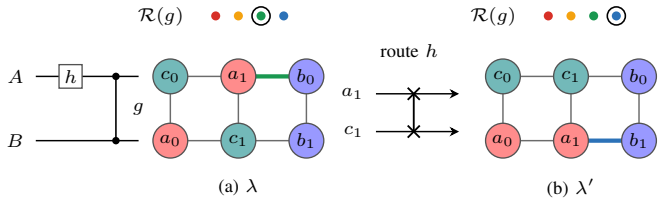


Fig. 2: The program applies h, a logical R_X on A, before the pending CR_z g on A and B. Under layout λ, only the green pairing (a_1, b_0) of g's four realizations (dots) is available. Routing h makes the rails of A adjacent by exchanging a_1 and c_1, a rail of a third logical qubit C, after which only the blue pairing (a_1, b_1) is available. A compiler that chooses the green realization under λ must insert SWAPs to execute it under λ'.

of the pending CR_z is adjacent under the initial layout. The R_X that precedes it needs its operand's rails adjacent, and the SWAP that achieves this breaks the green pairing and creates the blue one. A compiler that has already committed to green must insert SWAPs, whereas PRISM selects blue under the layout that routing reaches.

The formalism separates what makes these alternatives interchangeable from what makes them executable. Completions are the physical unitaries with the required action on encoded inputs, admissible completions also satisfy the fault model, and the compiler retains a finite set of realization circuits. A logical gate is available under a layout when one of its realizations can execute without routing. For dual-rail CR_z, the resulting logical availability graph is exactly the simple quotient of the occupied device graph by the rail pairs.

We compare PRISM with Greedy, which shares its boundary IR and physical router but fixes each realization at the start of each routing pass, and with Logical-SABRE, which routes on a fixed logical coupling graph before lowering. An ablation measures the effects of choosing a realization under the current layout and of delaying commitment until execution. This paper makes the following contributions.

- We implement PRISM, which carries realization sets through lowering and commits during routing. Across 25 workloads with 7 to 25 logical qubits on three device graphs, it reduces Greedy's routing SWAPs by 20.4–30.8% and physical depth by 8.8–10.1%, with SWAP reductions through 32 logical qubits on the grid and hexagonal lattice.
- We separate logically correct completions, admissible completions, and retained realization sets, and show that the four loss-admissible dual-rail CR_z realizations induce the simple quotient of the occupied device graph by the rail pairs.
- On modeled dual-rail cavity hardware, PRISM needs 44.3% fewer attempts per retained shot at 16 logical qubits as a geometric mean over workloads and device graphs, and in a three-round Steane memory under skip-gate leakage it lowers decoded logical error by 13.0–19.6% with flags at readout and by 22.6–27.7% with compiled erasure checks.

II. BACKGROUND

A. Compilation, Routing, and the Phase-Ordering Problem

General-purpose quantum compilers run synthesis and routing passes on circuits with explicit wires [11]–[13]. Physical routing maps these wires to a device graph and inserts SWAPs for unsupported interactions [3]. SABRE scores candidate SWAPs by ready-gate distances and a lookahead set [5].

Multi-level compiler frameworks represent changes in instruction and wire granularity across compilation stages [2], [14]–[16]. Realization choice and routing constrain each other, creating a phase-ordering problem [17].

B. Dual-Rail Qubits

A dual-rail qubit uses two physical modes, or rails, with logical states encoded as $|0_{\mathrm{L}}\rangle = |01\rangle$ and $|1_{\mathrm{L}}\rangle = |10\rangle$. Together these span the two-level code space $\mathcal{C} = \mathrm{span}\{|01\rangle, |10\rangle\}$, and its complement $\mathcal{C}^{\perp} = \mathrm{span}\{|00\rangle, |11\rangle\}$ defines the erasure space. The encoding can be summarized by the isometry

$$V = |01\rangle\langle 0| + |10\rangle\langle 1|, \qquad V^{\dagger}V = I_2, \quad VV^{\dagger} = P_{\mathcal{C}}. \tag{1}$$

Under the zero-temperature amplitude-damping model, photon loss maps both logical basis states to the vacuum $|00\rangle$, and erasure detection has been demonstrated in dual-rail cavities, dual-rail transmons, and neutral atoms [7], [18]–[20]. Mid-circuit erasure checks must measure the joint photon occupation of both rails without revealing the logical state [6], [21]. At the end of a circuit, each rail can be measured separately [7]. Outcomes 01 and 10 report logical values, while 00 and 11 flag an erasure.

Postselection discards any shot with a detected loss, so erasure probability sets the shot-acceptance rate [22]. An outer quantum error correction (QEC) code can instead use erasure information to correct the loss [23]–[26], since a code corrects twice as many erasures as Pauli errors [27] and erasure conversion improves thresholds and overheads [28], [29].

A surface code with erasure-flagging readout can recover erasure-like error scaling without mid-circuit checks [30]. Section V-E evaluates Steane-code memory circuits with flags at readout and with compiled erasure checks.

The circuit-QED architecture model in this paper uses long-lived cavity modes as rails and an ancilla transmon for joint erasure checks [6]–[8]. Parametric beamsplitters transfer excitations between cavities, including a full cavity–cavity exchange [31], [32]. The SWAP–wait–SWAP (SWS) sequence moves a selected control-rail excitation into a transmon coupler, accumulates a phase conditional on one target rail, and returns it [10]. After local Z corrections, it implements the physical CR_z instruction of Section III-D.

Photon loss dominates errors in dual-rail cavity hardware, occurring during idle periods about six times more often than phase errors and at least 150 times more often than bit flips, and erasure checks identify more than 99% of losses [7]. The SWS gate has approximately 0.5% erasure probability, at most 0.1% residual dephasing, and a bit-flip probability of a few parts per million [10], so the residual unflagged errors are mostly phase errors.

The routing model represents each cavity mode as a vertex and each supported two-mode interaction as an edge, and assumes that every edge supports the SWS interaction and a tunable beamsplitter that implements each routing SWAP as a full cavity–cavity exchange.

III. From Logical Gates to Logical Availability

Whether an encoded gate can execute without routing depends on its retained circuits and the current layout. For the dual-rail cavity instruction set architecture (ISA), logical availability is a quotient of the occupied device graph by the rail pairs.

A. Completions, Admissibility, and Realizations

Let V be the isometry for the encoded operands on which a logical gate g acts, including any fixed ancilla input state. A physical unitary U is a *completion* of g when

$$UV = Vg. \tag{2}$$

The corresponding completion set is

$$\mathcal{P}(g) = \{U : UV = Vg\}. \tag{3}$$

The *admissible completion set* $\mathcal{A}(g) \subseteq \mathcal{P}(g)$ contains completions satisfying the fault model's constraints on the full target-level unitary [9], [33], [34].

The compiler retains a finite *realization set* $\mathcal{R}(g)$ of circuits, chosen for example by synthesis cost [35]. Each realization $r \in \mathcal{R}(g)$ names a complete target-level circuit C_r whose unitary lies in $\mathcal{A}(g)$ and whose implementation obeys the declared circuit-level constraints.

B. Logical Availability

A logical gate is *available* under a layout when at least one of its realizations can execute without routing. Because routing moves the wires that each realization uses, it changes availability even while the device graph stays fixed (Fig. 4).

Each realization r has an *interaction pattern* I_r with one vertex for each wire of C_r and one labeled edge for each interaction between wires. Let G be the device graph, and let a layout λ map the program's physical data wires to vertices in G. An embedding $\phi \in \mathrm{Emb}_\lambda(I_r, G)$ maps the wires of C_r injectively to device vertices. It fixes each data wire w at $\lambda(w)$ and maps each labeled interaction to a supporting device edge. The *availability relation* under the current layout is

$$\mathcal{E}_g(G, \lambda) = \{(r, \phi) : r \in \mathcal{R}(g),\ \phi \in \mathrm{Emb}_\lambda(I_r, G)\}. \tag{4}$$

For a two-operand gate type g, the *logical availability graph* $H_g(G, \lambda)$ has one vertex for each logical qubit and an edge (p, q) when Eq. (4) is nonempty for g on those operands.

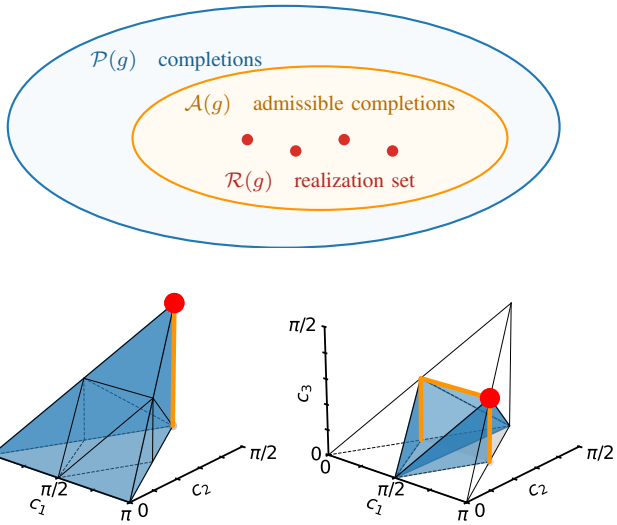


Fig. 3: The upper panel distinguishes the physical completions $\mathcal{P}(g)$, the admissible completions $\mathcal{A}(g)$, and the unitary $U(C_r)$ of each realization r in $\mathcal{R}(g)$, drawn as a red point. The lower panels show these sets in the Weyl chamber for $U_L = X$ (left) and $U_L = \sqrt{X}$ (right), where the completions form the blue surface, the admissible family of Eq. (7) forms the orange line, and the realization is the red point.

C. Dual-Rail Single-Qubit Completions

Up to global phase, a logical single-qubit gate has a completion that acts on each rail separately exactly when it is $R_Z(\theta)$ or $R_Z(\theta)X$. Of these gates, only $R_Z(\theta)$ has an admissible completion that acts on each rail separately. In the dual-rail cavity ISA, R_Z therefore executes on a single rail while the two rails of a qubit are separated, whereas R_X requires the two rails to be adjacent, so an upcoming R_X limits how long the router can keep that rail pair separated. We write logical rotations in upper case (R_X, R_Z) and physical rotations in lower case (R_z).

A physical gate that preserves the code and erasure spaces has the block form [36]

$$M(A, B) = \begin{pmatrix} A_{11} & 0 & 0 & A_{12} \\ 0 & B_{11} & B_{12} & 0 \\ 0 & B_{21} & B_{22} & 0 \\ A_{21} & 0 & 0 & A_{22} \end{pmatrix}, \tag{5}$$

where $B = U_L$ acts on $\mathcal{C} = \mathrm{span}\{|01\rangle, |10\rangle\}$ and A acts on $\mathcal{C}^\perp = \mathrm{span}\{|00\rangle, |11\rangle\}$. Figure 3 places the two-rail completions of a logical single-qubit gate in the Weyl chamber [37], [38], which groups two-qubit unitaries by nonlocal action. The origin represents product unitaries. The blue surface for logical X reaches the origin, so a product unitary can implement X. The surface for logical $\sqrt{X}$ does not reach the origin, so every completion of $\sqrt{X}$ entangles the rails. Appendix A derives the surfaces.

It follows from the canonical form of parity-preserving gates [36] that $M(A, B)$ is a product unitary only when A and

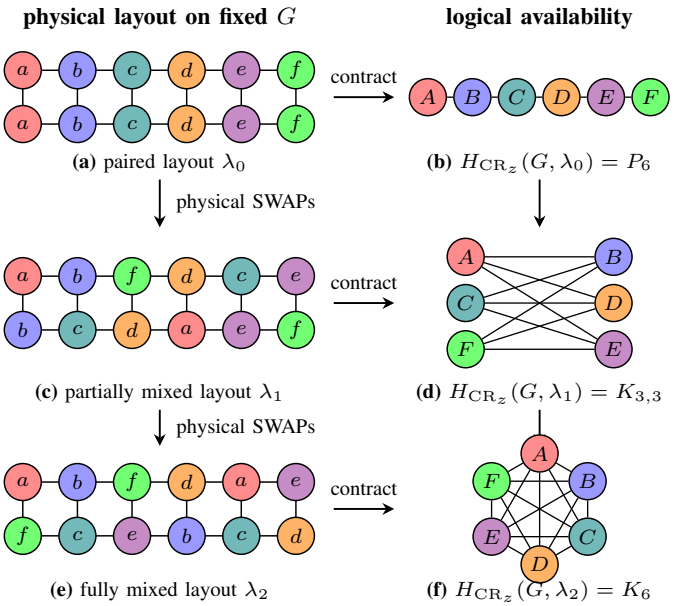


Fig. 4: Panels (a), (c), and (e) show layouts of six dual-rail qubits A to F before and after successive SWAPs, and panels (b), (d), and (f) show their logical availability graphs. Each SWAP interleaves rails of different rail pairs, and the graph grows from P_6 to $K_{3,3}$ to K_6.

