CARL: Crosstalk-Aware Quantum Compilation Based on Reinforcement Learning

Xiangzhi Zhang*, Yu Liu*, Yingling Mao†, Xu Xu‡, Yuanyuan Yang‡
*Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR
†Department of Electrical and Computer Engineering, The Hong Kong Polytechnic University, Hong Kong SAR
‡Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, USA

***Abstract*—Quantum Computing (QC) is promising for solving complex problems that are intractable for classical computers. However, QC on current quantum processors faces a critical challenge in achieving high execution fidelity due to decoherence and accumulated gate errors, limiting its practical applicability. Quantum circuit compilation (QCC), which transforms high-level quantum circuits into hardware-compliant implementations, directly affects execution fidelity. Existing work either neglects hardware constraints such as qubit connectivity or does not explicitly optimize for execution fidelity, which can significantly degrade circuit fidelity. This paper investigates the problem of circuit compilation, which jointly considers qubit connectivity constraints and optimizes execution fidelity explicitly. To address the problem, this paper presents a reinforcement learning–based QCC approach named CARL. In particular, CARL unifies mapping and routing into a single crosstalk-aware optimization pipeline to mitigate hardware noise. Extensive evaluations on real-world benchmark quantum circuits demonstrate that the proposed approach achieves significantly higher circuit fidelity than state-of-the-art baselines.**

I. Introduction

Quantum Computing (QC) has the potential to revolutionize fields where classical systems falter [1]. Beyond its well-known applications in integer factorization and database search, the technology is becoming critical for quantum simulation in material science [2] and drug discovery [3]. Substantial advances have been made in the development of quantum computing devices, particularly by major industrial research groups, in recent years. For example, in late 2025, IBM introduced its Nighthawk processor with enhanced qubit coupling and improved coherence properties [4]. Similarly, Google showcased its Willow chip, which incorporates advanced architectural and error control techniques to support more reliable and scalable quantum operations [5].

Despite these advances, executing quantum circuits with high fidelity on noisy intermediate-scale quantum (NISQ) hardware remains a fundamental challenge. In addition to intrinsic gate and decoherence errors [6], physical qubits on a chip can typically interact only with a limited set of neighboring qubits due to hardware connectivity constraints, where neighboring qubits are connected by couplers [7], which is particularly pronounced in leading superconducting qubit architectures. This restricted connectivity requires qubits to be routed into adjacent physical locations in order to execute two-qubit gates. Moreover, experimental studies have shown that quantum devices experience significant crosstalk when two-qubit gates are executed simultaneously on nearby couplers. The fidelity of two-qubit gates can degrade significantly when neighboring couplers are activated in parallel [8], [9]. Consequently, quantum circuits at the logical level cannot be executed directly on hardware and must first undergo processes to enforce hardware constraints and optimize execution fidelity.

Quantum circuit compilation (QCC) transforms a logical circuit into a hardware-compliant implementation while explicitly optimizing for noise resilience on NISQ hardware [10]–[12]. QCC has three core functionalities: circuit transformation, qubit mapping, and qubit routing. Circuit transformation optimizes the circuit by replacing subcircuits with functionally equivalent implementations that reduce gate count or depth, qubit mapping assigns logical qubits to physical qubits on the target device, and qubit routing inserts additional operations to satisfy hardware connectivity constraints when gate operations are not directly supported. In the presence of crosstalk, QCC should also take the spatial and temporal patterns of gate execution into consideration in order to mitigate interference between simultaneous operations. These components are tightly coupled, as decisions made in one component directly influence the others, and together they determine the final gate count, circuit depth, and execution time, ultimately impacting the fidelity of quantum circuit execution on NISQ hardware.

Existing approaches in quantum circuit compilation rarely consider circuit transformation, qubit mapping, and qubit routing jointly, and largely ignore the impact of crosstalk when optimizing execution fidelity. First, many methods focus on qubit mapping and routing for connectivity satisfaction without incorporating circuit transformation, resulting in unnecessary gate overhead and suboptimal performance. Second, most methods optimize proxy metrics such as gate count or depth without modeling execution fidelity or crosstalk. While such metrics may improve, aggressive parallelism can amplify crosstalk and consequently degrade circuit fidelity in practice. Third, existing compilers largely rely on handcrafted heuristics whose effectiveness diminishes rapidly with system scale, as they struggle to capture the complex, non-local, and schedule-dependent effects introduced by crosstalk.

