MARSA: JOINT LEARNING OF QUBIT MAPPING AND ROUTING WITH VERIFIABLE SELF-IMPROVEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Executing a quantum circuit on a processor with limited connectivity requires choosing an initial qubit mapping and a sequence of routing operations. These decisions are often exposed separately, but they are fundamentally coupled: a mapping is useful only through the routing trajectories it enables, while the best route depends on the mapping from which it starts. We study this coupling directly with **MARSA** (**M**apping **A**nd **R**outing with a **S**emantic **A**utomaton), a single autoregressive policy that generates an initial mapping followed by the complete physical routing trace. This representation creates a structured-generation problem: the legality of every new token depends on the current logical-to-physical mapping, the hardware graph, and the unexecuted circuit. We therefore introduce a *route-state automaton* that maintains these semantics progressively and masks invalid actions during both inference and reinforcement learning rollouts. To learn this long-horizon structured policy efficiently, we initialize from a strong heuristic expert using a two-axis curriculum over circuit width and length, then move beyond imitation through *verifier-guided self-improvement*: MARSA generates its own complete solutions, exact verification evaluates their correctness and SWAP cost, and GRPO reinforces the better generations. On Q20 random circuits, MARSA strictly improves on the training expert for 81.12% of circuits, matches or improves it for 97.15%, and reduces aggregate SWAP count by 16.40%; it also transfers from synthetic training to held-out real quantum algorithm circuits, reducing aggregate SWAP count by 71.35% relative to Qiskit LightSABRE. A separately trained model on a second 2D grid coupling graph also achieves strong performance. Together, these results show that joint mapping and routing can be cast as structured generation, enabling a learned compiler to optimize both and continue improving through its own verified solutions.

1 INTRODUCTION

Quantum computing promises computational advantages for problems that are believed to be difficult for classical machines, with quantum algorithms represented as circuits of gates acting on logical qubits. On quantum processors with sparse two-qubit connectivity, however, interactions are restricted by a hardware coupling graph. Compilation must therefore decide *where* logical qubits should be placed and *how* they should be moved when required interactions are not physically adjacent. These decisions are tightly coupled: a mapping that appears attractive under one routing strategy can be poor under another, because its value depends on the movement it induces over the rest of the circuit. Conversely, an effective router may exploit mappings that a separately optimized mapper would never select. Classical methods already reflect this dependence. SABRE greedily selects SWAPs using distances among front-layer and look-ahead interactions and improves the initial mapping through bidirectional routing (Li et al., 2019); FIDLS combines graph-based initial mapping with depth-limited SWAP search (Li et al., 2021); and exact formulations optimize assignment and movement within a common integer linear programming model (Nannicini et al., 2022).

Learning-based compilers have approached this problem from several directions. Reinforcement learning methods directly learn routing decisions, such as selecting SWAPs from the current routing state (Pozzi et al., 2022; Kremer et al., 2024). QRoute and AlphaRouter combine learned policies or value estimates with Monte Carlo tree search to capture long-horizon routing structure (Sinha et al., 2022; Tang et al., 2024). Other methods optimize initial placement as a separate stage, eval-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

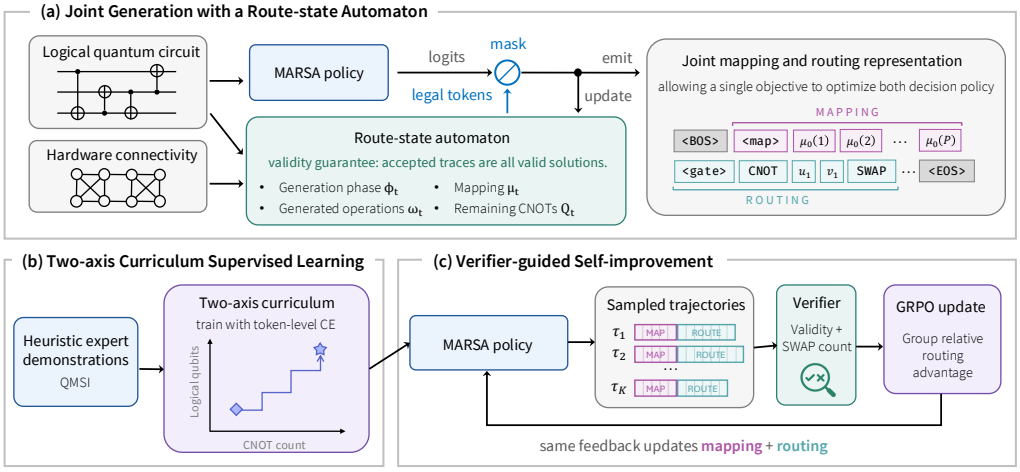


Figure 1: Overview of the MARSA framework. **(a)** MARSA jointly generates the initial mapping and routing trace under constrained generation, the output sequence is thus guaranteed to be a valid solution. **(b)** Two-axis curriculum supervised learning initializes the policy from expert demonstrations while progressively increasing circuit width and length. **(c)** Verifier-guided self-improvement uses exact route feedback and GRPO to improve the joint policy beyond the heuristic expert.

uating candidate mappings through the routing cost they induce (Fan et al., 2022; Elsayed Amer et al., 2024). Using supervised learning rather than reinforcement learning, Ren et al. (2024) train a Seq2Seq model to propose short SWAP sequences from a supplied circuit window. These approaches demonstrate that learning can capture useful compilation structure, but they retain a separation between initial mapping and the routing process that follows. This leaves a central unresolved problem: *can one learned representation describe the initial mapping and the complete route together, so that the same eventual routing objective can optimize both?*

We answer this question with **MARSA** (**M**apping **A**nd **R**outing with a **S**emantic **A**utomaton), which casts qubit mapping and routing as a single structured-generation problem. Given a logical circuit, a Transformer first generates the logical-to-physical mapping and then the complete physical routing trace with both CNOT and SWAP gates in the same sequence. Every routing decision is therefore conditioned on the mapping actually generated, and the same route-level objective can improve both the mapping prefix and the routing continuation. In this way, MARSA turns joint mapping and routing from separate compiler stages into a single autoregressive optimization problem.

This joint representation comes with a challenge for autoregressive generation: routing is not free-form sequence generation. A routing trace must obey both the syntax and the evolving routing semantics: a SWAP is legal only along a hardware edge, every SWAP changes the current mapping and hence which dependency-ready interactions are executable, and executing an interaction advances the circuit frontier. In many learned routers, the policy selects among hardware-edge SWAPs while a classical controller tracks dependencies and executes eligible circuit gates (Sinha et al., 2022; Tang et al., 2024). MARSA instead generates from a much larger space containing the initial mapping followed by a sequence of SWAP/CNOT tokens. The validity of the next token therefore depends on the evolving routing state. MARSA addresses this with a *route-state automaton* that incrementally tracks the current mapping, remaining circuit dependencies, hardware constraints, and partially generated operation. At every decoding step, it masks the inadmissible tokens, providing *validity by construction*. The resulting division of labor is clear: the model learns *which* admissible action is useful, while the automaton enforces *what* constitutes a valid routing solution. During reinforcement learning, the same mechanism prevents rollouts from being wasted on infeasible trajectories and confines exploration to the space of meaningful routing solutions.

Even within this constrained space, learning such a long-horizon structured policy from scratch would require exploring an enormous space of possible mapping–routing trajectories. We therefore begin with a strong teacher: the heuristic search expert from Ren et al. (2024), which we refer to as

QMSI (**Q**ubit **M**apping and **S**WAP **I**nsertion). Its complete solutions provide coordinated mapping and routing behavior. To make this difficult structure learnable at scale, we organize supervised training with a *two-axis curriculum* that progressively increases both circuit width and circuit length, capturing respectively the growth of the mapping space and the horizon over which early decisions affect downstream cost.

Supervised initialization provides an effective way into this difficult solution space, but imitation alone introduces a *teacher ceiling*: training only on QMSI solutions provides no direct mechanism for systematically discovering better ones. Qubit routing offers a natural path beyond this ceiling: although finding a high-quality route is difficult, a completed solution can be exactly verified and scored directly by its SWAP count. Once MARSA can reliably generate complete solutions, we shift from teacher supervision to *verifier-guided self-improvement*: MARSA samples groups of its own mapping–routing trajectories, and exact route feedback with GRPO (Shao et al., 2024) reinforce the better generations. Thus the teacher provides the starting point, while MARSA's own verified solutions drive improvement beyond it.

We evaluate the policy on two widely used hardware topologies: (1) IBM's 20-qubit Q20, a standard benchmark for quantum circuit compilation, and (2) a 2D grid, the native layout of Google's and many other superconducting processors, which directly supports surface-code encoding. The policy substantially surpasses its training expert across both topologies. On held-out Q20 random circuits, best-of-16 MARSA strictly improves on QMSI for 81.12% of instances, matches or improves it for 97.15%, and reduces aggregate SWAP count by 16.40%. Importantly, these gains are not solely an effect of inference-time candidate selection: with a single greedy trajectory, MARSA already strictly beats QMSI on 61.20% of circuits and reduces aggregate SWAP count by 7.01%. The result reproduces on the substantially larger 60-qubit 2D coupling graph, where MARSA achieves 78.85% strict wins and a 10.57% aggregate SWAP reduction. MARSA also transfers from synthetic training to held-out real quantum algorithm circuits. On Q20, it remains beyond its QMSI training expert and reduces aggregate SWAP count by 71.35%, 71.19%, and 70.86% relative to LightSABRE (Zou et al., 2024), ML-SABRE (Ping et al., 2025), and Qiskit AIRouting (Kremer et al., 2024), respectively. Together, these results show beyond-teacher policy improvement that persists across topology scale and transfers to unseen algorithmic workloads.