B are both diagonal or both anti-diagonal. Thus, up to global phase, the logical gates with product completions are

$$\mathcal{G}_{\mathrm{deloc}} = \{R_Z(\theta)\}_\theta \cup \{R_Z(\theta)X\}_\theta \cong \mathrm{O}(2). \tag{6}$$

This family is non-universal, so a general single-qubit program eventually needs a gate that couples the rails.

For a single dual-rail qubit, the loss model excludes completions that transfer population from the vacuum $|00\rangle$ to $|11\rangle$, because a later photon-loss event can then return the erased state to the code space (seepage) [39]. An off-diagonal A permits this transfer, so an admissible completion must preserve the rail pair's total excitation number, which requires

$$A = e^{i\alpha} R_z(\beta). \tag{7}$$

Varying α moves the completion along the orange line in Fig. 3, and varying β changes the admissible completion without changing its Weyl class. Because Eq. (7) makes A diagonal, an admissible completion is a product unitary only when B is diagonal and $\det A = \det B$, so $R_Z(\theta)$ is the only family in Eq. (6) with admissible product completions.

For logical X, $\mathcal{P}(X)$ contains every $M(A,X)$, while $\mathcal{A}(X)$ retains the completions whose Weyl classes trace the parametric-SWAP edge from iSWAP to SWAP [40]. The dual-rail cavity ISA includes the fractional-SWAP family SWAP^t with $M(I,X^t) = \mathrm{SWAP}^t$, so $\mathcal{R}(X)$ contains the native circuit for the SWAP endpoint $M(I,X)$. More generally, the encoding maps logical $R_X(\theta)$ to the physical fractional-SWAP

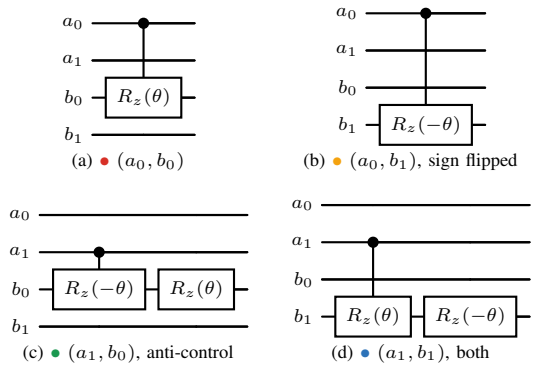


Fig. 5: Each of the four realizations of logical dual-rail $\mathrm{CR}_z(\theta)$ uses one rail pairing, which fixes the rotation sign and the single-rail R_z correction.

rotation $R_{\mathrm{SWAP}}(\theta)$, which couples the two rails, and $\mathcal{R}(R_Z(\theta))$ contains one single-rail rotation.

D. Dual-Rail Two-Qubit Completions

Let $V_{AB} = V_A \otimes V_B$ encode two logical qubits in four rails. A four-rail unitary U completes logical $\mathrm{CR}_z(\theta)$ when

$$UV_{AB} = V_{AB}\mathrm{CR}_z(\theta), \tag{8}$$

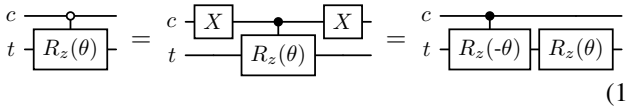
where we use the convention

$$\mathrm{CR}_z(\theta) = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes R_z(\theta). \tag{9}$$

For the loss model considered here, diagonal completions form an admissible family because they preserve the occupation of every rail and cannot return an erased state to the code space. We construct realizations only from this family without characterizing the full admissible set. Let logical qubit A occupy rails (a_0, a_1) and logical qubit B occupy rails (b_0, b_1). If a diagonal completion uses one interaction between selected rails a_i and b_j, its nonlocal action is a controlled phase up to local R_z rotations. The realization set holds four such circuits, one per rail pairing $i, j \in \{0, 1\}$. Each uses one physical CR_z between a_i and b_j and, when $i = 1$, one single-rail R_z. With $s_j = (-1)^j$, the circuits are

$$C_{ij}(\theta) = R_{z,b_j}(\mathbf{1}_{i=1} s_j \theta)\, \mathrm{CR}_{z,a_i b_j}\left((-1)^{i+j}\theta\right). \tag{10}$$

Choosing the other control rail inverts the control condition, while choosing the other target rail reverses the controlled-rotation angle in Eq. (10). An inverted control has the equivalent forms



(11)

The four circuits use the diagonal form on the right, because the two single-rail X gates of the middle form leave the dual-rail code space.

We implement the physical CR_z in Eq. (10) using the SWS sequence, which produces a controlled phase between

two selected cavity modes. Its wait time sets the phase, and calibrated local phase corrections produce CZ [10]. The controlled phase satisfies

$$\mathrm{CP}(\theta) = \mathrm{diag}(1,1,1,e^{i\theta}) = e^{i\theta/4}\big(R_z(\theta/2)\otimes I\big)\mathrm{CR}_z(\theta), \tag{12}$$

where the $R_z(\theta/2)$ rotation acts on the control. We absorb this local phase correction and the calibrated SWS phase corrections into the physical CR_z instruction.

A direct calculation on the four logical basis states gives $C_{ij}(\theta)V_{AB} = V_{AB}\mathrm{CR}_z(\theta)$. The normalized physical CR_z and every explicit gate in C_{ij} are diagonal at the target level, so each circuit preserves both pairs' occupation and is admissible.

The compiler therefore exposes the logical ISA $\{R_X, R_Z, \mathrm{CR}_z\}$, with $R_X(\theta)$ mapped to $R_{\mathrm{SWAP}}(\theta)$, $R_Z(\theta)$ to a single-rail rotation, and $\mathrm{CR}_z(\theta)$ to the four realizations of Eq. (10).

E. Dual-Rail Logical Availability as a Quotient

With the four realizations of Eq. (10), logical CR_z is available on two logical qubits exactly when a device edge connects a rail of one to a rail of the other. The logical availability graph is therefore the simple quotient of the occupied device graph by the rail pairs. Let $S_q(\lambda)$ contain the two vertices occupied by logical qubit q, and let $G_\lambda = G[\bigcup_q S_q(\lambda)]$ be the subgraph induced by the occupied vertices. The sets $S_q(\lambda)$ form the partition Π_λ, and the quotient multigraph $Q(G_\lambda, \Pi_\lambda)$ has one vertex for each logical qubit, an edge for each device edge between two rail pairs, and a self-loop for each device edge within one rail pair. Removing self-loops and merging parallel edges gives $H_{\mathrm{CR}_z}(G,\lambda)$. This construction holds only for realizations with one interaction. For richer realization sets, such as the aligned logical SWAP of Section IV-C (Fig. 8a), which needs two disjoint edges, Eq. (4) still defines availability but the quotient does not compute it.

Representing the two rails as one logical vertex combines their physical neighborhoods, so on a device of maximum degree Δ the logical vertex has at most 2Δ distinct neighbors. An edge between the rails becomes a discarded self-loop, so separating the rails can expose more external neighbors. In Fig. 4, the paired layout λ_0 gives the path P_6 in panels (a) and (b). The SWAPs shown produce the logical availability graphs $K_{3,3}$ in panels (c) and (d) and then K_6 in panels (e) and (f).

IV. Compiler Design

PRISM carries each logical gate across the logical–physical boundary as a set of realizations and commits to one when routing reaches a layout in which that realization can execute. Greedy commits before each router call, and Logical-SABRE after routing on a fixed logical coupling graph (Fig. 6a).

A. Boundary Intermediate Representation

LOWEREXPANSION produces the boundary IR that PRISM and Greedy share. It replaces each logical qubit with the wires that encode it and each logical gate with a LOWEREDOP over those wires. A LOWEREDOP stores the realization set and depends on every wire any realization uses until commitment.

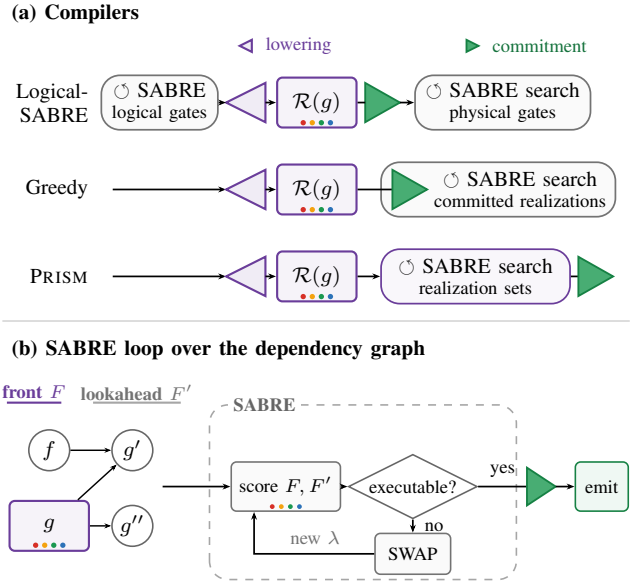


Fig. 6: (a) Each purple prism is the logical–physical boundary, where LOWEREXPANSION expands a logical gate into its realization set $\mathcal{R}(g)$ (colored dots), and each green arrowhead marks commitment to one realization. Logical-SABRE commits at lowering, Greedy before each router call, and PRISM when each gate executes. (b) The router scores each unresolved gate by its nearest realization, commits executable gates in the front layer, and otherwise inserts the best-scoring SWAP.

A logical CR_z on (a_0, a_1) and (b_0, b_1) becomes one LOWEREDOP containing the four circuits from Eq. (10), with interaction patterns

$$\{(a_i, b_j) : i, j \in \{0,1\}\}. \tag{13}$$

At the circuit's start and at each logical reset, LOWEREXPANSION emits a two-rail preparation of $|0_L\rangle$ per logical qubit, which the router must execute on adjacent sites. Measurement reads each rail separately and therefore needs no adjacency.

B. Commitment at Execution

Algorithm 1 routes the directed acyclic graph (DAG) D_{phys} of lowered operations. Each LOWEREDOP remains unresolved until the current layout λ supports a realization. For a one-interaction realization r on wires (u_r, v_r), let

$$\delta_\lambda(r) = d_G(\lambda(u_r), \lambda(v_r)), \tag{14}$$

where d_G is the shortest-path distance on the device graph G. A realization with no interaction, such as a single-rail rotation, has $\delta_\lambda(r) = 0$. The routing cost of an unresolved gate g is

$$h_\lambda(g) = \min_{r \in \mathcal{R}(g)} \delta_\lambda(r). \tag{15}$$

SABRE uses this cost in the front layer and lookahead set (line 17). The favored realization can therefore change with the layout while the rest of SABRE's candidate-SWAP neighborhood, weighting, and decay rules remain unchanged [5].

Algorithm 1 PRISM's inner SABRE routing loop.

```
1: function ROUTE(D_phys, G, λ_0)
2:     λ ← λ_0;  ν ← 0 on every site of G
3:     while FRONTLAYER(D_phys) is non-empty do
4:         F ← FRONTLAYER(D_phys);  progress ← FALSE
5:         for each g ∈ F do
6:             R_λ(g) ← {r ∈ R(g) : δ_λ(r) ≤ 1}
7:             if R_λ(g) is non-empty then
8:                 r* ← LEASTUSED(R_λ(g))
9:                 EMIT(C_r*);  REMOVE(g, D_phys)
10:                ν ← ν + 1 at both sites of r* if |R(g)| > 1
11:                progress ← TRUE
12:            end if
13:        end for
14:        if not progress then
15:            ν ← 0.9 ν if gates followed the last SWAP
16:            F' ← LOOKAHEAD(D_phys, F)
17:            s* ← SABRESWAP(F, F', G, λ, h_λ)
18:            EMIT(s*);   λ ← swap(λ, s*)
19:        end if
20:    end while
21:    return the emitted operations and λ
22: end function
```

An unresolved gate is executable when at least one realization has $\delta_\lambda(r) \le 1$ (line 6), so a realization with no interaction, which has $\delta_\lambda(r) = 0$, is always executable. To shorten the schedule, PRISM keeps a usage count ν at each device site and favors less-used sites when choosing among executable realizations. If several realizations are executable, LEASTUSED (line 8) commits the one whose two sites have the lowest total count and breaks remaining ties in a fixed order. The router inserts SWAPs until a front-layer gate becomes executable, then executes every gate that becomes executable. If gates have executed since the last SWAP, it multiplies all counts by 0.9 before choosing the next SWAP (line 15). Every forward pass, backward pass, and final trial restores each LOWEREDOP's full realization set and resets all counts to zero, so only the layout carries over.

An operation with one realization, such as R_{SWAP} or preparation, enters the loop as a one-element set without changing usage counts. A backward pass routes the preparations last. If a refined layout separates a rail pair, the next forward or final pass must reunite it before preparing it, and those SWAPs are counted. The lowering pass replaces each committed LOWEREDOP with its selected circuit.

C. Baseline Compilers

a) Greedy: Before each router call, Greedy selects for every unresolved gate g the nearest realization under its entry layout λ:

$$r_\lambda(g) = \arg\min_{r \in \mathcal{R}(g)} \sum_{(u,v) \in I_r} d_G(\lambda(u), \lambda(v)). \qquad (16)$$

Greedy breaks ties using the same fixed ordering as PRISM. Greedy holds these selections fixed while the router inserts SWAPs, so all final routing trials use the same selections. The entry layout can separate a rail pair that the route has yet to prepare, as it separates the rails of a in Fig. 7c.

b) Logical-SABRE: Logical-SABRE routes before it lowers. It forms disjoint adjacent rail pairs from a maximum-cardinality matching of the device graph that NetworkX [41] returns, which maximizes the number of supported logical qubits and breaks ties among maximum matchings by the order in which the device edges are listed. SABRELAYOUT and SABRESWAP then route the logical gates on the logical coupling graph, the simple quotient of the subgraph induced by the matched sites, with one vertex per matched edge and none for an unmatched site. Routing on this graph guarantees at least one available realization for each logical CR_z, and lowering then commits each gate to the first realization whose interactions are all available at the matched sites. Logical-SABRE's search uses the same budget of initial layouts, refinement iterations, and final trials as Greedy and PRISM (Section V-A), but selects routes by logical SWAP count without accounting for physical SWAPs added later. Its cost depends on its fixed site matching (Section V-C).

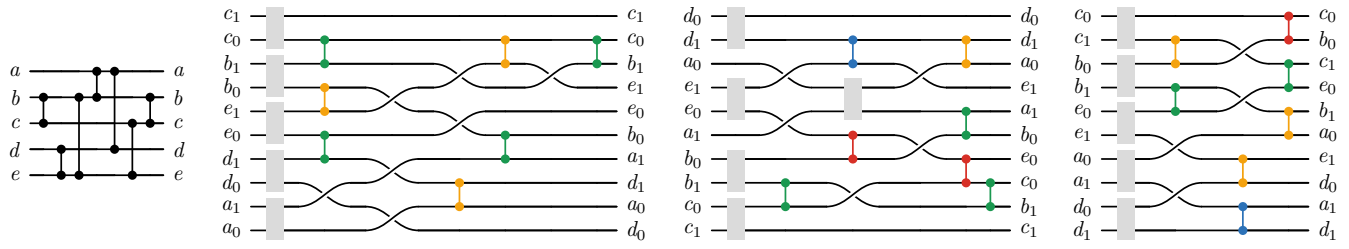
Unlike Greedy and PRISM, which insert only physical SWAPs, Logical-SABRE inserts logical SWAPs, which move both rails of each qubit even when moving one would make a gate executable. Each logical SWAP lowers to one of the circuits in Fig. 8, either two aligned SWAPs (a) or two crossed SWAPs followed by two intra-pair SWAPs (b) that remove the X_L the crossing leaves on each qubit. Circuit (c) would instead track that X_L as a Pauli frame, which Logical-SABRE does not do. Circuits (a) and (b) both need two disjoint device edges between the rail pairs, which with the two matched edges form a four-cycle. Logical-SABRE commits to the first of (a) and (b) whose interactions are available at the matched sites and otherwise commits to (a) and leaves it to Qiskit's stock SABRESWAP on the device graph, so on a device graph without four-cycles every logical SWAP takes this fallback. We count two physical SWAPs for circuit (a), four for (b), and any SWAPs added by physical routing. When these SWAPs move rails away from their matched sites, the router routes the previously chosen CR_z circuits as fixed two-qubit gates.

Figure 7 compares the three compilers on a constructed example using Qiskit's stock SABRE search budget of 20 random initial layouts, four refinement iterations, and 20 final trials. Logical-SABRE uses eight routing SWAPs at depth six, Greedy five at depth five, and PRISM four at depth three.

V. EVALUATION

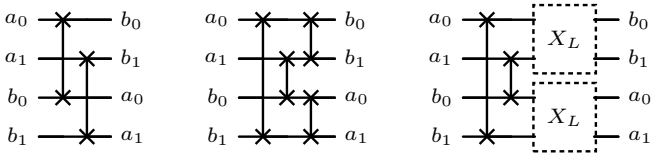
A. Experimental Setup

Seventeen workloads come from MQT Bench 2.2.2 at eight logical qubits, or seven for the VBE adder, which has no eight-qubit instance [42]. We exclude nine other MQT workloads on which Greedy and PRISM insert at most four routing SWAPs in every run on each device graph. The other eight workloads are the largest circuit of each QASMBench medium-scale family



(a) logical input (b) Logical-SABRE (8 SWAPs, 7 drawn, depth 6) (c) Greedy (5 SWAPs, depth 5) (d) PRISM (4 SWAPs, depth 3)

Fig. 7: Delaying commitment reduces routing SWAPs and depth on a constructed five-qubit example on a ten-site line. Panel (a) is the logical input, and (b)–(d) are the routes of the three compilers. Wires cross at each SWAP and keep their rail labels, and gray bars mark preparation. Colors identify the (control rail, target rail) pairing used for each logical gate in panel (a), as in Fig. 5. Each column is one layer, with gates placed as early as their wires allow. Depth omits preparation and single-rail phase corrections, and panel (b) shows seven SWAPs because cleanup removes one before measurement (Section V-A).



(a) aligned (b) crossed, corrected (c) crossed, frames

Fig. 8: Physical realizations of a logical dual-rail SWAP. (a) Two aligned SWAPs move each rail to the same-index rail of the other pair. (b) Two crossed SWAPs leave each pair role-reversed, and two intra-pair SWAPs restore the roles. (c) The dashed X_L boxes are frames tracked in software.

that needs routing, at its native width of 11 to 25 logical qubits [43]. A workload of n logical qubits runs on a grid, a hexagonal lattice, and a heavy-hex graph [44], each with at least $2n + 2$ sites, which hold its rail pairs and one spare pair (Appendix B). The primary experiment averages results over 64 compiler seeds, and error bars and $\pm$ values are 95% paired-bootstrap confidence intervals over seeds. The scaling and hardware experiments drop randomcircuit because it is a sampled random circuit and use the other 16 MQT workloads (Table IV) on the grid and hexagonal lattice, plus heavy-hex for hardware, and average results over five seeds.

We implement LOWEREXPANSION as a pass in Qiskit 2.4.1 [11]. Before it, a logical stage shared by the three dual-rail compilers decomposes multi-controlled gates with one clean ancilla, applies Toffoli, Pauli-frame, and two Hadamard-reduction rewrites, translates the circuit to R_X, R_Z, and CR_z, absorbs input SWAPs into the wire permutation, and fixes the operand order of each CR_z and thereby its SWS control. Our router is a fork of Qiskit 2.3.1's Rust LightSABRE router extended to process LOWEREDOP interactions. Greedy and PRISM use the same modified router, downstream physical compiler, stock SABRE heuristic, set of initial layouts, seed schedule, and search budget.

Following LightSABRE's randomized search procedure [45], each compilation tests 40 random initial layouts and the dense, identity, and reverse layouts of Qiskit's SabreLayout. Each layout receives 16 forward–backward refinement iterations, followed by a 40-trial final route. The compiler selects the completed route with the fewest routing SWAPs, using physical depth and then active gate count to break ties. A critical-path list scheduler then reorders commuting R_z and CR_z gates without changing the SWAPs and keeps each preparation before every gate on its rails. Under the SWAP objective it reschedules the selected route. Under the depth and duration objectives of Section V-C it reschedules every layout's route before selection.

We evaluate each final circuit by routing SWAP count, physical depth, critical-path duration, and compilation time. All three compilers end with a cleanup pass that removes SWAPs directly preceding a measurement, and routing SWAPs are counted before it. Logical X gates, implemented as intra-pair SWAPs, are not counted as routing SWAPs, and software-tracked Z rotations do not count toward depth. For each additive metric we report the reduction in the sum of workload means and the mean per-workload reduction, which weights workloads equally and omits those where Greedy's value is zero.

B. Delayed Commitment Reduces Routing Cost

PRISM reduces Greedy's total routing SWAP count by 30.8% [30.4, 31.3] on the grid, 26.0% [25.6, 26.5] on the hexagonal lattice, and 20.4% [20.1, 20.8] on heavy-hex, or by 30.0% [26.9, 31.1], 23.1% [21.7, 24.3], and 17.2% [16.9, 17.6] with equal workload weights, where brackets give 95% paired seed-bootstrap intervals. It reduces physical depth by 8.8%, 10.0%, and 10.1% suite-wide and by 8.2%, 9.2%, and 8.6% with equal weights (Fig. 9d, h). PRISM uses fewer routing SWAPs in 73 of the 75 workload and device-graph means and ties in two, although 202 of the 4,800 individual runs use more SWAPs than Greedy. Physical depth improves in 74 of the 75 means and rises by 0.2 layers for dj on the grid, where route selection minimizes SWAP count before depth. Relative to Logical-SABRE, Greedy reduces total routing SWAPs by 34.0% on the grid, 26.8% on the hexagonal lattice, and 52.9% on heavy-hex, while PRISM reduces them by 54.3%, 45.8%, and 62.5% and physical depth by 16.6%, 19.7%, and 34.4%. On heavy-hex these comparisons cover 24 workloads, because the site matching there has 24 pairs and cannot place knn.

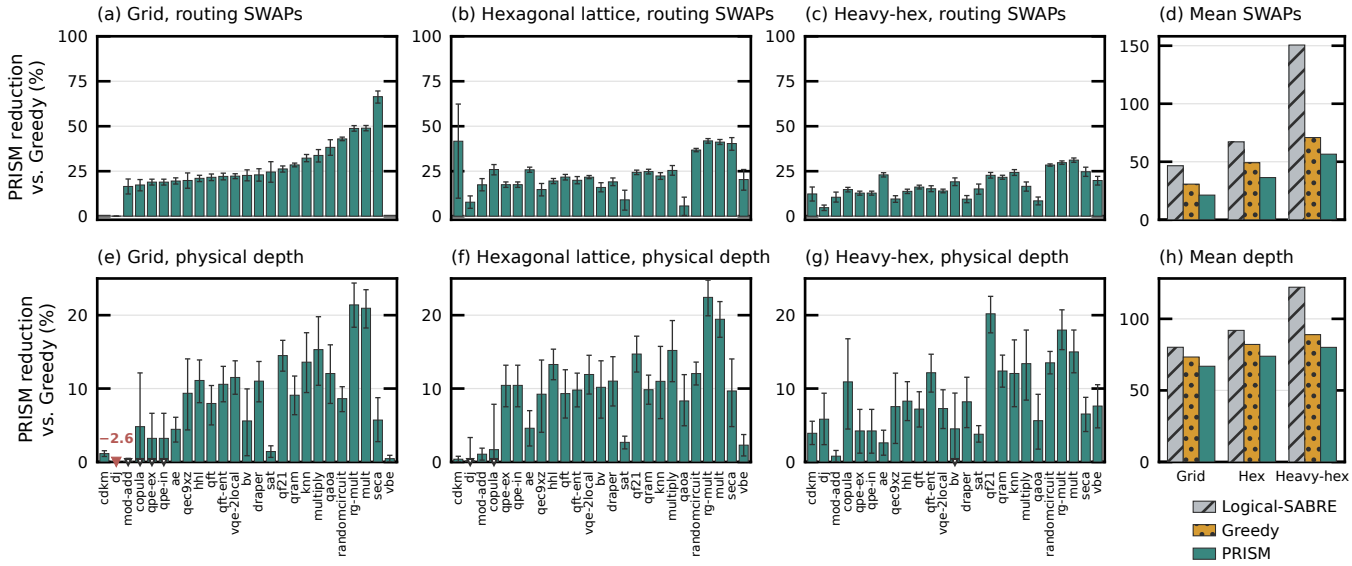


Fig. 9: Panels (a) to (c) and (e) to (g) show PRISM's SWAP and depth reductions relative to Greedy on each device graph. Bars compare 64-seed means with 95% paired seed-bootstrap intervals. Gray triangles mark intervals that extend below the axis, the red triangle gives the one negative mean (dj on the grid, -2.6% depth), and SWAP bars are omitted where Greedy averages under one routing SWAP (vbe and cdkm on the grid). Panels (d) and (h) show each compiler's mean routing SWAPs and physical depth, over the 24 workloads that Logical-SABRE can place on heavy-hex.