In summary, our main contributions are:

- **Joint autoregressive mapping and routing.** To our knowledge, MARSA is the first learned policy to generate the initial mapping and the complete routing trace as a single trajectory, allowing both to be optimized by the final routing objective.
- **A route-state automaton for validity by construction.** We introduce an incremental routing automaton that tracks the evolving mapping, circuit frontier, and hardware constraints, restricting inference and training rollouts to semantically valid operations.
- **Verifier-guided self-improvement.** We combine a two-axis curriculum from a strong heuristic expert with GRPO on MARSA's own verified generations, enabling the policy to systematically improve beyond its demonstrations.
- **Strong beyond-expert performance and synthetic-to-real transfer.** MARSA improves substantially on its QMSI training expert across two hardware topologies and transfers from synthetic training to held-out real quantum algorithm circuits.

2 Mapping and routing as autoregressive optimization

2.1 Problem formulation

Let $G = (V, E)$ be the hardware coupling graph, where V is the set of $P = |V|$ physical qubits and E the set of allowed two-qubit couplings. Following the convention in mapping and routing literature, we use *logical qubits* to refer to input circuit qubits and *physical qubits* to hardware qubits. Because single-qubit gates impose no connectivity constraints, we formulate the mapping and routing problem over the circuit's CNOT interactions. Let $C = (g_1, \ldots, g_M)$ denote the ordered CNOT gates of a logical circuit on $n \leq P$ logical qubits, with $g_j = \mathrm{CNOT}(\ell_j, k_j)$. Their execution constraints form a dependency DAG: each CNOT is a node, and directed edges encode the order

that must be preserved along shared logical qubits. A CNOT is *dependency-ready* when all of its predecessors have been executed.

An initial mapping μ_0 assigns logical qubits to distinct physical qubits. At routing step r, a dependency-ready CNOT between logical qubits (ℓ, k) can be executed when their current physical locations are adjacent,

$$\big(\mu_{r-1}(\ell), \mu_{r-1}(k)\big) \in E. \tag{1}$$

If they are not, routing inserts SWAPs along hardware edges; the mapping evolves according to

$$\mu_r = \begin{cases} \sigma_{uv} \circ \mu_{r-1}, & a_r = \mathrm{SWAP}(u, v), \\ \mu_{r-1}, & a_r = \mathrm{CNOT}(u, v), \end{cases} \tag{2}$$

where σ_{uv} exchanges the logical occupants of adjacent physical sites u and v.

We represent a complete solution of initial mapping and routing as

$$y = [\mu_0, a_1, \ldots, a_R], \tag{3}$$

where each a_r is either an executed CNOT or an inserted SWAP. Let $\mathcal{Y}(C, G)$ denote the solutions that execute all CNOTs while respecting the dependency DAG and hardware connectivity. With $S(y)$ equal to the number of inserted SWAPs, the joint mapping–routing objective is

$$\min_{y \in \mathcal{Y}(C,G)} S(y). \tag{4}$$

Because y includes both the initial mapping and the routing trajectory that determines its subsequent evolution, this objective jointly selects μ_0 and the routing decisions to minimize the final cost.

2.2 Joint autoregressive representation

MARSA models the complete mapping and routing solution with a single encoder–decoder Transformer (Vaswani et al., 2017),

$$p_\theta(y \mid C) = p_\theta(\mu_0 \mid C) \prod_{r=1}^{R} p_\theta(a_r \mid C, \mu_0, a_{<r}), \tag{5}$$

The model first generates the initial mapping and then the complete physical routing trace. In our primary representation, the input is

$$[\texttt{<BOS>}, \texttt{<gate>}, (\mathrm{CNOT}, \mathtt{q}\ell_j, \mathtt{q}k_j)_{j=1}^{M}, \texttt{<EOS>}], \tag{6}$$

and the output is

$$[\texttt{<BOS>}, \texttt{<map>}, \mathtt{q}\mu_0(1{:}P), \texttt{<gate>}, (o_r, \mathtt{q}u_r, \mathtt{q}v_r)_{r=1}^{R}, \texttt{<EOS>}], \tag{7}$$

where <BOS>, <EOS> are structural tokens denoting the beginning and ending of sequence, respectively, ℓ_j, k_j are logical qubits, $o_r \in \{\mathrm{CNOT}, \mathrm{SWAP}\}$, and u_r, v_r are physical qubits. The same qubit token vocabulary is reused throughout: qubit tokens in the input denote logical qubits, whereas those in the mapping prefix and routed operations denote physical qubits. For serialization, μ_0 is extended to a permutation over all P indices by assigning inactive logical placeholders to otherwise unused physical qubits. This keeps the inverse physical-to-logical mapping well defined and allows SWAPs involving idle physical qubits without introducing a separate empty-qubit token.

Because the mapping and routing trace belong to the same generated sequence, a route-level objective has a likelihood path through both.

3 Validity by construction with the route-state automaton

Though the model is trained to learn the syntax and routing semantics of the joint representation, adhering to them perfectly over a long generated trace is difficult: a single invalid decision can invalidate the entire solution. Following the general principle of constrained decoding (Scholak et al., 2021; Koo et al., 2024), we introduce a *route-state automaton* that tracks the evolving routing state and masks invalid tokens during generation.

3.1 AUTOMATON STATE AND TRANSITIONS

For a circuit C and hardware graph G, we define the route-state automaton

$$\mathcal{M}_{C,G} = (\mathcal{S}, \Sigma, \delta, s_0, \mathcal{S}_{\text{acc}}), \tag{8}$$

where Σ is the output vocabulary and $\delta : \mathcal{S} \times \Sigma \rightharpoonup \mathcal{S}$ is a partial transition function. At decoding step t, the state is

$$s_t = (\phi_t, \mu_t, \mathbf{Q}_t, \omega_t), \tag{9}$$

where ϕ_t specifies the class of token expected next (mapping, operation type, or qubit index); μ_t is the partial or complete mapping; ω_t is a pending-operation buffer: it is $\emptyset$ at an operation boundary, becomes (o) after the operation type is generated and (o, u) after the first qubit index, and returns to $\emptyset$ when the second qubit completes the operation and its semantic update is applied. For each logical qubit ℓ, $Q_{\ell,t}$ is an ordered queue of logical qubits participating in its remaining CNOT interactions in circuit order. We refer to this other qubit as the *interaction partner*, so $\text{head}(Q_{\ell,t})$ is the logical qubit paired with ℓ in its next unexecuted CNOT. The collection $\mathbf{Q}_t = (Q_{\ell,t})_{\ell=1}^{n}$ therefore records the remaining circuit dependencies.

As tokens are emitted, ϕ_t advances through the generation phases and ω_t accumulates the tokens of the current operation. Once an operation is complete, its semantic update is applied to μ_t and $\mathbf{Q}_t$. Appendix B gives the complete token-level state machine and transition specification.

At operation boundaries, its semantic updates are:

$$\text{SWAP}(u, v): \quad (\mu_t, \mathbf{Q}_t) \mapsto (\sigma_{uv} \circ \mu_t, \mathbf{Q}_t), \qquad (u, v) \in E, \tag{10}$$

$$\text{CNOT}(u, v): \quad (\mu_t, \mathbf{Q}_t) \mapsto (\mu_t, \text{pop}_{\ell,k}(\mathbf{Q}_t)), \tag{11}$$

where σ_{uv} exchanges the indices of the physical sites u and v, and in Eq. 11 the logical occupants $\ell = \mu_t^{-1}(u)$ and $k = \mu_t^{-1}(v)$ must satisfy

$$(u, v) \in E, \qquad \text{head}(Q_{\ell,t}) = k, \qquad \text{head}(Q_{k,t}) = \ell. \tag{12}$$

Here $\text{pop}_{\ell,k}$ removes these reciprocal queue heads. During mapping, assigning an already used physical qubit is undefined. The automaton reaches an accepting terminal state only at an operation boundary after all queues are empty, indicating that the circuit has been fully executed.

3.2 CONSTRAINED AUTOREGRESSIVE GENERATION

Because δ is partial, each automaton state determines the admissible next-token set

$$\mathcal{L}(s_t) = \{v \in \Sigma : \delta(s_t, v) \text{ is defined}\}. \tag{13}$$

Given Transformer logits $z_\theta(v \mid C, y_{<t})$, MARSA masks tokens outside $\mathcal{L}(s_t)$ and renormalizes,

$$q_\theta(v \mid C, y_{<t}) = \frac{\mathbf{1}\{v \in \mathcal{L}(s_t)\} \exp(z_\theta(v \mid C, y_{<t})/\tau)}{\sum_{u \in \mathcal{L}(s_t)} \exp(z_\theta(u \mid C, y_{<t})/\tau)}, \tag{14}$$

where τ is the sampling temperature.

Proposition 1 (Soundness of accepted generations)**.** *For any generated sequence y,*

$$\delta^*(s_0, y) \in \mathcal{S}_{\text{acc}} \quad \Longrightarrow \quad y \in \mathcal{Y}(C, G), \tag{15}$$

where δ^* is the extension of δ to token sequences.