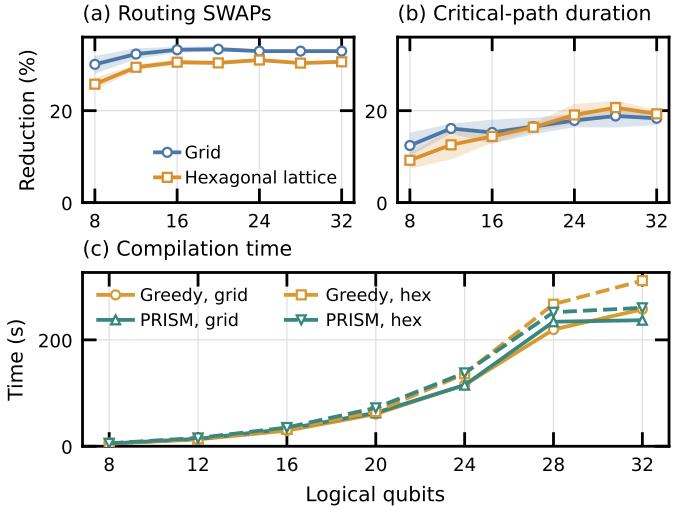


Fig. 10: PRISM results from 8 to 32 logical qubits. (a), (b) Reductions in routing SWAPs and critical-path duration relative to Greedy. (c) Per-compilation wall time. All average results over five seeds for each of the 16 MQT workloads.

Across the 16 MQT workloads from eight to 32 logical qubits, PRISM uses fewer routing SWAPs than Greedy or ties it for every combination of workload, device graph, and width except two on the hexagonal lattice, one at eight and one at 20 logical qubits. From 12 through 32 logical qubits, it reduces total routing SWAPs by 32.3–33.4% on the grid and 29.4–31.0% on the hexagonal lattice (Fig. 10). At 32 logical qubits the reductions are 33.0% and 30.6% (31.8% and 26.0% with equal workload weights), and PRISM also reduces physical depth by 22.6% and 23.4% and critical-path duration by 18.3% and 19.3%.

The PRISM-to-Greedy compilation-time ratio averages 1.03 across the device-graph and width means and spans 0.84 to 1.12 on an AMD Ryzen 5 5600X (Fig. 10c). Scoring an unresolved operation g at a fixed layout takes $O\left(\sum_{r\in\mathcal{R}(g)} |I_r|\right)$ time, a constant for the dual-rail ISA. Appendix C gives the router's time and space bounds, which are the same for both compilers.

C. Ablations and Sensitivity

On the 16 MQT workloads, an ablated PRISM commits each gate when it first enters the front layer, retaining layout-aware choice but preventing further layout changes before commitment (Fig. 11a). On entry it commits an executable realization if one exists and otherwise the nearest one under that layout (Eq. (16)), and it scores lookahead gates by Eq. (15) as PRISM does. Over 20 seeds, the layout-aware choice alone reduces Greedy's total routing SWAP count by 24.8% on the grid, 22.2% on the hexagonal lattice, and 17.3% on heavy-hex, and the choice kept open reduces the ablated count by a further 5.8%, 5.2%, and 1.4%. The ablated compiler averages more routing SWAPs than Greedy only for qaoa on the hexagonal lattice; PRISM does not show this increase.

PRISM also breaks ties among executable realizations by usage count, which Greedy never needs. Disabling the counts leaves every per-run SWAP count of the 16 MQT workloads at eight logical qubits on 20 seeds unchanged, and PRISM's depth reductions on the grid and hexagonal lattice move from

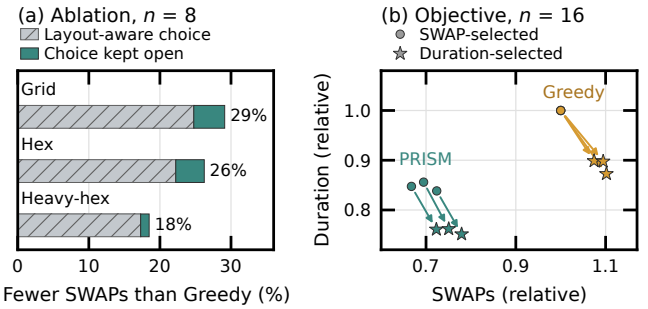


Fig. 11: Panel (a) shows routing SWAP reduction relative to Greedy at eight logical qubits. Gray bars (Layout-aware choice) show an ablated PRISM that commits gates on entering the front layer, and teal bars (Choice kept open) show PRISM's further reduction. Panel (b) plots routing SWAPs against duration at 16 logical qubits relative to Greedy, with one arrow per device graph from SWAP-selected to duration-selected routes.

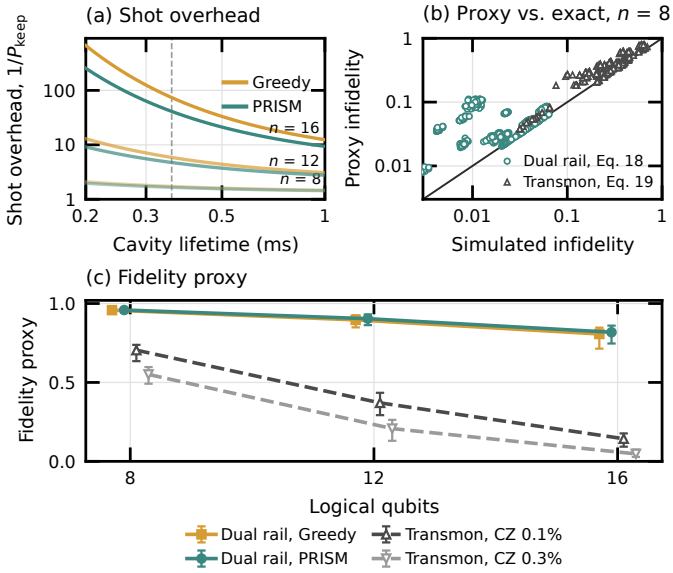


Fig. 12: Hardware projections use the 16 MQT workloads (five seeds) on the grid, hexagonal lattice, and heavy-hex. Panel (a) plots shot overhead P_{keep}^{-1} (Eq. (17)) against cavity lifetime, geometrically averaged over workloads at 8, 12, and 16 logical qubits. The dashed line marks the measured device. Panel (b) compares proxy infidelity (Eqs. (18) and (19)) with exact simulation at eight logical qubits, with transmons on the grid and hexagonal lattice only. Panel (c) shows median fidelity proxy and interquartile range. Dual-rail fidelity is over retained shots, and the transmon retains every shot.

11.1% and 10.4% to 9.9% and 9.9%, so commitment timing alone accounts for the SWAP difference.

On 25- and 36-site grids at eight logical qubits, which add spare sites, Greedy's total routing SWAP count stays within 0.3% of its 20-site value, and PRISM reduces it by 28.8% and 28.5%.

On the 16 MQT workloads at 16 logical qubits, selecting completed routes by depth or critical-path duration instead of SWAP count shifts both compilers alike, trading 6–10% more SWAPs for 9–13% shorter schedules (Fig. 11b). PRISM's advantage stays at 27.6–34.4% fewer SWAPs and 12.5–16.4% shorter schedules under every objective. The SWAP search still scores distance, so only the final route choice changes.

Selecting Greedy's realizations from the sites where each pair is prepared, instead of from each call's entry layout, changes 11% of its selections in a sample of 120 runs. This alternative changes Greedy's routing SWAPs by -2.7% to $+1.2\%$ per device graph and lengthens its critical-path duration by 1.3% to 2.4%. Logical-SABRE's routing SWAP count depends on its site matching. At eight logical qubits, using eleven alternative maximum matchings on each graph changes Logical-SABRE's routing SWAP count relative to its original matching, yielding 24% fewer to 9% more SWAPs on the grid and 0.5% to 36% fewer on heavy-hex. Four of the heavy-hex matchings leave a disconnected logical coupling graph on which the circuit cannot be placed. Logical-SABRE's fallback to the aligned circuit (Fig. 8a) lowers 50% of its logical SWAPs on the grid and every logical SWAP on the hexagonal lattice and heavy-hex. The SWAPs that its physical routing pass adds make up 35%, 53%, and 48% of its routing SWAPs on the grid, hexagonal lattice, and heavy-hex.

D. Shorter Routes Reduce Shot Overhead

Terminal erasure checks discard shots with photon loss, so the probability of loss along a route determines its shot overhead, the expected number of attempts per retained shot. Residual dephasing and bit flips are not flagged by the erasure checks, so affected shots remain in the retained sample and reduce its fidelity. We project both costs for the scaling routes at 8, 12, and 16 logical qubits (Table I).

TABLE I: Device models [10], [46], [47]. Dual-rail CZ is the physical CR_z (SWS) gate, with values for its control/target pair (C/T), and dual-rail 1Q is R_{SWAP}, averaged over the $\sqrt{X}$ gates measured on the control and target qubits. Dual-rail T_1 is the reciprocal of the mean decay rate of the four cavities, and T_ϕ is the modeled coherence decay time of a rail pair (Section V-D). Erasure is per rail pair touched, and residual is per operation. Transmon residuals are r_g in Eq. (19).

	Dual rail		Transmon	
T_1	357 μs		137 μs	
T_ϕ	2.2 ms		132 μs ($T_2 = 89\,\mu$s)	
	CZ	1Q	CZ	1Q
Duration (ns)	500	200	42	25
Erasure (10^{-4})	40 / 9.6	7.2	–	–
Residual (10^{-4})	3.9 / 1.1	0.99	10 or 30	3

The dual-rail model in Table I takes its parameters from measurements of Mehta et al.'s cavity device [10], and the transmon model uses typical published parameters [46], [47]

with two tested CZ errors. Mehta et al. estimate the phase-flip probability of an idle dual-rail qubit as t/T_{echo} and report echo times T_{echo} of 4.0 and 4.8 ms for their two dual-rail qubits. We set γ to the mean of the two rates $1/T_{\mathrm{echo}}$, $\gamma \approx (4.4\,\mathrm{ms})^{-1}$, so an idle rail pair suffers a phase flip with probability $(1 - e^{-2\gamma t})/2 \approx \gamma t$ and its coherence decays as $e^{-2\gamma t}$, with decay time $T_\phi = 1/(2\gamma) = 2.2\,\mathrm{ms}$. A cavity rail dephases far more slowly than it loses its photon ($T_\phi \gg T_1$), so most of its errors are detected, whereas a transmon has $T_\phi \approx T_1$. Both models assume ideal preparation, readout, and erasure checks. For every R_{SWAP} angle and routing SWAP between pairs, our model uses the values measured for the $\sqrt{X}$ beamsplitter gate, since the two-qubit device characterizes neither a full SWAP nor one between pairs.