Proof. Consider the automaton state after each completed physical operation. We maintain the invariant that μ_t is a valid mapping and that $\mathbf{Q}_t$ contains exactly the unexecuted CNOT occurrences of C in dependency-consistent per-qubit order. The invariant holds at initialization. An accepted SWAP satisfies $(u, v) \in E$ and applies the transposition σ_{uv} to the mapping without modifying $\mathbf{Q}_t$, so the invariant is preserved. An accepted CNOT satisfies Eq. 12; hence it executes a hardware-adjacent, dependency-ready CNOT and removes exactly its two reciprocal queue heads without changing the mapping. The invariant therefore holds inductively after every operation. Acceptance requires all queues to be empty. Hence every CNOT in C has been executed exactly once while respecting both the circuit dependencies and hardware connectivity, so $y \in \mathcal{Y}(C, G)$. □

Soundness does not guarantee completion—a rollout that exhausts its generation budget is treated as failed. In implementation, we also suppress simple redundant behavior, such as immediately reversing a SWAP; these restrictions improve search efficiency but are not required for soundness.

4 BREAKING THE TEACHER CEILING

The automaton restricts generation to executable trajectories, but the legal solution space remains enormous. We therefore use a strong search expert to provide a high-quality prior, then replace imitation with exact route feedback once the model can generate complete solutions of its own.

4.1 TWO-AXIS CURRICULUM SUPERVISED LEARNING

We use QMSI, the heuristic search expert from Ren et al. (2024), to generate complete mapping–routing demonstrations. For each training circuit C, the expert provides a solution

$$y^E = [\mu_0^E, a_{1:R}^E]. \tag{16}$$

We train the model with token-level cross entropy to learn from the expert,

$$\mathcal{L}_{\mathrm{CE}}(\theta) = -\frac{1}{\sum_{(C,y^E)\in\mathcal{D}} |y^E|} \sum_{(C,y^E)\in\mathcal{D}} \sum_{t=1}^{|y^E|} \log p_\theta(y_t^E \mid C, y_{<t}^E). \tag{17}$$

To control the training complexity, we gradually increase the circuit width and length in a two-axis staircase curriculum. Let $n(C)$ denote the number of active logical qubits in circuit C, and let $m(C)$ denote its number of CNOTs. At curriculum stage s, the training set is restricted to

$$\mathcal{D}_s = \left\{(C, y^E) \in \mathcal{D} : n(C) \le n_s,\ m(C) \le m_s\right\}, \tag{18}$$

where n_s and m_s are the stage-specific upper bounds on these two quantities. We increase one bound at a time while keeping the other fixed; neither bound decreases, so earlier examples remain available as the difficulty frontier expands. After reaching the full range, we continue training on the full distribution. Appendix C gives the complete schedule.

4.2 VERIFIER-GUIDED SELF-IMPROVEMENT

Once the supervised model can reliably generate complete solutions, we post-train it on trajectories sampled from the model itself. For each circuit C_i, we sample a group of K mapping–routing trajectories

$$y_{i,1}, \ldots, y_{i,K} \sim q_{\theta_{\mathrm{old}}}(\cdot \mid C_i), \tag{19}$$

where $q_{\theta_{\mathrm{old}}}$ denotes the policy under the route-state constraints. Let S_i^E be the SWAP count of the QMSI solution. For a valid trajectory, we define the $[-1, 1]$-normalized SWAP-reduction reward as

$$r_{i,k} = \mathrm{clip}\left(\frac{S_i^E - S(y_{i,k})}{\max\{S_i^E, 1\}}, -1, 1\right), \tag{20}$$

while failed or incomplete trajectories receive a reward of -1.

We compare solutions generated for the same input circuit using the group-relative advantage of GRPO (Shao et al., 2024). Our primary formulation uses

$$\widehat{A}_{i,k} = r_{i,k} - \frac{1}{K}\sum_{j=1}^{K} r_{i,j}. \tag{21}$$

Motivated by Dr. GRPO (Liu et al., 2025), we omit within-group standard-deviation normalization, preserving the explicit per-circuit scaling in Eq. 20 which would otherwise be canceled.

Because both the reward and the advantage are defined for the complete mapping–routing trajectory, we optimize them with a sequence-level clipped objective. Following GSPO (Zheng et al., 2025), we use the length-normalized sequence likelihood ratio

$$\bar{\rho}_{i,k}(\theta) = \exp\left[\frac{1}{|\mathcal{T}(y_{i,k})|} \sum_{t\in\mathcal{T}(y_{i,k})} \log \frac{q_\theta(y_{i,k,t} \mid C_i, y_{i,k,<t})}{q_{\theta_{\mathrm{old}}}(y_{i,k,t} \mid C_i, y_{i,k,<t})}\right], \tag{22}$$

where $\mathcal{T}(y)$ denotes the generated response tokens included in the likelihood. We then optimize

$$\mathcal{J}(\theta) = \mathbb{E}_{i,k}\left[\min\left(\bar{\rho}_{i,k}(\theta)\widehat{A}_{i,k}, \mathrm{clip}(\bar{\rho}_{i,k}(\theta), 1-\epsilon, 1+\epsilon)\,\widehat{A}_{i,k}\right)\right] - \beta\mathcal{R}(\theta), \tag{23}$$

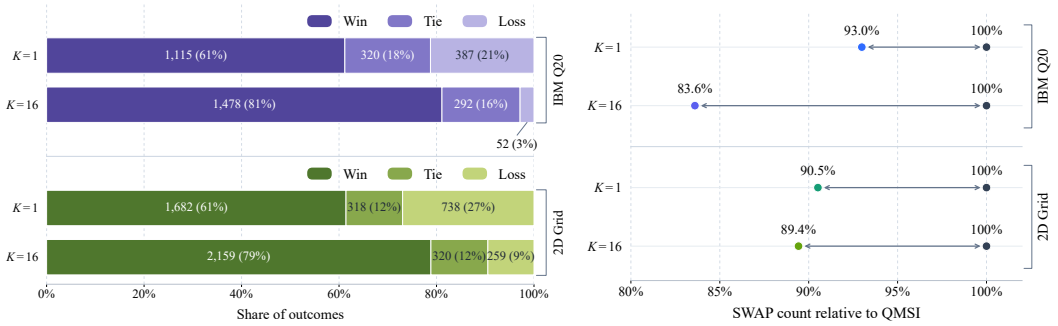
where $\mathcal{R}$ regularizes the policy toward the frozen supervised reference. The exact sequence-level construction, reward and advantage variants, and reference-policy regularization are given in Appendix C. The same trajectory-level advantage is applied to the entire generated solution. A lower-cost trajectory therefore reinforces both its initial mapping and the routing decisions that follow, allowing the final routing objective to optimize the two jointly.

5 Experiments

5.1 Main results

Experimental setup. We evaluate MARSA on IBM Q20 and a 60-qubit 2D grid, training a separate model for each topology. Random circuit evaluation spans 3–20 and 3–60 active logical qubits, respectively, with up to 300 CNOTs; application benchmarks are drawn from QASMBench (Li et al., 2023), the original SABRE suite (Li et al., 2019), MQTBench (Quetschlich et al., 2023), and IBM QX Mapping (Zulehner et al., 2018). Unless stated otherwise, MARSA samples $K = 16$ solutions and returns the valid route with the fewest SWAPs. We report instance-level win/tie/loss and aggregate SWAP reduction $100(1 - \sum_i S_i^{\mathrm{MARSA}} / \sum_i S_i^{\mathrm{base}})\%$. A failure to produce a valid route is counted as a loss; aggregate SWAP reduction is computed over the paired valid cases. Topology illustrations, dataset construction, and training configurations are deferred to Appendix D.

Beyond-teacher performance. On 1,822 held-out Q20 random circuits, MARSA finds a strictly lower-SWAP route than QMSI on 1,478 circuits (81.12%), and matches or improves QMSI on 97.15%, reducing the aggregate SWAP count by 16.40% (Figure 2). Thus, MARSA not only exceeds its training expert on the large majority of instances, but does so with a substantial reduction in overall routing cost. The result reproduces on the substantially larger 2D-grid topology: across 2,738 circuits, MARSA strictly wins on 78.85%, wins or ties on 90.54%, and reduces aggregate SWAP count by 10.57%. The second setting changes both the scale and the connectivity structure of the coupling graph while retaining the same learning framework, demonstrating that MARSA achieves beyond-teacher improvement across distinct topology families. Because QMSI generates the supervised demonstrations, these comparisons directly measure improvement beyond the quality of the training expert.

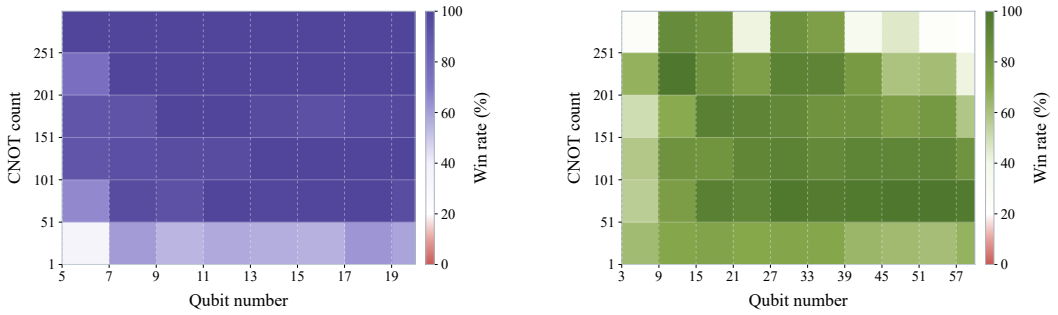


(a) Win/tie/loss rate.

(b) Aggregate SWAP count relative to QMSI.

Figure 2: Beyond-teacher performance on random circuits. On IBM Q20 and 2D grid, MARSA outperforms the QMSI teacher with both greedy ($K = 1$) and best-of-16 ($K = 16$) decoding.

The advantage is broad across problem difficulty rather than concentrated in a narrow subset of circuits. Figure 3 breaks the same evaluation down by circuit width and length. On Q20, MARSA achieves a strict win rate above 90% in every populated bin with at least seven active qubits and more than 50 CNOTs. Strong gains persist across the width–length plane on the 2D grid, with the margin narrowing only in the most demanding long-circuit regime. The heatmaps therefore show that the aggregate improvements in Figure 2 are supported by consistently strong performance across circuit scales, rather than being driven by a particular circuit scale.



(a) IBM Q20. (b) 60-qubit 2D grid.

Figure 3: Difficulty-resolved win rate against QMSI under $K = 16$. Circuits are grouped by active logical-qubit count and CNOT count; color denotes the strict win rate in each bin.

Policy improvement beyond inference-time search. The beyond-teacher performance is not created merely by best-of-16 candidate selection. With a single greedy trajectory ($K = 1$), MARSA already strictly beats QMSI on 61.20% of circuits on Q20 and 61.43% on 2D-grid, reducing aggregate SWAP count by 7.01% and 9.48%, respectively. These substantial greedy gains show that improvement is already present in the learned policy, while verified candidate selection further amplifies it at inference time.

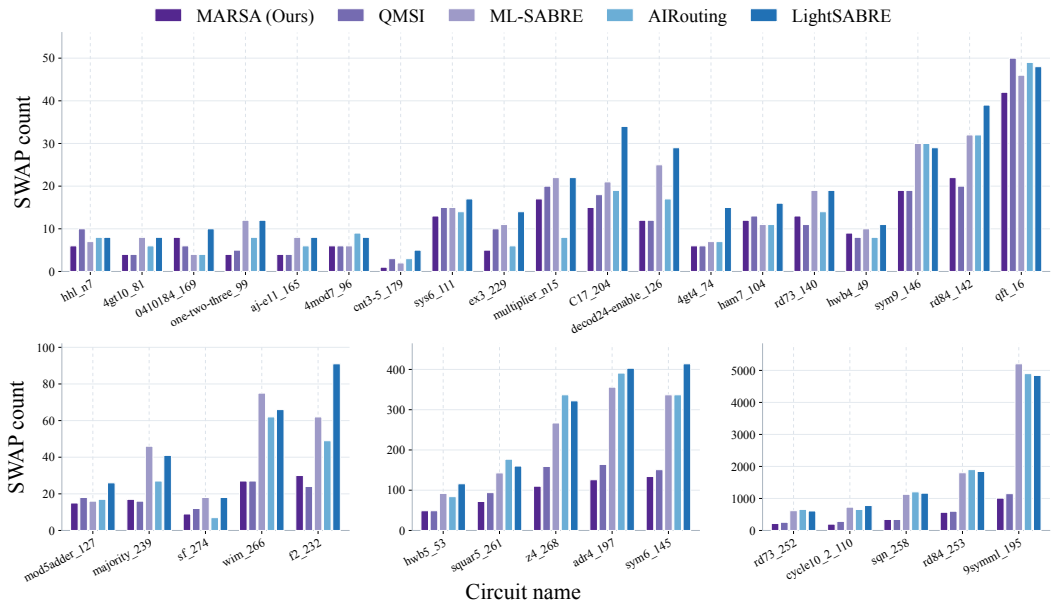


Figure 4: Synthetic-to-real transfer on algorithm circuits for IBM Q20. MARSA outperforms competitive modern baselines across the benchmark; circuits are grouped by SWAP-count range.

Strong synthetic-to-real transfer. Random circuits provide controlled coverage of routing difficulty, but practical compilation operates on quantum algorithm circuits. We therefore train MARSA on a synthetic distribution designed to reproduce their aggregate statistics and evaluate on held-out real algorithm circuits, with every evaluation circuit explicitly excluded from training. We compare against a set of competitive modern baselines, including the QMSI training expert, LightSABRE, ML-SABRE, and Qiskit AIRouting. Detailed baseline configurations are provided in Appendix D.

The transfer to unseen quantum algorithm circuits is strong. On Q20, MARSA reduces aggregate SWAP count by 71.35% relative to LightSABRE, 71.19% relative to ML-SABRE, and 70.86% relative to AIRouting. It also remains beyond its QMSI training expert, reducing aggregate SWAP

count by 11.81%. Figure 4 shows the corresponding per-circuit comparison. MARSA improves across circuits with a broad range of routing costs, with particularly large absolute savings on several high-overhead instances, where reducing routing overhead is most consequential in practice. The same transfer behavior persists on the 2D grid. MARSA reduces aggregate SWAP count by 10.90% relative to LightSABRE, by 4.60% relative to QMSI; the reductions relative to ML-SABRE and adapted AIRouting are 10.02% and 14.86%, respectively. Per-circuit results are given in Appendix D.2. Together, these results show that the improvements learned from synthetic training carry to unseen algorithmic circuits across distinct hardware topologies and competitive routing baselines.

5.2 Ablation study

Joint mapping and routing. We isolate the central modeling choice in MARSA: generating the initial mapping and subsequent routing trajectory as a single solution. We replace MARSA's generated mapping with the QMSI mapping while keeping the learned routing policy and inference budget unchanged. The effect is large: on Q20, this cuts MARSA's aggregate improvement over QMSI from 16.40% to 8.56%; on the 2D grid, it nearly eliminates the gain, from 10.57% to 0.72%. Among circuits completed under both conditions, MARSA's generated mapping leads to a lower routing cost on 71.79% of Q20 circuits and 80.98% of 2D-grid circuits. The initial mapping is therefore not merely a prefix to the routing sequence: MARSA actively chooses mappings that enable substantially better downstream routes, directly supporting the joint mapping–routing formulation.

Table 1: Cumulative ablation of MARSA components.

Topology	Variant	QMSI W/T/L	ΔSWAP ↑	Invalid ↓	
				Mapping	Routing
Q20	Supervised learning, no curriculum	924/391/507	+2.38%	173	9,767
	+ two-axis curriculum	1,084/360/378	+4.68%	208	7,614
	+ route-state automaton	1,211/352/259	+7.52%	0	0
	+ verifier-guided post-training	1,478/292/52	**+16.40%**	0	0
2D grid	Supervised learning, no curriculum	1,628/356/754	+2.72%	1,413	14,309
	+ two-axis curriculum	1,680/352/706	+3.52%	1,284	13,241
	+ route-state automaton	1,814/345/579	+4.67%	0	0
	+ verifier-guided post-training	2,159/320/259	**+10.57%**	0	0

Cumulative contribution of each component. Table 1 traces the successive contributions of the two-axis curriculum, route-state automaton, and verifier-guided self-improvement. For the invalidity columns, counts are aggregated over all 16 sampled candidates per circuit. Under matched training compute, the curriculum increases aggregate SWAP reduction over QMSI, but syntax-only decoding still produces 208/7,614 invalid mapping/routing on Q20 and 1,284/13,241 on the 2D grid. Adding the route-state automaton eliminates all invalid generations and further raises the SWAP reductions to 7.52% and 4.67%. Verifier-guided self-improvement then produces the largest gain, reaching 16.40% on Q20 and 10.57% on the 2D grid. Together, these results show a clear progression: curriculum learning strengthens the initial policy, the automaton ensures valid generation, and self-improvement drives the largest beyond-teacher gains.

6 Conclusion

We introduced MARSA, a learned compiler that treats qubit mapping and routing as a single autoregressive optimization problem. A route-state automaton provides validity by construction, while a two-stage learning strategy first acquires strong routing behavior from expert demonstrations and then improves it through verifier-guided self-improvement. MARSA substantially surpasses its training expert across both Q20 and a 60-qubit 2D coupling graph, with gains already present under greedy decoding and further amplified by verified sampling; the same framework also transfers from synthetic training to unseen quantum algorithm circuits, demonstrating the effectiveness of jointly learning mapping and routing with verifiable self-improvement.

AI USE STATEMENT

In this work, we used generative AI tools to help implement synthetic-data generation, assist in the writing of proofs, provide feedback on research methodology and experiments, implement methods, and clean and reformat datasets. We also used generative AI tools for research ideation and execution, literature retrieval and discovery, drafting portions of the manuscript, and writing and language polishing. We did not use generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims or provide critical ingredients for proving them, propose or refine hypotheses, assist with translation, support qualitative or thematic data analysis, or interpret experimental results. AI-assisted code, experimental outputs, technical claims, and manuscript text were manually reviewed and verified by the authors. The authors take full responsibility for the content of the paper and the reported results.

REPRODUCIBILITY STATEMENT

We provide the methodological and experimental details needed to reproduce the main results throughout the paper and appendix. Sections 2–4 specify the joint mapping–routing formulation, route-state automaton, supervised curriculum learning, and verifier-guided self-improvement procedure, while Section 5 defines the evaluation protocol and reported metrics. Appendix B describes the automaton implementation, guarantees, and independent route verification; Appendix C provides the details of training pipelines; and Appendix D describes the datasets, baselines, and evaluation protocol. We will additionally provide the implementation and experiment code to reviewers and area chairs through an anonymized repository during the review period, following the third option described in the conference guidance for anonymous code sharing.

REFERENCES

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Guiding LLMs the right way: Fast, non-invasive constrained generation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 3658–3673. PMLR, 2024. URL https://proceedings.mlr.press/v235/beurer-kellner24a.html.

Poulami Das, Suhas K. Vittal, and Moinuddin Qureshi. Foresight: Reducing SWAPs in NISQ programs via adaptive multi-candidate evaluations, 2022.