Each dual-rail qubit holds one excitation for the whole circuit, and its rail pair loses that excitation at rate $1/T_1$ while it idles. Each operation also erases each pair it touches with a measured probability. The probability that a shot passes the terminal erasure checks is

$$P_{\mathrm{keep}} = \exp\left(-\frac{T_{\mathrm{idle}}}{T_1}\right) \prod_g (1 - e_g)^{K_g}, \tag{17}$$

where g ranges over three roles (the control pair of a CR_z, its target pair, and a pair in an R_{SWAP} or routing SWAP), e_g is the erasure probability of role g, and K_g counts the rail pairs touched in that role, so each CR_z adds one to the control count and one to the target count. The idle time T_{idle} sums, over the rail pairs that hold a logical qubit, the time within the scheduled duration $[0, D]$ during which neither rail of the pair is in an operation. The model treats every pair as prepared at time zero by an instantaneous, error-free preparation, so T_{idle} and K_g include the time and routing SWAPs before a pair's preparation. The shot overhead is P_{keep}^{-1} [22]. For the retained shots, following success-probability models [48], the product of the idle coherence factor and the gate fidelities gives the fidelity proxy

$$F_{\mathrm{cond}} = e^{-\gamma T_{\mathrm{idle}}} \prod_{g \in \{\mathrm{C},\mathrm{T},R_{\mathrm{SWAP}}\}} (1 - \epsilon_g)^{N_g} \tag{18}$$

for operation counts N_g with residual errors ϵ_g, where N_{C} and N_{T} both equal the CR_z count and $N_{R_{\mathrm{SWAP}}}$ counts each logical R_X realization and full-angle routing SWAP once, however many pairs it touches. For a SWAP between two pairs, we exclude its duration from both pairs' idle time, apply the SWAP erasure probability to each of the two rail pairs in Eq. (17), and use one factor $1 - \epsilon_{R_{\mathrm{SWAP}}}$, although $\epsilon_{R_{\mathrm{SWAP}}}$ was measured on a single pair. The factor $e^{-\gamma T_{\mathrm{idle}}}$ is a first-order approximation to, and a lower bound on, the probability that no pair suffers a logical phase flip while it idles. We aggregate each quantity as the geometric mean of the per-workload PRISM-to-Greedy ratios.

At 16 logical qubits, PRISM shortens the route of every workload, reducing total duration by 15.3%, 14.4%, and 16.2% on the grid, hexagonal lattice, and heavy-hex (11.5%, 13.7%, and 15.0% with equal workload weights). Because the probability of keeping a shot decays exponentially with idle time (Eq. (17)), the shot overhead falls by more than the duration. It falls on every workload but one, by 41.8%, 42.9%, and 48.1% as geometric means over workloads (44.3% over workloads and all three device graphs, Fig. 12a) and by 18.2%, 26.6%, and 28.7% for the median workload. The proxy infidelity $1 - F_{\mathrm{cond}}$ also falls on every workload but one, by 7.4%, 9.1%, and 10.0% as geometric means, a smaller gain because erasure conversion [23], [24] leaves few residual errors for a shorter route to remove.

For an unencoded transmon, which retains every shot, the same construction gives the fidelity proxy

$$F_{\mathrm{T}} = \exp\left[-T_{\mathrm{idle}}^{\mathrm{T}} \left(\frac{1}{4T_1} + \frac{1}{2T_2}\right)\right] \prod_g \left(1 - \frac{d_g + 1}{d_g}\, r_g\right)^{N_g}, \tag{19}$$

where $T_{\mathrm{idle}}^{\mathrm{T}}$ is the summed idle time of the active transmons, r_g is the average error of gate type g, and $d_g = 2^k$ for a k-qubit gate. The gate factor converts average to process fidelity [49], and the idle factor follows by the same conversion from the average idle fidelity under relaxation and dephasing [50].

We check Eqs. (17)–(19) by simulating the routed circuits under the same channels. At eight logical qubits we simulate exactly, with density matrices in Qiskit, all 480 dual-rail circuits on the three device graphs and 160 transmon circuits on the grid and hexagonal lattice. At 16 logical qubits we sample 200 quantum trajectories for each of the 320 dual-rail circuits on the grid and hexagonal lattice. The simulated acceptance equals Eq. (17) on every circuit. The fidelity proxies treat every residual error as fatal, although many leave the output state unchanged, so they overstate infidelity by a median factor of 1.6 at eight logical qubits and 1.8 at 16 for dual rail and 1.2 for the transmon (Fig. 12b).

Under a global-depolarizing assumption that counts both discarded shots and residual errors, dual rail needs less sampling time than an unencoded transmon by 16 logical qubits with either compiler, and PRISM lowers the ratio at every width (Fig. 12c). The transmon uses one qubit per logical qubit. Qiskit compiles each circuit onto the same device graphs at optimization level 3 into the basis $\{\mathrm{CZ}, \sqrt{X}, X, R_z\}$ with free R_z, and we keep the candidate with the highest estimated success probability among 80, from 40 transpiler seeds on the circuit before and after our logical stage. From 8 to 16 logical qubits the transmon's fidelity proxy at 0.1% CZ error falls from 0.70 to 0.14, while that of the retained dual-rail shots with PRISM stays between 0.96 and 0.82. To compare sampling time, we assume that the residual noise acts as global depolarizing noise, which local noise approaches in scrambling circuits [51]. A fidelity F then scales each observable's signal by F, so a fixed precision takes $1/F^2$ retained shots [22], [52], [53], and sampling time is proportional to the critical-path duration divided by $P_{\mathrm{keep}}F^2$ for dual rail and by F^2 for the transmon. Because dual-rail residual errors are mostly dephasing, which leaves Z-basis observables unattenuated, this assumption overstates the dual-rail cost. On the grid and hexagonal lattice at 8, 12, and 16 logical qubits, dual rail needs 2.5, 1.2, and 0.34 times the transmon's sampling time at 0.1%

CZ error with PRISM and 2.9, 1.8, and 0.72 times with Greedy, as geometric means over paired circuits. If the transmon instead ran two copies on the qubits that dual rail spends on second rails, every ratio would at most double, so at 16 logical qubits PRISM would still need at most 0.67 times the transmon's time and Greedy at most 1.44 times. These sampling-time estimates omit preparation, readout, reset, state-dependent coupler loss, erasure-check overhead, correlated errors, and decoding.

Of the varied hardware parameters, cavity lifetime matters most (Fig. 12a). At 16 logical qubits, the median workload needs 10.5 attempts per retained shot with PRISM and 12.6 with Greedy, and twice the cavity lifetime lowers these to 4.7 and 5.2, against 8.1 and 9.6 for a beamsplitter twice as fast and 9.3 and 11.7 for half the beamsplitter erasure probability. An erasure check that detects only a fraction η of erasures leaves the rest in the retained shots. The median fidelity proxy falls from 0.82 to 0.81 at Mehta et al.'s measured $\eta = 99.6\%$ [10] and to 0.75 at the $\eta = 96.3\%$ of de Graaf et al.'s check [21] with PRISM, and from 0.81 to 0.80 and 0.74 with Greedy. Doubling the residual gate errors lowers the two medians to 0.76 and 0.75. PRISM keeps its advantage under every variation, but at the measured parameters the median workload keeps only 10% of its shots at 16 logical qubits, motivating an outer code that can correct losses instead of discarding every affected shot.

E. Shorter Routes Lower Decoded Logical Error

Under an outer code, the decoder learns of an erasure only when a check or readout flags it. Checks inside the circuit flag erasures sooner, but each check forces the two rails of its pair to be adjacent. We compile a three-round $[[7, 1, 3]]$ Steane memory [54] using the qLDPC package's default extraction circuit [55] on the 48-site hexagonal lattice, where seven data and six ancilla qubits require 26 rails or 13 transmons. The circuit measures X checks with CX and Z checks with CZ from ancillas prepared in $|+\rangle$, and the X and Z memories, which differ in preparation and readout, are compiled separately. After lowering (Appendix D), each memory has 72 CR_z gates and 78 $R_X(\pm\pi/2)$ gates, each an R_{SWAP} that requires a qubit's rails to be adjacent.

With flags at readout only, the circuit has no intermediate erasure checks and each pair's two-rail readout flags its erasures. With compiled checks, each data pair also receives a joint erasure check after its last two-qubit gate of rounds one and two, and both compilers route the 14 checks like other two-rail operations. Ancilla pairs are read in every round. The checks follow de Graaf et al.'s measured check [21] on a different device, and the other faults follow Table I. We model an erasure as skip-gate leakage [30]. A lost pair is empty, every gate it takes part in does not happen until a check or readout flags it, and each flag resets the pair to a completely mixed state. Stim's tableau simulator [56] samples every fault at its scheduled location under the channels of Appendix D. A belief-propagation and ordered-statistics (BP+OSD) decoder [57], [58] receives erasure information only through the flags, and for each flag it includes the errors that a loss at any of the pair's locations since its last observation could have caused.

TABLE II: Decoded logical failure probability P_L of the three-round Steane memory on the hexagonal lattice under skip-gate leakage, averaged over twenty routes with 100,000 shots each, or 200,000 for the transmon rows. Brackets are 95% paired seed-bootstrap intervals. Reductions compare means.

Implementation	X memory P_L	Z memory P_L
Flags at readout only		
Dual rail, Greedy	0.7645%	0.7165%
Dual rail, PRISM	0.6648%	0.5760%
Relative reduction	13.0% [2.1, 22.5]	19.6% [12.3, 26.3]
Compiled erasure checks		
Dual rail, Greedy	1.2070%	1.2607%
Dual rail, PRISM	0.8728%	0.9756%
Relative reduction	27.7% [21.6, 32.8]	22.6% [13.8, 30.5]
Transmon, 0.1% CZ error	2.1724%	2.0526%
Transmon, 0.3% CZ error	5.6250%	5.2801%

Greedy and PRISM use twenty paired routing seeds, the search budget of Section V-A, and no final rescheduling. Each compiler completes one route per initial layout, 43 in all, and keeps the one most likely to have no erasure or Z fault. It minimizes the sum of $-\ln(1-e) - \ln(1-p_Z)$ over the scheduled fault locations, where e and p_Z are the erasure and Z fault probabilities of a location, with separate CR_z control and target rates and each pair's idle time from preparation to readout, while candidate SWAPs are still scored by distance and final trials by SWAP count. The transmon baseline is compiled as in Section V-D on the same graph with the same 16 refinement iterations and 40 final trials, is selected among its 80 candidate routes by the analogous no-fault probability, and has its thermal channel Pauli twirled [59] for Stim. Its comparison with dual rail depends on the hardware models, the transmon's instantaneous readout and reset, and the fixed extraction circuit.

With flags at readout only, PRISM lowers P_L by 13.0% in the X memory and 19.6% in the Z memory (Table II), on 14 and 18 of 20 seeds, with the lower confidence bound for the X-memory reduction near zero. Greedy's routes have more loss locations, so more of its shots lose two pairs, and such shots cause most failures. With checks, PRISM lowers P_L by 27.7% and 22.6%, on 19 and 17 seeds. Its checked routes use 42% and 35% fewer routing SWAPs than Greedy's and shorten the circuits from 20.8 to 17.3 μs and from 20.9 to 17.4 μs. The checks add one to four SWAPs to PRISM's routes and 3 to 14 to Greedy's, so PRISM retains its advantage despite the checks' adjacent-rail requirement. At these check parameters, checks raise both compilers' P_L, Greedy's to 1.6 and 1.8 times and PRISM's to 1.3 and 1.7 times their readout-only values.

VI. DISCUSSION

A. Interpreting the Results

Greedy and PRISM share the boundary IR, the router, and the search budget, so their performance difference reflects the effect of commitment timing. Logical-SABRE also routes on

a fixed logical coupling graph over a fixed site matching, so its gap mixes commitment timing with that design.

Choosing realizations before routing and choosing them during routing have the same global optimum, because every complete route fixes a realization assignment. Finding that optimum requires optimizing over m^{N_g} assignments of m realizations to N_g gates and solving a qubit-routing problem that is NP-hard even on a line [60]. PRISM uses a SABRE-based heuristic, which does not guarantee an optimal route [61].

The available choices depend on how long a pair's rails can remain apart. Logical X, Hadamard, and general R_X rotations need the two rails to be adjacent because only R_Z has an admissible product completion (Eq. (7)). Intermediate erasure checks also require adjacent rails (Section V-E). Between these operations the rails may separate, and the longer they stay apart, the more realization choices routing can use. While its rails are apart, a qubit can take part only in gates that preserve its logical Z value, because each operation touches at most one of its rails. Teleportation and inter-module communication [62], [63], measurement and feedforward [64], [65], ancilla-mediated embeddings [66], and microwave links between modules [67] lie outside this model.

B. Limitations and Future Work

A realization set holds only circuits that act on fixed wires, leave the same wire mapping and logical frame, and embed without SWAPs. For the logical SWAP inserted by Logical-SABRE, variant (c) leaves an X_L on each qubit, which a compiler that tracked Pauli frames [68] could absorb, and its other variants need two inter-pair edges at one layout. These restrictions limit the retained realizations, not the programs PRISM can compile, because every logical CR_z has a single-interaction realization that the router can make executable on any connected device graph. The definitions in Sections III-A and III-B can be used with other encodings once their admissible realization circuits and interaction patterns are specified. Such circuits may need routing between their own interactions. A logical graph weighted by routing cost would then generalize the availability graph.

The router keeps unresolved CR_z gates that share a logical qubit in program order, because each depends on all four rails of its operands, although their diagonal realizations commute. An executable gate can therefore wait behind one that still needs routing. The order constrains routing but not the emitted circuit, in which each committed gate depends only on its own two rails, so gates committed to disjoint rails of the shared qubit run in the same layer. Jointly selecting realizations for commuting gates could relax this ordering constraint.

Each realization also fixes which operand's rail is the SWS control. Because $\mathrm{CP}(\theta)$ is symmetric (Eq. (12)), exchanging the roles costs only software R_Z rotations and would double each CR_z realization set. We have not evaluated this choice, which could matter under an outer code because losing the control also dephases the target (Appendix D).

The boundary IR does not depend on SABRE. Lookahead, exact, and time-aware routers such as t|ket⟩'s [3], OLSQ [69], and TOQM [70] could use the boundary IR by testing which realizations embed under each layout and keeping alternatives until commitment.

VII. RELATED WORK

Synthesis supplies circuits for realization sets. Logical Clifford synthesis finds low-cost physical Clifford circuits [71], hardware-tailored synthesis uses freedom outside the code space to satisfy connectivity [72], stabilizer equivalence removes inaccessible interactions [73], and code automorphisms [74], [75] and chain maps [76] give circuits for a logical action.

Iceberg co-compilation jointly searches the algorithmic circuit and flexible fault-tolerant gadgets [77], and multi-qubit Iceberg compilation assigns program qubits to logical positions [78]. Surface-code compilers represent logical operations through patches, measurement regions, and routing paths [79]–[81], interleaved logical qubits derive connectivity from their physical constituents [82], and quantum low-density parity-check (qLDPC) compilation combines code symmetries with hardware routing [83]. Each targets a particular code and architecture, and none of them retains an encoded gate's alternative realizations through physical routing. Marol's device state machine [84] tracks where a fixed instruction can execute but does not retain alternative realizations.

The coupling between realization choice and layout is the phase-ordering problem of classical compilers. Instruction selection chooses among machine implementations of one IR operation [85], [86], register allocation and scheduling set each choice's cost, and committing either phase first can degrade the other [17]. Classical compilers exchange partial results between phases [87]–[90], search both phases jointly [91], or delay binding until later phases supply the required context [92].

Quantum mappers that exploit gate equivalence choose among implementations on the same physical qubits. They apply commutation rules [93], reorder Trotter terms [94], resynthesize blocks under a free output permutation [95], or estimate SWAP costs from later savings [96]. Orchestrated Trios routes a Toffoli's operands before selecting its decomposition [97], QContext selects decomposition variants after routing [98], and lazy synthesis interleaves SWAP insertion with synthesis [99]. MIRAGE is the closest prior approach, because its SABRE-style loop chooses between a two-qubit gate and a mirrored form that absorbs a SWAP [100]. An encoded gate's realizations instead use different wires and so need different device edges.

VIII. CONCLUSION

Routing changes which realization of an encoded gate can execute without SWAPs, so PRISM keeps each gate's realizations open until the router can execute one. For dual-rail cavity qubits, this cuts Greedy's routing SWAPs by 20.4–30.8% and physical depth by 8.8–10.1% across 25 workloads on three device graphs. On modeled cavity hardware, the shorter routes need 44.3% fewer attempts per retained shot at 16 logical qubits and lower decoded error in a three-round Steane memory.

Code Availability

PRISM is open source under the Apache-2.0 license at https://github.com/Yale-QCS/prism and installs as the Qiskit package `prism-qc`. The repository also includes the code, scripts, and data that reproduce every figure and number in this paper.

Acknowledgments

The authors thank Joe Aulicino, Raj Pabari, Nishaad Khedkar, and Chao Zhou for helpful discussions. This material is based upon work supported in part by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Center, Co-design Center for Quantum Advantage (C2QA) under Contract No. DE-SC0012704, and in part by Air Force Office of Scientific Research MURI (FA9550-26-1-B036). Y.D. acknowledges partial support by the National Science Foundation (under awards CCF-2312754 and CCF-2338063), by Quantum CT (under NSF Engines award ITE-2302908), by Boehringer Ingelheim, by NSF NQVL-ERASE (under award OSI-2435244), and by NSF QLCI Practiqal (under award OSI-2552449). External interest disclosure: Y.D. is a consultant and equity holder of Toric Labs and D-Wave Quantum. R.J.S. is a consultant and equity holder of D-Wave Quantum.

References

[1] J. Preskill, "Fault-tolerant quantum computation," in *Introduction to Quantum Computation and Information*. World Scientific, 1998, pp. 213–269.

[2] A. Paz and D. Grossman, "qstack: Compositional end-to-end compilation for fault-tolerant quantum programs," 2026, arXiv:2605.16595.

[3] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, "On the qubit routing problem," in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 135, 2019, pp. 5:1–5:32.

[4] X.-T. M. To, N. Quetschlich, A. Elsharkawy, M. Schulz, R. Wille, and D. Kranzlmüller, "Evaluating the effect of the order of optimization passes in quantum circuit optimization," 2026, arXiv:2609.29406.

[5] 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*, 2019, pp. 1001–1014.

[6] J. D. Teoh, P. Winkel, H. K. Babla, B. J. Chapman, J. Claes, S. J. de Graaf, J. W. O. Garmon, W. D. Kalfus, Y. Lu, A. Maiti, K. Sahay, N. Thakur, T. Tsunoda, S. H. Xue, L. Frunzio, S. M. Girvin, S. Puri, and R. J. Schoelkopf, "Dual-rail encoding with superconducting cavities," *Proceedings of the National Academy of Sciences*, vol. 120, no. 41, p. e2221736120, 2023.

[7] K. S. Chou, T. Shemma, H. McCarrick, T.-C. Chien, J. D. Teoh, P. Winkel, A. Anderson, J. Chen, J. C. Curtis, S. J. de Graaf, J. W. O. Garmon, B. Gudlewski, W. D. Kalfus, T. Keen, N. Khedkar, C. U. Lei, G. Liu, P. Lu, Y. Lu, A. Maiti, L. Mastalli-Kelly, N. Mehta, S. O. Mundhada, A. Narla, T. Noh, T. Tsunoda, S. H. Xue, J. O. Yuan, L. Frunzio, J. Aumentado, S. Puri, S. M. Girvin, S. H. Moseley, Jr., and R. J. Schoelkopf, "A superconducting dual-rail cavity qubit with erasure-detected logical measurements," *Nature Physics*, vol. 20, no. 9, pp. 1454–1460, 2024.

[8] A. Koottandavida, I. Tsioutsios, A. Kargioti, C. R. Smith, V. R. Joshi, W. Dai, J. D. Teoh, J. C. Curtis, L. Frunzio, R. J. Schoelkopf, and M. H. Devoret, "Erasure detection of a dual-rail qubit encoded in a double-post superconducting cavity," *Physical Review Letters*, vol. 132, no. 18, p. 180601, 2024.

[9] T. Tsunoda, J. D. Teoh, W. D. Kalfus, S. J. de Graaf, B. J. Chapman, J. C. Curtis, N. Thakur, S. M. Girvin, and R. J. Schoelkopf, "Error-detectable bosonic entangling gates with a noisy ancilla," *PRX Quantum*, vol. 4, p. 020354, 2023.

[10] N. Mehta, J. D. Teoh, T. Noh, A. Agrawal, A. Anderson *et al.*, "An entangling gate for dual-rail erasure qubits," *Nature*, vol. 656, pp. 47–53, 2026.

[11] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with Qiskit," 2024, arXiv:2405.08810.

[12] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t|ket⟩: A retargetable compiler for NISQ devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, 2021.

[13] E. Younis, C. C. Iancu, W. Lavrijsen, M. Davis, and E. Smith, "Berkeley Quantum Synthesis Toolkit (BQSKit) v1," Computer software, DOE CODE, 2021.

[14] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "MLIR: Scaling compiler infrastructure for domain specific computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2021, pp. 2–14.

[15] A. McCaskey and T. Nguyen, "A MLIR dialect for quantum assembly languages," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2021, pp. 255–264.

[16] Qiskit Team, "Release news: Qiskit v2.5 is here!" IBM Quantum Computing Blog, Jul. 2026, release notes: https://github.com/Qiskit/qiskit/releases/tag/2.5.0. [Online]. Available: https://www.ibm.com/quantum/blog/qiskit-2-5-release-summary

[17] S. R. Vegdahl, "Phase coupling and constant generation in an optimizing microcode compiler," in *Proceedings of the 15th Annual Workshop on Microprogramming (MICRO-15)*, 1982, pp. 125–133.

[18] W. Huang, X. Sun, J. Zhang, Z. Guo, P. Huang, Y. Liang, Y. Liu, D. Sun, Z. Wang, Y. Xiong, X. Yang, J. Zhang, L. Zhang, J. Chu, W. Guo, J. Jiang, S. Liu, J. Niu, J. Qiu, Z. Tao, Y. Zhou, X. Linpeng, Y. Zhong, and D. Yu, "Logical multi-qubit entanglement with dual-rail superconducting qubits," *Nature Physics*, vol. 22, no. 4, pp. 591–597, 2026.

[19] S. Ma, G. Liu, P. Peng, B. Zhang, S. Jandura, J. Claes, A. P. Burgers, G. Pupillo, S. Puri, and J. D. Thompson, "High-fidelity gates and mid-circuit erasure conversion in an atomic qubit," *Nature*, vol. 622, no. 7982, pp. 279–284, 2023.

[20] P. Scholl, A. L. Shaw, R. B.-S. Tsai, R. Finkelstein, J. Choi, and M. Endres, "Erasure conversion in a high-fidelity Rydberg quantum simulator," *Nature*, vol. 622, no. 7982, pp. 273–278, 2023.

[21] S. J. de Graaf, S. H. Xue, B. J. Chapman, J. D. Teoh, T. Tsunoda, P. Winkel, J. W. Garmon, K. M. Chang, L. Frunzio, S. Puri, and R. J. Schoelkopf, "A mid-circuit erasure check on a dual-rail cavity qubit using the joint-photon number-splitting regime of circuit QED," *npj Quantum Information*, vol. 11, no. 1, p. 1, 2025.

[22] S. J. Griffiths, J. Friel, and B. Vlastakis, "The limits of erasure-based postselection for quantum error mitigation," 2026, arXiv:2606.31428.

[23] Y. Wu, S. Kolkowitz, S. Puri, and J. D. Thompson, "Erasure conversion for fault-tolerant quantum computing in alkaline earth Rydberg atom arrays," *Nature Communications*, vol. 13, no. 1, p. 4657, 2022.

[24] A. Kubica, A. Haim, Y. Vaknin, H. Levine, F. Brandão, and A. Retzker, "Erasure qubits: Overcoming the T_1 limit in superconducting circuits," *Physical Review X*, vol. 13, no. 4, p. 041022, 2023.

[25] M. Violaris, L. Henaut, J. Wills, G. Consani, J. Friel, and B. Vlastakis, "Developments in superconducting erasure qubits for hardware-efficient quantum error correction," *Materials for Quantum Technology*, vol. 6, no. 3, p. 033001, 2026.

[26] K. Chang, S. Singh, J. Claes, K. Sahay, J. Teoh, and S. Puri, "Surface code with imperfect erasure checks," *PRX Quantum*, vol. 6, no. 4, p. 040355, 2025.

[27] M. Grassl, T. Beth, and T. Pellizzari, "Codes for the quantum erasure channel," *Physical Review A*, vol. 56, no. 1, pp. 33–38, 1997.

[28] S. Gu, A. Retzker, and A. Kubica, "Fault-tolerant quantum architectures based on erasure qubits," *Physical Review Research*, vol. 7, p. 013249, 2025.

[29] S. Gu, Y. Vaknin, A. Retzker, and A. Kubica, "Optimizing quantum error correction protocols with erasure qubits," *PRX Quantum*, vol. 6, p. 040354, 2025.

[30] M. Pavlovich, I. Rojkov, C. Wang, and S. Puri, "Erasure surface code circuit without mid-circuit erasure checks," 2026, arXiv:2607.29443.

[31] B. J. Chapman, S. J. de Graaf, S. H. Xue, Y. Zhang, J. Teoh, J. C. Curtis, T. Tsunoda, A. Eickbusch, A. P. Read, A. Koottandavida, S. O. Mundhada, L. Frunzio, M. H. Devoret, S. M. Girvin, and R. J.