Norhan Elsayed Amer, Walid Gomaa, Keiji Kimura, Kazunori Ueda, and Ahmed El-Mahdy. On the optimality of quantum circuit initial mapping using reinforcement learning. *EPJ Quantum Technology*, 11(1):19, 2024.

Hongxiang Fan, Ce Guo, and Wayne Luk. Optimizing quantum circuit placement via machine learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 19–24, 2022.

Ching-Yao Huang, Chi-Hsiang Lien, and Wai-Kei Mak. Reinforcement learning and DEAR framework for solving the qubit mapping problem. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2022. doi: 10.1145/3508352.3549472.

Terry Koo, Frederick Liu, and Luheng He. Automata-based constraints for language model decoding. In *First Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2407.08103.

David Kremer, Victor Villar, Hanhee Paik, Ivan Duran, Ismael Faro, and Juan Cruz-Benito. Practical and efficient quantum circuit synthesis and transpiling with reinforcement learning, 2024.

Ankit Kulshrestha and Xiaoyuan Liu. CO-MAP: A reinforcement learning approach to the qubit allocation problem, 2026.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm,

Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2411.15124.

Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul McVay, Michael Rabbat, and Yuandong Tian. Beyond A*: Better planning with transformers via search dynamics bootstrapping, 2024. Conference on Language Modeling (COLM 2024).

Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation. *ACM Transactions on Quantum Computing*, 4(2):1–26, 2023.

Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pp. 1001–1014, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362405. doi: 10.1145/3297858.3304023. URL https://doi.org/10.1145/3297858.3304023.

Sanjiang Li, Xiangzhen Zhou, and Yuan Feng. Qubit mapping based on subgraph isomorphism and filtered depth-limited search. *IEEE Transactions on Computers*, 70(11):1777–1788, November 2021. ISSN 2326-3814. doi: 10.1109/tc.2020.3023247. URL http://dx.doi.org/10.1109/TC.2020.3023247.

Yang-Zhi Li, Wen Liu, Guo-Sheng Xu, Mao-Duo Li, Kai Chen, and Shou-Li He. Quantum circuit mapping based on discrete particle swarm optimization and deep reinforcement learning. *Swarm and Evolutionary Computation*, 95:101923, 2025. doi: 10.1016/j.swevo.2025.101923.

Hui Liu, Bingjie Zhang, Yu Zhu, Hanxiao Yang, and Bo Zhao. An efficient qubit mapping method based on dynamic look-ahead strategy. *Scientific Reports*, 14:13118, 2024. doi: 10.1038/s41598-024-64061-0.

Ji Liu, Peiyi Li, and Huiyang Zhou. Not all SWAPs have the same cost: A case for optimization-aware qubit routing. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 709–725, 2022. doi: 10.1109/HPCA53966.2022.00058.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. In *Conference on Language Modeling (COLM)*, 2025.

Abtin Molavi, Amanda Xu, Martin Diges, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. Qubit mapping and routing via MaxSAT. In *55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022. doi: 10.1109/MICRO56248.2022.00077.

Giacomo Nannicini, Lev S. Bishop, Oktay Günlük, and Petar Jurcevic. Optimal qubit assignment and routing via integer programming. *ACM Transactions on Quantum Computing*, 4(1):1–31, 2022. doi: 10.1145/3544563.

Kien X. Nguyen, Ankit Kulshrestha, Ilya Safro, and Xiaoyuan Liu. QAP-router: Tackling qubit routing as dynamic quadratic assignment with reinforcement learning, 2026.

Shuohao Ping, Naren Sathishkumar, Wan-Hsuan Lin, Hanyu Wang, and Jason Cong. A high-performance multilevel framework for quantum layout synthesis, 2025. URL https://arxiv.org/abs/2505.24169.

Matteo G. Pozzi, Steven J. Herbert, Akash Sengupta, and Robert D. Mullins. Using reinforcement learning to perform qubit routing in quantum compilers. *ACM Transactions on Quantum Computing*, 3(2), 2022. doi: 10.1145/3520434.

Nils Quetschlich, Lukas Burgholzer, and Robert Wille. Mqt bench: Benchmarking software and design automation tools for quantum computing. *Quantum*, 7:1062, 2023.

Xiangyu Ren, Tianyu Zhang, Xiong Xu, Yi-Cong Zheng, and Shengyu Zhang. Leveraging machine learning for quantum compilation optimization. In *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024. doi: 10.1145/3649329.3663510.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9895–9901. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.779. URL https://aclanthology.org/2021.emnlp-main.779/.

Irfansha Shaik and Jaco van de Pol. Optimal layout synthesis for deep quantum circuits on NISQ processors with 100+ qubits. In *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*, volume 305 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 26:1–26:18, 2024. doi: 10.4230/LIPIcs.SAT.2024.26.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=lNAyUngGFK.

Animesh Sinha, Utkarsh Azad, and Harjinder Singh. Qubit routing using graph neural network aided monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 9935–9943, 2022. doi: 10.1609/aaai.v36i9.21231.

Bochen Tan and Jason Cong. Optimal layout synthesis for quantum computing. In *Proceedings of the 39th IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020. doi: 10.1145/3400302.3415620.

Wei Tang, Yiheng Duan, Yaroslav Kharkov, Rasool Fakoor, Eric Kessler, and Yunong Shi. Alpharouter: Quantum circuit routing with reinforcement learning and tree search. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 930–940, 2024. doi: 10.1109/QCE60285.2024.00112.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Robert Wille, Lukas Burgholzer, and Alwin Zulehner. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *Proceedings of the 56th Annual Design Automation Conference (DAC)*, 2019. doi: 10.1145/3316781.3317859.

Yueran Zhao, Kaiqi Li, Ziying Guo, and Jialin Zhang. A graph-based deep reinforcement learning framework for quantum circuit mapping with look-ahead rewards and biased exploration. In *2025 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–9, 2025. doi: 10.1109/ICCAD66269.2025.11240780.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization, 2025. URL https://arxiv.org/abs/2507.18071.

Xiangzhen Zhou, Yuan Feng, and Sanjiang Li. A monte carlo tree search framework for quantum circuit transformation. In *Proceedings of the 39th IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020. doi: 10.1145/3400302.3415621.

Xiangzhen Zhou, Yuan Feng, and Sanjiang Li. Quantum circuit transformation: A monte carlo tree search framework. *ACM Transactions on Design Automation of Electronic Systems*, 27(6), 2022. doi: 10.1145/3514239.

Henry Zou, Matthew Treinish, Kevin Hartman, Alexander Ivrii, and Jake Lishman. Lightsabre: A lightweight and enhanced sabre algorithm, 2024. URL https://arxiv.org/abs/2409.08368.

Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the ibm qx architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1226–1236, 2018.

A Related work

Qubit Mapping and Routing. A large body of work has examined qubit mapping and routing problems from different perspectives. Here we highlight work of which the settings are consistent with ours, namely SWAP-based approaches on connectivity-constrained quantum processors.

Among early representative approaches, heuristic and search-based methods form an important line of work. SABRE (Li et al., 2019) introduced an efficient bidirectional heuristic based on front-layer and look-ahead distances and has since become a standard baseline for scalable qubit routing. Later work improves routing quality through bounded-depth search (Li et al., 2021), Monte Carlo Tree Search (MCTS) (Zhou et al., 2020; 2022), optimization-aware cost functions (Liu et al., 2022), multi-candidate search (Das et al., 2022), and dynamic look-ahead (Liu et al., 2024). Another important line formulates qubit mapping and routing as exact optimization problems, solved via SAT/SMT (Wille et al., 2019; Tan & Cong, 2020; Shaik & van de Pol, 2024), integer programming (Nannicini et al., 2022), and MaxSAT (Molavi et al., 2022). In general, heuristic methods are efficient but often introduce suboptimality, while exact solvers offer stronger solution guarantees at a higher computational cost.

With the rapid development of machine learning (ML), learning-based methods have been explored to better balance solution quality and compilation cost. A subset of work focuses only on initial mapping, using reinforcement learning (RL) to map logical qubits (Elsayed Amer et al., 2024; Kulshrestha & Liu, 2026). However, most learning-based work focuses on routing. Early methods use deep Q-learning (DQN) to select SWAPs (Pozzi et al., 2022) or combine graph representations with MCTS (Sinha et al., 2022). LARBE (Zhao et al., 2025) further improves RL-based routing with a GNN, look-ahead rewards, and biased exploration. Meanwhile, another line adopts Transformer-based models to capture longer circuit dependencies. QCO (Ren et al., 2024) applies a sequence-to-sequence Transformer to predict short SWAP sequences from upcoming gates using search-generated supervision. AlphaRouter (Tang et al., 2024) combines a Transformer policy with MCTS and iterative policy improvement. Both methods, however, operate with relatively short circuit contexts. More recently, QAP-Router (Nguyen et al., 2026) formulates routing as a dynamic quadratic assignment problem and incorporates this structure into a Transformer-based RL policy. The aforementioned three Transformer-based methods all focus on routing and rely on a given initial mapping. Hybrid methods have also been explored to address both stages. Huang et al. (2022) learn initial mappings using an RL-trained sequence-to-sequence model and combine this with the DEAR framework for subsequent routing. Similarly, Li et al. (2025) use discrete particle swarm optimization for initial mapping and a separately trained Double DQN policy for routing, leaving the two stages as separate optimization modules.

Taken together, existing learning-based methods either focus on single stage of the mapping pipeline or combine initial mapping and routing through separate models. In contrast, our approach represents the initial mapping and the complete routing trajectory within a unified token sequence and autoregressively generates both using a single Transformer model. This allows initial mapping and SWAP decisions to be learned and optimized simultaneously under the same circuit context. We further improve this joint policy through self-improvement after the initial training stage.