Schoelkopf, "High-on-off-ratio beam-splitter interaction for gates on bosonically encoded qubits," *PRX Quantum*, vol. 4, p. 020355, 2023.
[32] Y. Lu, A. Maiti, J. W. O. Garmon, S. Ganjam, Y. Zhang, J. Claes, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf, "High-fidelity parametric beamsplitting with a parity-protected converter," *Nature Communications*, vol. 14, p. 5767, 2023.
[33] S. Puri, L. St-Jean, J. A. Gross, A. Grimm, N. E. Frattini, P. S. Iyer, A. Krishna, S. Touzard, L. Jiang, A. Blais, S. T. Flammia, and S. M. Girvin, "Bias-preserving gates with stabilized cat qubits," *Science Advances*, vol. 6, no. 34, p. eaay5901, 2020.
[34] P. Reinhold, S. Rosenblum, W.-L. Ma, L. Frunzio, L. Jiang, and R. J. Schoelkopf, "Error-corrected gates on an encoded qubit," *Nature Physics*, vol. 16, no. 8, pp. 822–826, 2020.
[35] E. McKinney and L. S. Bishop, "GULPS: Two-qubit gate synthesis via linear programming for heterogeneous instruction sets," 2025, arXiv:2505.00543.
[36] D. J. Brod and E. F. Galvão, "Extending matchgates into universal quantum computation," *Physical Review A*, vol. 84, no. 2, p. 022310, 2011.
[37] J. Zhang, J. Vala, S. Sastry, and K. B. Whaley, "Geometric theory of nonlocal two-qubit operations," *Physical Review A*, vol. 67, no. 4, p. 042313, 2003.
[38] R. R. Tucci, "An introduction to Cartan's KAK decomposition for QC programmers," 2005, arXiv:quant-ph/0507171.
[39] C. J. Wood and J. M. Gambetta, "Quantification and characterization of leakage errors," *Physical Review A*, vol. 97, no. 3, p. 032306, 2018.
[40] G. E. Crooks, "Gates, states, and circuits: Quantum gates," Three Plus One, Tech. Note 014, 2026, version 0.11.0 beta, 2026-02-21. [Online]. Available: https://threeplusone.com/pubs/on_gates.pdf
[41] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy 2008)*, 2008, pp. 11–15.
[42] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking software and design automation tools for quantum computing," *Quantum*, vol. 7, p. 1062, 2023.
[43] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "QASMBench: A low-level quantum benchmark suite for NISQ evaluation and simulation," *ACM Transactions on Quantum Computing*, vol. 4, no. 2, pp. 1–26, 2023.
[44] C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross, "Topological and subsystem codes on low-degree graphs with flag qubits," *Physical Review X*, vol. 10, no. 1, p. 011022, 2020.
[45] H. Zou, M. Treinish, K. Hartman, A. Ivrii, and J. Lishman, "LightSABRE: A lightweight and enhanced SABRE algorithm," 2024, arXiv:2409.08368.
[46] M. AbuGhanem, "IBM quantum computers: Evolution, performance, and future directions," *The Journal of Supercomputing*, vol. 81, p. 687, 2025.
[47] Google Quantum AI and Collaborators, "Quantum error correction below the surface code threshold," *Nature*, vol. 638, pp. 920–926, 2025.
[48] J. M. Baker, A. Litteken, C. Duckering, H. Hoffmann, H. Bernien, and F. T. Chong, "Exploiting long-distance interactions and tolerating atom loss in neutral atom quantum architectures," in *Proceedings of the ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 818–831.
[49] M. A. Nielsen, "A simple formula for the average gate fidelity of a quantum dynamical operation," *Physics Letters A*, vol. 303, no. 4, pp. 249–252, 2002.
[50] F. Bao, H. Deng, D. Ding, R. Gao, X. Gao *et al.*, "Fluxonium: An alternative qubit platform for high-fidelity operations," *Physical Review Letters*, vol. 129, no. 1, p. 010502, 2022.
[51] A. M. Dalzell, N. Hunter-Jones, and F. G. S. L. Brandão, "Random quantum circuits transform local noise into global white noise," *Communications in Mathematical Physics*, vol. 405, p. 78, 2024.
[52] J. Vovrosh, K. E. Khosla, S. Greenaway, C. Self, M. Kim, and J. Knolle, "Simple mitigation of global depolarizing errors in quantum simulations," *Physical Review E*, vol. 104, p. 035309, 2021.
[53] R. Takagi, S. Endo, S. Minagawa, and M. Gu, "Fundamental limits of quantum error mitigation," *npj Quantum Information*, vol. 8, p. 114, 2022.
[54] A. Steane, "Multiple-particle interference and quantum error correction," *Proceedings of the Royal Society of London. Series A*, vol. 452, pp. 2551–2577, 1996.
[55] M. A. Perlin, "qLDPC," GitHub repository, 2026, version 0.4.1. [Online]. Available: https://github.com/qLDPCOrg/qLDPC
[56] C. Gidney, "Stim: A fast stabilizer circuit simulator," *Quantum*, vol. 5, p. 497, 2021.
[57] P. Panteleev and G. Kalachev, "Degenerate quantum LDPC codes with good finite length performance," *Quantum*, vol. 5, p. 585, 2021.
[58] J. Roffe, D. R. White, S. Burton, and E. Campbell, "Decoding across the quantum low-density parity-check code landscape," *Physical Review Research*, vol. 2, no. 4, p. 043423, 2020.
[59] M. R. Geller and Z. Zhou, "Efficient error models for fault-tolerant architectures and the Pauli twirling approximation," *Physical Review A*, vol. 88, no. 1, p. 012314, 2013.
[60] T. Ito, N. Kakimura, N. Kamiyama, Y. Kobayashi, and Y. Okamoto, "Algorithmic theory of qubit routing," in *Algorithms and Data Structures*, ser. Lecture Notes in Computer Science, vol. 14079. Springer, 2023, pp. 533–546.
[61] J. Li, K. Li, R. Wang, Y. Zhao, K. Huang, Y. Wang, J. Zhang, Z.-A. Wang, X. Sun, and H. Fan, "QROB: Quantifying realization overhead in quantum compilation via reverse construction," 2026, arXiv:2609.07776.
[62] D. Devulapalli, E. Schoute, A. Bapat, A. M. Childs, and A. V. Gorshkov, "Quantum routing with teleportation," *Physical Review Research*, vol. 6, no. 3, p. 033313, 2024.
[63] A. Wu, H. Zhang, G. Li, A. Shabani, Y. Xie, and Y. Ding, "AutoComm: A framework for enabling efficient communication in distributed quantum programs," in *Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1027–1041.
[64] E. Bäumer, V. Tripathi, D. S. Wang, P. Rall, E. H. Chen, S. Majumder, A. Seif, and Z. K. Minev, "Efficient long-range entanglement using dynamic circuits," *PRX Quantum*, vol. 5, no. 3, p. 030339, 2024.
[65] E. Bäumer, V. Tripathi, A. Seif, D. Lidar, and D. S. Wang, "Quantum Fourier transform using dynamic circuits," *Physical Review Letters*, vol. 133, no. 15, p. 150602, 2024.
[66] N. Berthusen, D. Devulapalli, E. Schoute, A. M. Childs, M. J. Gullans, A. V. Gorshkov, and D. Gottesman, "Toward a 2D local implementation of quantum low-density parity-check codes," *PRX Quantum*, vol. 6, no. 1, p. 010306, 2025.
[67] J. D. Teoh, N. Cottet, P. Winkel, L. D. Burkhart, L. Frunzio, and R. J. Schoelkopf, "Robust quantum communication through lossy microwave links," 2025, arXiv:2509.18547.
[68] C. Chamberland, P. Iyer, and D. Poulin, "Fault-tolerant quantum computing in the Pauli or Clifford frame with slow error diagnostics," *Quantum*, vol. 2, p. 43, 2018.
[69] B. Tan and J. Cong, "Optimal layout synthesis for quantum computing," in *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD)*, 2020.
[70] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen, and E. Z. Zhang, "Time-optimal qubit mapping," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021, pp. 360–374.
[71] N. Rengaswamy, R. Calderbank, S. Kadhe, and H. D. Pfister, "Logical Clifford synthesis for stabilizer codes," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–17, 2020.
[72] E. J. Kuehnke, K. Levi, J. Roffe, J. Eisert, and D. Miller, "Hardware-tailored logical Clifford circuits for stabilizer codes," 2025, arXiv:2505.20261.
[73] J. Weinberg and N. Rengaswamy, "Stabilizers for compiling logical circuits under hardware constraints," 2026, arXiv:2604.25042.
[74] M. Grassl and M. Roetteler, "Leveraging automorphisms of quantum codes for fault-tolerant quantum computation," in *2013 IEEE International Symposium on Information Theory*, 2013, pp. 534–538.
[75] H. Sayginel, S. Koutsioumpas, M. Webster, A. Rajput, and D. E. Browne, "Fault-tolerant logical Clifford gates from code automorphisms," *PRX Quantum*, vol. 6, no. 3, p. 030343, 2025.
[76] A. Benhemou and N. Berthusen, "Automated logical Clifford gadgets for heterogeneous architectures via chain maps," 2026, arXiv:2607.02482.
[77] Y. Jin, Z. He, T. Hao, S. Omanakuttan, D. Amaro, S. Tannu, R. Shaydulin, and M. Pistoia, "Iceberg beyond the tip: Co-compilation of a quantum error detection code and a quantum algorithm," 2025, arXiv:2504.21172.
[78] C. Mazzetti, S. Sethi, R. Rines, P. Gokhale, and J. M. Baker, "Logical compilation for multi-qubit Iceberg patches," 2026, arXiv:2604.09956.
[79] D. Litinski, "A game of surface codes: Large-scale quantum computing with lattice surgery," *Quantum*, vol. 3, p. 128, 2019.

[80] A. Molavi, A. Xu, S. Tannu, and A. Albarghouthi, "Dependency-aware compilation for surface code quantum architectures," *Proceedings of the ACM on Programming Languages*, vol. 9, no. OOPSLA1, pp. 82:1–82:28, 2025.

[81] G. Watkins, H. M. Nguyen, K. Watkins, S. Pearce, H.-K. Lau, and A. Paler, "A high performance compiler for very large scale surface code computations," *Quantum*, vol. 8, p. 1354, 2024.

[82] J. Viszlai, S. Lin, S. Dangwal, C. Bradley, V. Ramesh, J. Baker, H. Bernien, and F. T. Chong, "Interleaved logical qubits in atom arrays," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025, pp. 261–274.

[83] K. Liu, T. Tsunoda, S. H. Xue, E. McKinney, Z. Zhou, S. Xu, R. J. Schoelkopf, and Y. Ding, "Efficient routing of quantum LDPC codes on programmable 2D toric architectures," 2026, arXiv:2604.18714.

[84] A. Molavi, A. Xu, E. Cecchetti, S. Tannu, and A. Albarghouthi, "Generating compilers for qubit mapping and routing," *Proceedings of the ACM on Programming Languages*, vol. 10, no. POPL, pp. 2265–2294, 2026.

[85] G. Hjort Blindell, *Instruction Selection: Principles, Methods, and Applications*. Springer International Publishing, 2016.

[86] D. R. Koes and S. C. Goldstein, "Near-optimal instruction selection on DAGs," in *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO'08)*, 2008, pp. 45–54.

[87] D. G. Bradlee, S. J. Eggers, and R. R. Henry, "Integrating register allocation and instruction scheduling for RISCs," in *Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)*, 1991, pp. 122–131.

[88] S. S. Pinter, "Register allocation with instruction scheduling: A new approach," in *Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation (PLDI'93)*, 1993, pp. 248–257.

[89] W. Ambrosch, M. A. Ertl, F. Beer, and A. Krall, "Dependence-conscious global register allocation," in *Programming Languages and System Architectures*, ser. Lecture Notes in Computer Science, vol. 782. Springer, 1994, pp. 125–136.

[90] C. Norris and L. L. Pollock, "An experimental study of several cooperative register allocation and instruction scheduling strategies," in *Proceedings of the 28th Annual International Symposium on Microarchitecture (MICRO-28)*, 1995, pp. 169–179.

[91] R. C. Lozano, M. Carlsson, G. H. Blindell, and C. Schulte, "Combinatorial register allocation and instruction scheduling," *ACM Transactions on Programming Languages and Systems*, vol. 41, no. 3, pp. 1–53, 2019.