State-Dependent Constrained Generation. For problems with underlying constraints, autoregressive generation does not inherently guarantee the correctness. To address this issue, constrained generation methods restrict the admissible next-token set during decoding. For example, PICARD (Scholak et al., 2021) incrementally parses partial generations and rejects tokens that cannot lead to valid structured outputs. DOMINO (Beurer-Kellner et al., 2024) improves the efficiency of constrained decoding while accounting for the mismatch between formal constraints and subword tokenization, and automata-based decoding (Koo et al., 2024) uses finite-state and related automata constructions to efficiently enforce formal language constraints. These methods demonstrate how carefully designed mechanisms can restrict generation to valid continuations. To adapt to the qubit mapping and routing problem, MARSA uses a route-state automaton to update the logical-to-physical mapping and remaining gate dependencies after each generated operation and then derives the admissible next actions from the resulting state.

Verifiable Self-Improvement. A complementary line of work improves autoregressive models using feedback on their own generations. Searchformer (Lehnert et al., 2024) represents both the prob-

lem and the search dynamics of A^* as token sequences, and subsequently applies search-dynamics bootstrapping to learn shorter search traces than those used for initial supervision. ReST^{EM} (Singh et al., 2024) similarly alternates between generating candidate solutions, filtering them using externally verifiable feedback, and retraining on the retained samples. More recent RL approaches directly optimize verifiable rewards: DeepSeekMath introduces Group Relative Policy Optimization (GRPO) (Shao et al., 2024), while Tülu 3 (Lambert et al., 2025) develops Reinforcement Learning with Verifiable Rewards (RLVR). MARSA brings this verifiable self-improving paradigm to joint qubit mapping and routing, where generated solutions can be exactly checked for feasibility and assigned an objective value through their SWAP count. This enables the model to improve both mapping and routing decisions from its own verified solutions.

B Route-state automaton

The main text defines the route-state automaton and its semantic transitions. Here we describe its token-level implementation, dependency tracking, and search restrictions in greater detail.

B.1 Incremental state machine

The decoder maintains the automaton state in Eq. 9 progressively throughout generation. Figure 5 gives the phase-transition structure. Table 2 summarizes the grammar phases for the autoregressive representation.

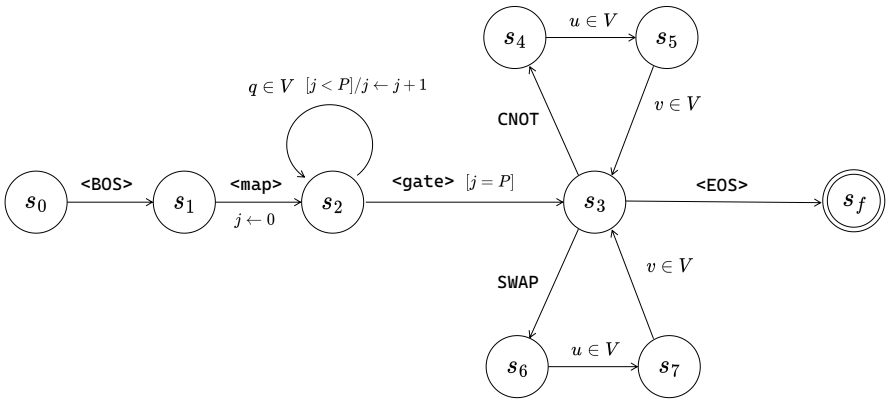


Figure 5: Token-level phases of the route-state automaton. Each node denotes the automaton phase before the next token is emitted. Edge labels indicate admissible token classes. Completing the second physical-qubit token applies the corresponding CNOT or SWAP state update and returns the automaton to the operation phase. <EOS> is admissible only when all CNOT queues are empty.

Table 2: Generation phases of the route-state automaton.

Phase	Admissible token class	State update
Start	<BOS>	start sequence
Mapping marker	<map>	enter mapping
Mapping	unused physical qubit	append one permutation entry
Gate marker	<gate>	enter routing
Operation	CNOT / SWAP / <EOS>	choose operation type
First endpoint	admissible physical qubit	store u
Second endpoint	admissible partner of u	execute operation

During mapping generation, qubit indices that have already been used are masked, so the completed mapping block is guaranteed to be a permutation of the physical qubits. During routing, the decoder stores the current mapping and inverse mapping, the remaining interaction queues, and any partially generated operation. For a pending CNOT, the first qubit must hold a logical qubit whose reciprocal

queue-head partner is dependency-ready and physically adjacent; after the first qubit is chosen, the second qubit is restricted to that partner's physical location. Completing the CNOT removes the corresponding heads from both interaction queues. For a SWAP, the two physical qubits must be adjacent in the hardware graph; completing the operation exchanges their logical occupants. The <EOS> token is admissible only at an operation boundary after all CNOTs have been consumed.

Running example. Consider three physical qubits arranged as a path p_0–p_1–p_2 and the logical circuit $C = [\mathrm{CNOT}(\ell_0, \ell_2)]$. Suppose the generated mapping is $\mu(\ell_0) = p_0$, $\mu(\ell_1) = p_1$, and $\mu(\ell_2) = p_2$. The corresponding dependency queues are $Q_0 = [\ell_2]$, $Q_1 = []$, and $Q_2 = [\ell_0]$. One valid generated sequence is

$$[\texttt{<BOS>}, \texttt{<map>}, q_0, q_1, q_2, \texttt{<gate>}, \mathrm{SWAP}, q_1, q_2, \mathrm{CNOT}, q_0, q_1, \texttt{<EOS>}].$$

Here the q_i tokens following <map> and <gate> denote physical-qubit indices. After <map>, the automaton is in the mapping phase with $\omega = \varnothing$; the three subsequent qubit tokens construct μ. After <gate>, it enters the operation phase. The routing portion then evolves as follows.

Last emitted token	ϕ (next phase)	ω	Semantic state/change
<gate>	operation	$\varnothing$	$\mu = (0, 1, 2)$, $Q = ([2], [], [0])$
SWAP	first qubit	(SWAP)	unchanged
q_1	second qubit	$(\mathrm{SWAP}, 1)$	unchanged
q_2	operation	$\varnothing$	$\mu = (0, 2, 1)$; Q unchanged
CNOT	first qubit	(CNOT)	unchanged
q_0	second qubit	$(\mathrm{CNOT}, 0)$	unchanged
q_1	operation	$\varnothing$	μ unchanged; $Q = ([], [], [])$
<EOS>	accept	$\varnothing$	routing complete

The automaton context is cached separately for each rollout and updated incrementally as new tokens are generated. Without caching, reconstructing the state by replaying the prefix after every token would require $\sum_{t=1}^{T} O(t) = O(T^2)$ prefix-processing work for an output of length T; retaining the current context reduces this overhead to $O(T)$. The cached state requires $O(P + M)$ memory for the mapping, inverse mapping, and remaining interaction queues. Legality-mask construction incurs additional per-step work: in our implementation, identifying executable CNOT choices may scan the P physical qubits and their hardware neighbors, requiring $O(P\Delta)$ time at the relevant routing decision points, where Δ is the maximum hardware degree.

B.2 Dependency queues and repeated interactions

For each logical qubit ℓ, the queue Q_ℓ is built by scanning the input CNOT sequence in order and appending the interaction partner of every CNOT involving ℓ. Repeated CNOTs between the same logical pair remain distinct occurrences in the queues.

Proposition 2 (Frontier characterization)**.** *An unexecuted CNOT occurrence between logical qubits (ℓ, k) is dependency-ready if and only if*

$$\mathrm{head}(Q_\ell) = k, \qquad \mathrm{head}(Q_k) = \ell. \tag{24}$$

Proof. If the occurrence is at the head of both queues, no earlier unexecuted CNOT remains on either participating logical wire, so all of its predecessors have been executed. Conversely, if the occurrence is not at the head of at least one queue, an earlier unexecuted CNOT remains on that logical wire and the occurrence is not dependency-ready. Repeated interactions preserve their circuit order in both queues, so the reciprocal-head condition also distinguishes repeated occurrences. □

The queue representation therefore enforces the dependency DAG used in the main text without explicitly maintaining or repeatedly traversing the full DAG during decoding.

B.3 Cost-preserving pruning

Beyond semantic legality, the decoder removes two classes of redundant routing trajectories. First, it uses an *eager-CNOT* rule: whenever at least one dependency-ready CNOT is executable under the

current mapping, a CNOT must be executed before another SWAP is inserted. Second, it suppresses an immediate reversal of the preceding SWAP. Both restrictions reduce the search space without changing the minimum achievable SWAP count.

Proposition 3 (Cost-preserving eager execution)**.** *Let s be a routing state in which a dependency-ready CNOT is executable under the current mapping. For every valid completion from s, there exists a valid completion with no larger SWAP count that executes an executable CNOT before inserting the next SWAP.*

Proof. Executing a CNOT changes the dependency queues but not the logical-to-physical mapping and incurs zero SWAP cost. Consider a valid completion that inserts SWAPs before executing a CNOT occurrence that is already executable in s. Move that occurrence before the intervening SWAPs and remove it from its original position. It is legal in s by assumption, while the subsequent SWAP sequence remains physically well defined because the mapping immediately after the moved CNOT is unchanged. Executing a ready CNOT only removes a dependency and therefore cannot invalidate a later interaction that was already legal. Repeating this exchange for any newly exposed executable CNOTs yields a valid completion in which executable CNOTs precede the next SWAP, with the same SWAP count. □

Proposition 4 (Dominance of immediate SWAP reversal)**.** *No SWAP-minimal route contains two consecutive SWAPs on the same hardware edge with no CNOT between them.*

Proof. Two consecutive applications of the same transposition restore the previous mapping. With no CNOT between them, the dependency queues are also unchanged. Removing both SWAPs therefore reaches the same routing state with two fewer SWAPs. □

Propositions 3 and 4 show that these restrictions remove redundant trajectories while preserving at least one SWAP-optimal route. Our implementation also supports a stronger optional mapping-history guard that suppresses revisiting any mapping encountered since the previous CNOT. Because the queues are unchanged between CNOTs, a repeated mapping closes a positive-cost SWAP cycle and can likewise be removed. We treat this stronger guard as an optional search restriction rather than part of the core soundness argument.