[92] W. A. Wulf and K. V. Nori, "Delayed binding in PQCC generated compilers," Carnegie-Mellon University, Tech. Rep. CMU-CS-82-138, 1982, DTIC ADA125935.

[93] T. Itoko, R. Raymond, T. Imamichi, and A. Matsuo, "Optimization of quantum circuit mapping using gate transformation and commutation," *Integration*, vol. 70, pp. 43–50, 2020.

[94] L. Lao and D. E. Browne, "2QAN: A quantum compiler for 2-local qubit Hamiltonian simulation algorithms," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, 2022, pp. 351–365.

[95] J. Liu, E. Younis, M. Weiden, P. Hovland, J. Kubiatowicz, and C. Iancu, "Tackling the qubit mapping problem with permutation-aware synthesis," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2023, pp. 745–756.

[96] J. Liu, P. Li, and H. Zhou, "Not all SWAPs have the same cost: A case for optimization-aware qubit routing," in *Proceedings of the 28th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 709–725.

[97] C. Duckering, J. M. Baker, A. Litteken, and F. T. Chong, "Orchestrated trios: Compiling for efficient communication in quantum programs with 3-qubit gates," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 375–385.

[98] J. Liu, M. Bowman, P. Gokhale, S. Dangwal, J. Larson, F. T. Chong, and P. D. Hovland, "QContext: Context-aware decomposition for quantum gates," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–5.

[99] S. Martiel and T. G. de Brugière, "Architecture aware compilation of quantum circuits via lazy synthesis," *Quantum*, vol. 6, p. 729, 2022.

[100] E. McKinney, M. Hatridge, and A. K. Jones, "MIRAGE: Quantum circuit decomposition and routing collaborative design using mirror gates," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 704–718, arXiv:2308.03874.

APPENDIX A
WEYL-CHAMBER COORDINATES OF DUAL-RAIL COMPLETIONS

The surfaces and lines in Fig. 3 follow from the canonical form of parity-preserving gates [36]. Write the nonlocal part of a two-rail unitary as $U_{\mathrm{nl}} = \exp[i(c_1 X\otimes X + c_2 Y\otimes Y + c_3 Z\otimes Z)/2]$, so that (c_1, c_2, c_3) are its Weyl-chamber coordinates, and for the blocks of Eq. (5) let $\cos\phi_B = |B_{11}|$ and $\cos\phi_A = |A_{11}|$. Before the coordinates are folded into the Weyl chamber, $c_1 = \phi_B + \phi_A$, $c_2 = \phi_B - \phi_A$, and c_3 is set by the relative phase of $\det A$ and $\det B$. Fixing the logical gate fixes B and the unfolded sum $c_1 + c_2 = 2\phi_B$, while varying A sweeps the folded surface. The surface reaches the local equivalence class at the origin only when $\phi_B \in \{0, \pi/2\}$, which means that B is diagonal or anti-diagonal and gives the family of Eq. (6).

An admissible completion has the diagonal A of Eq. (7), so $\phi_A = 0$. Since $\det A = e^{2i\alpha}$ does not depend on β, the phase α alone sweeps c_3 along the orange line $c_1 = c_2 = \phi_B$. For logical X, $\phi_B = \pi/2$ and this line is the parametric-SWAP edge from iSWAP to SWAP.

APPENDIX B
DEVICE GRAPH DIMENSIONS

Table III lists the device graphs at every evaluated width n. All three come from Qiskit's coupling-map generators. The grid has $R = \lceil\sqrt{2n+2}\,\rceil$ rows and $C = \lceil(2n+2)/R\rceil$ columns. The hexagonal lattice has $k \times k$ hexagons with $k = \lceil\sqrt{n+1}\,\rceil$, which gives $2(k+1)^2 - 2$ sites. The heavy-hex graph has the smallest odd distance $d \geq 3$ whose $(5d^2 - 2d - 1)/2$ sites hold the workload. Under the $k \times k$ rule the hexagonal lattice has more spare sites than the grid at every width, and at ten of the 13 widths the next smaller $k \times k$ lattice would also hold the workload. MQT Bench's VBE adder on m-bit operands has $3m + 1$ qubits, so it exists at 16 and 28 logical qubits. At the other widths it has 7, 10, 19, 22, and 31 logical qubits and uses the graphs for $n = 8$, 12, 20, 24, and 32. The larger grids of Section V-C are 5×5 and 6×6.

APPENDIX C
ROUTER COMPLEXITY

For L initial layouts with ρ forward–backward refinements and τ final trials each, N lowered operations, at most S inserted SWAPs per pass, and n logical qubits on a device of n_{phys} sites and maximum degree Δ, both compilers run the same router in time

$$O\big(L(2\rho+\tau)\big(N + S(n_{\mathrm{phys}} + n\Delta\log(n\Delta))\big) + L(2\rho+1)\,n_{\mathrm{phys}}^3\big) \tag{20}$$

and space $O(n_{\mathrm{phys}}^2 + N)$, plus $O(N_{\mathrm{out}}^2)$ time to schedule the N_{out} gates of each scheduled route. The $n\Delta\log(n\Delta)$ term is the per-step sort of candidate SWAPs by physical endpoints, and the n_{phys} term is a per-step scan of all device sites that

TABLE III: Site counts appear in parentheses for each width n. Heavy-hex is not evaluated at $n = 24$, 28, and 32.

n	$2n+2$	Grid $R \times C$	Hexagonal k	Heavy-hex d
8	18	5×4 (20)	3 (30)	3 (19)
11	24	5×5 (25)	4 (48)	5 (57)
12	26	6×5 (30)	4 (48)	5 (57)
13	28	6×5 (30)	4 (48)	5 (57)
15	32	6×6 (36)	4 (48)	5 (57)
16	34	6×6 (36)	5 (70)	5 (57)
17	36	6×6 (36)	5 (70)	5 (57)
19	40	7×6 (42)	5 (70)	5 (57)
20	42	7×6 (42)	5 (70)	5 (57)
24	50	8×7 (56)	5 (70)	
25	52	8×7 (56)	6 (96)	5 (57)
28	58	8×8 (64)	6 (96)	
32	66	9×8 (72)	6 (96)	

TABLE IV: Of the 25 workloads in Fig. 9, those from MQT Bench 2.2.2 are algorithm-level benchmarks with $n = 8$, except vbe with $n = 7$. The 16 MQT workloads used for scaling appear above the horizontal rule. The QASMBench medium-scale files end in their width n, which the labels omit.

Label	MQT Bench benchmark
dj	`dj`
qaoa	`qaoa`
draper	`draper_qft_adder`
mod-add	`modular_adder`
qft	`qft`
qpe-ex	`qpeexact`
qpe-in	`qpeinexact`
qft-ent	`qftentangled`
copula	`bmw_quark_copula`
hhl	`hhl`
ae	`ae`
cdkm	`cdkm_ripple_carry_adder`
vbe	`vbe_ripple_carry_adder`
vqe-2local	`vqe_two_local`
rg-mult	`rg_qft_multiplier`
mult	`multiplier`
randomcircuit	`randomcircuit`
Label	**QASMBench file**
sat	`sat_n11`
seca	`seca_n11`
multiply	`multiply_n13`
qf21	`qf21_n15`
qec9xz	`qec9xz_n17`
bv	`bv_n19`
qram	`qram_n20`
knn	`knn_n25`

finds the rails of unresolved front-layer gates and decays the usage counts. For each candidate SWAP, the router rescores only the front-layer gates on its two sites and at most 20 lookahead gates, so scoring takes constant time. The n_{phys}^3 term recomputes the distances between device sites, by a bit-parallel breadth-first search from every site, at the start of each of the $2\rho + 1$ router calls per layout. The $O(n_{\text{phys}}^2 + N)$ space bound accounts for the distance matrix and routing state.

TABLE V: Fault channels at each rail-pair location of the dual-rail Steane memory, with the erasure and residual probabilities e_g and ϵ_g of Eqs. (17) and (18) and Table I. t_{idle} is the length of an idle gap, and a routing SWAP's residual acts on the pair that ends on its first operand. The transmon rows of Table II use the Pauli-twirled thermal channel instead.

Location	Erasure	Residual
Idle gap	$1 - e^{-t_{\text{idle}}/T_1}$	Z, $(1 - e^{-2\gamma t_{\text{idle}}})/2$
R_{SWAP}	$e_{R_{\text{SWAP}}}$	Z, $\epsilon_{R_{\text{SWAP}}}$
Routing SWAP	$e_{R_{\text{SWAP}}}$ on each pair	Z, $\epsilon_{R_{\text{SWAP}}}$ on one pair
CR_z control	$e_{\text{C}}/2$ before, $e_{\text{C}}/2$ after	Z, ϵ_{C} after
CR_z target	e_{T} after	Z, ϵ_{T} after
Joint check	idle gap, $t_{\text{idle}} = 2.94\,\mu$s	idle Z, depolarizing 0.12%
Readout	none	none

APPENDIX D
STEANE MEMORY NOISE MODEL

Circuit. Lowering writes H as $R_Z(\pi/2)R_X(\pi/2)R_Z(\pi/2)$, CX as CZ conjugated by H on the target, and CZ as $\text{CR}_z(\pi)$ with an R_Z correction, and the transmon baseline uses the same rules over CZ, $\sqrt{X}$, X, and R_z. The logical stage absorbs the Hadamard gate of each $|+\rangle$ preparation into the checks, so the ancilla pair is the CR_z control in the Z checks of the X memory and in the X checks of the Z memory.

Faults. Every fault location belongs to one rail pair (Table V). An idle gap is an interval between a pair's preparation or reset and its readout in which no operation of nonzero duration touches either rail. Preparation and readout are otherwise ideal. Following Mehta et al. [10], we assign half of the CR_z control's erasure probability before the gate.

Leakage. An erased pair is empty until a check or readout flags it [30]. Until then, every operation that touches it is skipped as a unit, including a CR_z with it as either operand, the R_Z corrections on the intact partner, and the pair's own residual faults. A control lost before its CR_z therefore leaves the target without a phase that depends on the lost control, so the target receives Z with probability $1/2$, or $e_{\text{C}}/4 = 10^{-3}$ per gate. A flag resets the pair to the completely mixed state, and a false flag resets an intact pair in the same way, because the hardware cannot tell the two apart. Ancilla pairs are also reset in every round, and data pairs are never otherwise reset. Stim has no leakage state, so each shot with a fault runs Stim's tableau simulator and skips the empty pair's operations.

Checks. A check takes $2.94\,\mu$s including readout. It flags an erased pair with probability $1 - f_n$ and an intact one with probability f_p, with $f_n = 3.7\%$ and $f_p = 0.51\%$. A missed loss stays unreported until the pair's next check or readout, and a readout flags every unreported loss and never flags falsely. With an ideal flag at every loss instead, which resets each lost pair before it skips a gate, PRISM lowers P_L by 25.7% [18.5, 32.3] in the X memory and 25.4% [17.1, 32.8] in the Z memory over the same twenty routes with 200,000 shots each.

Decoder. A flag does not identify when its pair was lost, so for flag k the decoder considers each fault location ℓ since the pair's previous flagged check, readout, or reset. With e_ℓ

the erasure probability of ℓ and a_ℓ the number of unflagged checks between ℓ and the flag, $w_\ell = e_\ell f_n^{a_\ell} \prod_{\ell' < \ell}(1 - e_{\ell'})$ is the probability that the pair was first lost at ℓ and every later check missed it. For a loss at ℓ, the possible errors are the fault mechanisms in Stim's detector error model after removing the pair's operations from ℓ to the flag and depolarizing it at the flag. Each such mechanism becomes a decoder column with prior

$$q_k(\ell) = \frac{1}{2} \frac{w_\ell}{\sum_{\ell'} w_{\ell'} + c_k f_p/(1 - f_n)}, \tag{21}$$

where c_k is 1 for a check and 0 for a readout, so the f_p term weighs a false flag. Contributions from several locations combine as $p + q - 2pq$, and the reset's own mechanisms, which occur for every candidate loss location and also carry the false-flag weight, receive prior $1/2$. Without a flag, the decoder keeps each location's residual Z prior and, at a check, two thirds of the depolarizing probability on each of its X and Z faults. BP+OSD runs seven serial product-sum iterations and order-2 combination-sweep OSD.

APPENDIX E
AI USE

The authors wrote the original software and manuscript. Claude (Anthropic) and ChatGPT (OpenAI) later assisted in revising both, and the authors reviewed and verified all AI-assisted changes.