B.4 CNOT DIRECTION RECOVERY

Each routed operation occupies three tokens: an operation type followed by two physical-qubit indices. For routing, a CNOT is serialized without direction: the generated route records the two interacting physical qubits but does not separately encode which operand is the control and which is the target. This is sufficient for dependency readiness, hardware adjacency, and SWAP insertion on the undirected coupling graphs considered here.

After generation, the control–target order is reconstructed deterministically from the source circuit. For an emitted CNOT on physical qubits (u, v), the current inverse mapping identifies the corresponding logical pair. The dependency queues determine which pending interaction between that pair is executed next, and hence which source-circuit CNOT occurrence is being consumed. The control–target order stored for that occurrence is then mapped to the current physical locations. Repeated CNOTs between the same logical pair remain unambiguous because their occurrence order is preserved on both logical wires. Thus, although direction is omitted from generation, the directed routed circuit can be reconstructed exactly from the source circuit and the routing state.

B.5 INDEPENDENT ROUTE VERIFICATION

The implementation applies two complementary correctness checks. First, serialized expert targets are replayed through the same production transition rules used for constrained generation, ensuring that supervised demonstrations satisfy the automaton grammar and routing constraints. Second, completed model outputs are replayed by a separate route verifier that reconstructs the evolving mapping, checks every SWAP against the hardware graph, verifies that each emitted CNOT corresponds to a dependency-ready source occurrence, and confirms completion of the logical circuit. Constrained generation therefore enforces validity online, while independent replay provides a separate end-to-end check of accepted outputs.

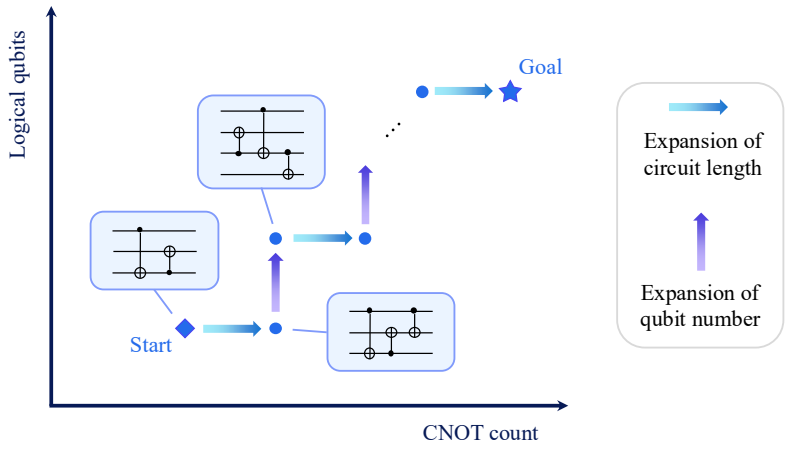


Figure 6: An illustration of the two-axis supervised curriculum framework. Starting from an initial setting, either the number of logical qubits or the CNOT count is increased at each stage, progressively incorporating more challenging circuits into the training set. After the target setting is reached, training continues for an additional 10 rounds before termination.

C TRAINING

We use the same learning framework on both hardware topologies. The hardware graph, supported circuit width, and QMSI demonstrations are topology-specific, but the optimization procedure is shared. Separate policies are trained for each topology. Supervised learning uses two data regimes—random circuits for the controlled beyond-teacher experiments and synthetic algorithm-like circuits for transfer—followed by a common verifier-guided post-training procedure. We summarize the methodological settings here; exact dataset manifests, launch arguments, and checkpoint identities are included with the released code.

C.1 TWO-AXIS SUPERVISED CURRICULUM

The supervised stage trains on complete joint mapping–routing demonstrations produced by QMSI using the token-mean cross-entropy objective in Eq. 17. In both data regimes, the two-axis curriculum of Eq. 18 expands the admissible circuit width and CNOT length monotonically, increasing one frontier at a time while retaining examples from earlier stages. CNOT frontiers are determined from empirical quantiles of the corresponding training distribution, and training continues for ten full-distribution rounds after the complete support is reached.

For the *random* regime, circuits contain up to 300 CNOTs and are sampled approximately uniformly by input-CNOT token mass. The supported width follows the topology: 3–20 active qubits on Q20 and 3–60 on the 60-qubit graph. For the *synthetic algorithm* regime, circuits contain up to 1,000 CNOTs and are drawn from the structured synthetic generator described in Appendix D.1. Exact stage boundaries, batch geometry, optimizer schedules, and distributed-training settings are recorded in the released run configurations.

C.2 VERIFIER-GUIDED SELF-IMPROVEMENT

Post-training starts from the supervised policy and keeps a frozen supervised checkpoint as the reference policy. For each circuit C_i, the constrained rollout policy samples a group of complete mapping–routing trajectories, the independent verifier evaluates their validity and SWAP counts, and the normalized reward in Eq. 20 is used for the policy update. Constant-reward groups contain no within-prompt preference signal and are omitted from the gradient update.

Reward and group-relative advantage variants. Equation 20 is our primary reward. Normalization by the expert cost makes the improvement signal comparable across circuits with different routing overhead while retaining the magnitude of the improvement. A simpler alternative is

$$r_{i,k} = \begin{cases} +1, & S(y_{i,k}) < S_i^E, \\ 0, & S(y_{i,k}) = S_i^E, \\ -1, & S(y_{i,k}) > S_i^E, \end{cases} \tag{25}$$

with failed or incomplete trajectories also assigned -1. This variant records whether a route beats the reference but discards the magnitude of the difference. It also assigns the same reward to an incomplete rollout and to any completed route worse than the reference. In our experiments on real circuits, the finer normalized signal was useful both for suppressing failed generations and for driving larger SWAP reductions, and we therefore adopt Eq. 20 as the default reward.

Given rewards within a group, our primary advantage is the mean-centered form in Eq. 21. The implementation also supports the standard GRPO normalization (Shao et al., 2024),

$$\widehat{A}_{i,k}^{\mathrm{std}} = \frac{r_{i,k} - \bar{r}_i}{\mathrm{std}(r_{i,1:K}) + \varepsilon_A}, \qquad \bar{r}_i = \frac{1}{K}\sum_{j=1}^{K} r_{i,j}, \tag{26}$$

as a viable alternative.

Reference-policy regularization. Writing the average response log probability as

$$\bar{\ell}_\theta(y \mid C) = \frac{1}{|\mathcal{T}(y)|} \sum_{t \in \mathcal{T}(y)} \log q_\theta(y_t \mid C, y_{<t}), \tag{27}$$

the primary reported runs use the squared sequence-level reference regularizer

$$\mathcal{R}(\theta) = \mathbb{E}_{i,k}\left[\left(\bar{\ell}_\theta(y_{i,k} \mid C_i) - \bar{\ell}_{\mathrm{ref}}(y_{i,k} \mid C_i)\right)^2\right]. \tag{28}$$

This is the term weighted by β in Eq. 23, which we use as a reference-policy regularizer rather than an exact KL divergence. The trainer also supports the sequence-level analogue of the reverse-ratio regularizer used in GRPO,

$$\mathcal{R}_{\mathrm{rev}}(\theta) = \mathbb{E}_{i,k}\left[\exp\left(\bar{\ell}_{\mathrm{ref}}(y_{i,k} \mid C_i) - \bar{\ell}_\theta(y_{i,k} \mid C_i)\right) - \left(\bar{\ell}_{\mathrm{ref}}(y_{i,k} \mid C_i) - \bar{\ell}_\theta(y_{i,k} \mid C_i)\right) - 1\right]. \tag{29}$$

Both regularizers discourage large departures from the supervised reference on the same constrained action support.

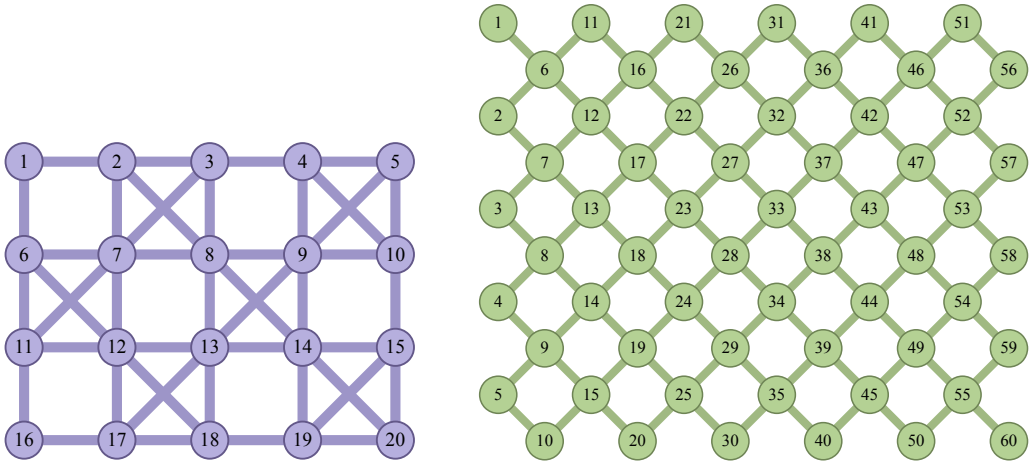
D Experiments

D.1 Experimental setup

Hardware topologies. The two hardware coupling graphs are illustrated in Figure 7. IBM Q20 contains 20 qubits and 43 undirected couplings. The second topology is a substantially larger 2D grid with 60 qubits and 99 undirected couplings. We train a separate MARSA policy for each topology.

Dataset construction. For the controlled random circuit experiments, dataset circuits contain 1–300 CNOTs and span 3–20 active logical qubits on Q20 and 3–60 on the 2D grid. The complete generated corpora contain 912,851 and 1,369,277 circuits, respectively. Final random circuit results are evaluated on held-out dataset split, with 1,822 Q20 circuits and 2,738 2D grid circuits respectively. For synthetic-to-real transfer, we construct a separate synthetic corpus of 912,851 circuits for each topology, spanning 2–20 active logical qubits and 1–1000 CNOTs. Evaluation uses the application benchmarks described in the main text.

The synthetic-to-real training data is created by sampling from circuit distribution designed to mimic the pattern of actual quantum algorithm circuits. We only use aggregated statistics to determine the frozen synthetic mixture; individual benchmark circuits are never used as training examples. We reject exact benchmark matches under both directed and endpoint-normalized CNOT representations, reject generated circuits containing a 64-CNOT contiguous segment from the held-out benchmark, and remove exact duplicates across data splits. The complete generator definitions and sampling configuration are included with the released code.



(a) Q20. (b) 60-qubit 2D grid.

Figure 7: Hardware coupling graphs, where circles represent physical qubits and edges represent couplers. CNOT and SWAP gates can only be applied between directly connected qubits.

Table 3: Training configuration.

Configuration	Supervised learning	Verifier-guided self-improvement
Training GPUs	8	8
Batch size per rank	16	2
Gradient accumulation	2	4
Learning rate	10^{-4}	10^{-6}
LR schedule	cosine	cosine
Warmup steps	2,000	200
Maximum length cap	8,192	8,192
Candidates per prompt	–	16
Sampling temperature	–	1
PPO epochs	–	4
PPO minibatch size	–	4
Clip parameter ϵ	–	0.2
Reference coefficient β	–	0.01
Policy-ratio granularity	–	sequence
Reference regularizer	–	squared
Constant-reward group filter	–	yes

Model architecture. MARSA uses an encoder–decoder Transformer with nine encoder and nine decoder layers, hidden dimension 768, and four attention heads per layer. We use RMS normalization, rotary positional embeddings, and dropout 0.1. The implementation supports sequences up to 2^{16} positions, while the reported training runs use an 8,192-token generation limit.

Training configurations. Table 3 summarizes the optimization configuration used in the reported experiments. We use the same optimization hyperparameters across the two hardware topologies; only the supported circuit widths, length distribution, and resulting curriculum frontiers depend on the dataset.

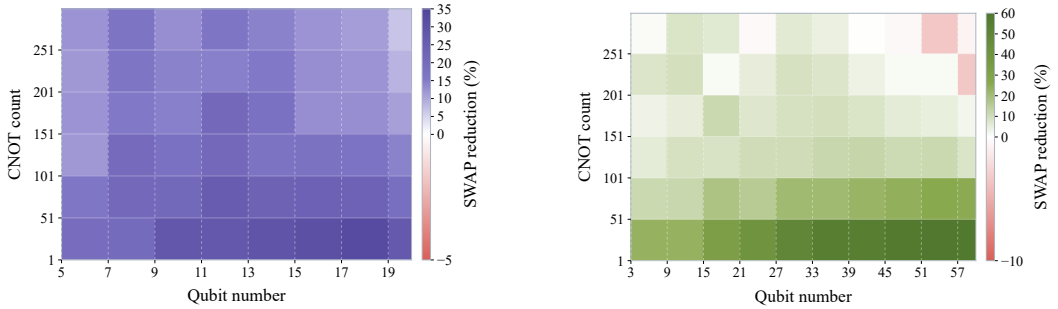
Baseline configurations. QMSI (Ren et al., 2024) is the training expert and primary beyond-teacher reference. For LightSABRE (Zou et al., 2024), we use Qiskit 2.4.1 and the default setting in the paper: `swap_trials=20`, `layout_trials=20`, `max_iterations=4`, `heuristic=decay`. For ML-SABRE (Ping et al., 2025), we use the released code with ten multilevel cycles, initial embedding enabled, 500 coarsest-level search trials, and 100 interpolation attempts. For Qiskit AIRouting (Kremer et al., 2024), we use optimized-layout mode with SWAP

count as the sole optimization objective (`optimization_preferences=[''n_cnots'']`) and the default level-2 search budget: `full_its=16`, `its=16`, `reps=8`, `runs=1`, and `max_time=30`. Its optimized-layout mode uses `SabreLayout` with 16 layout and 16 SWAP trials. To retain an explicit routing trace for exact SWAP accounting, we disable output-DAG fusion and cancellation in the local AIRouting implementation.

Inference protocol. Unless otherwise stated, the sampled MARSA results in the paper use $K = 16$ candidates, while greedy results use a single deterministic trajectory. Candidate selection depends only on exact route validity and verified SWAP count. Circuits with at most 1000 CNOTs are routed in a single invocation. Longer application circuits are processed sequentially in 1000-CNOT windows: the final mapping of each completed window becomes the fixed starting mapping of the next, and the concatenated trace is therefore a full route of the complete source circuit.

D.2 Additional results

Difficulty-resolved SWAP reduction. Figure 8 complements the win-rate heatmaps in Figure 3 with aggregate SWAP reduction over the same width–length bins. Win rate measures how broadly MARSA improves, whereas aggregate reduction captures the magnitude of the routing-cost improvement within each region. On Q20, the reduction is positive and relatively smooth across nearly the entire difficulty plane, indicating that the aggregate gain is not driven by a narrow subset of circuit sizes. The 60-qubit grid shows a more heterogeneous pattern, but positive reductions still cover most width–length regimes, with the strongest gains concentrated in regions with larger qubit numbers. Only a small number of bins at the largest widths and CNOT counts approach zero or become slightly negative. Together with Figure 3, these results show that MARSA's improvement over QMSI is both widespread across instances and substantial in aggregate routing cost.



(a) Q20.

(b) 60-qubit 2D grid.

Figure 8: Aggregate SWAP reduction over QMSI across circuit width and CNOT count. Bins follow the difficulty partition used in Figure 3.

Synthetic-to-real transfer on the 2D grid. Figure 9 gives the corresponding per-circuit synthetic-to-real comparison on the 60-qubit 2D grid, complementing the Q20 results in Figure 4. MARSA achieves moderate gains among the smaller instances where all methods incur relatively similar routing costs. The separation becomes more pronounced on many of the higher-overhead circuits: MARSA frequently produces the lowest SWAP count and in several cases reduces the gap substantially relative to both its QMSI teacher and the external baselines. This concentration of gains on high-overhead circuits is particularly valuable in practice, since larger absolute SWAP reductions remove more additional two-qubit operations, where routing overhead can have the greatest impact on execution fidelity.

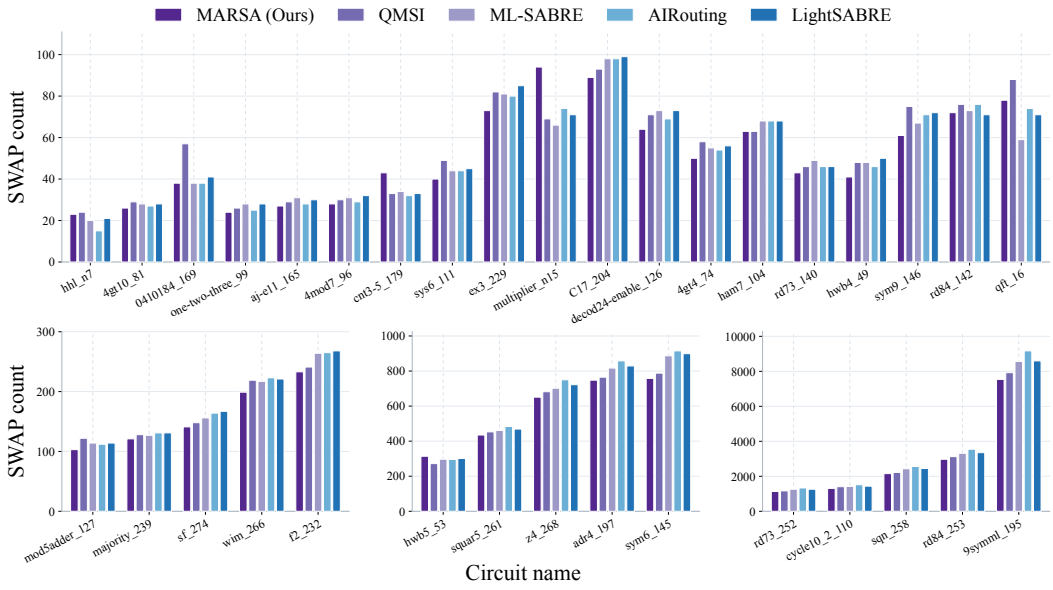


Figure 9: Synthetic-to-real transfer on algorithm circuits for the 2D grid. MARSA outperforms competitive modern baselines across the benchmark; circuits are grouped by SWAP-count range.