To address these limitations, this paper studies crosstalk-aware quantum circuit compilation by jointly considering circuit transformation, qubit mapping, and qubit routing, with

This work was supported in part by the NSF under grant numbers CNS-2403202. Yu Liu is the corresponding author.

the explicit goal of maximizing execution fidelity on NISQ hardware. We propose an RL framework for the crosstalk-aware joint optimization of the entire quantum compilation, including mapping, routing, and circuit transformation. To address the computational complexity of large-scale compilation, our framework adopts a hierarchical action space that coordinates qubit routing and circuit transformation decisions. To capture long-term dependencies, we introduce a decay-based lookahead mechanism, which guides the agent to make decisions that remain beneficial for downstream circuit execution. By jointly balancing gate count reduction and circuit depth minimization, the proposed framework directly maximizes execution fidelity and achieves substantially improved performance.

The main contributions of this work are as follows: First, we formulate quantum circuit compilation as a fidelity-driven joint optimization problem that simultaneously accounts for circuit transformations, qubit mapping, and routing, while explicitly modeling gate errors, decoherence, and crosstalk. Second, we propose CARL, a crosstalk-aware compilation framework based on RL that integrates dependency-aware preprocessing, topology-aware subcircuit transformation, and hierarchical decision making to efficiently solve the formulated problem. Third, we conduct extensive experiments on real-world benchmark circuits and realistic hardware models, demonstrating that CARL outperforms state-of-the-art compilers in execution fidelity and robustness to crosstalk.

The remainder of this paper is organized as follows. Section II reviews related work. Section III provides the motivation and background. Section IV formulates the compilation problem, and Section V details the proposed RL-based framework. Section VI presents the performance evaluation. Finally, Section VII concludes the paper.

II. Related Work

Research on QCC has received significant attention over the past decade. This section reviews the most relevant prior work and highlights the gaps that motivate our approach.

Existing work primarily focuses on qubit mapping and routing to address qubit connectivity constraints. Due to the NP-hard nature of the problem, most existing approaches rely on heuristic algorithms. For example, in [10], Li *et al.* proposed SABRE, a heuristic that uses bidirectional search and a decay-based scoring mechanism to optimize initial mapping and reduce `SWAP` insertions during routing; this algorithm has been widely adopted as a core component of Qiskit [13]. Similarly, in [14], Niu *et al.* introduced HA, a noise-aware heuristic that integrates calibration data into the cost function to select higher-fidelity paths. To prioritize routing efficiency, the tket compiler [15] employs a graph-based greedy routing strategy. This approach enables rapid mapping while simultaneously simplifying the circuit through local gate reordering and redundancy elimination. To overcome the local optimality of greedy heuristics, RL-based approaches have also been explored for qubit mapping and routing. For instance, Pozzi *et al.* presented an approach based on deep Q-learning in [16], and Russo *et al.* incorporated graph neural networks into their RL-based framework to enhance the agent's perception of the hardware topology in [12]. However, the above approaches focus exclusively on qubit mapping and routing in QCC to minimize the number of `SWAP` gates, and lack the capability to transform circuits using functionally equivalent alternatives to further reduce gate count and circuit depth, thereby missing significant optimization opportunities offered by circuit transformation.

Several studies have investigated circuit transformation as a means of optimizing circuit execution performance by replacing circuits with hardware-efficient equivalents. Some approaches achieve this by replacing subcircuits with functionally equivalent implementations retrieved from precomputed datasets. For example, Quartz [17] utilizes a precomputed database of graph rewrites to adapt circuits to different hardware backends, while Quarl [18] introduces a circuit transformation framework that leverages such datasets to optimize circuits via RL. Additionally, Qmactr [19], proposed by Zhang *et al.*, also optimizes quantum circuits by retrieving functionally equivalent subcircuits from a dataset. However, Qmactr mainly targets entanglement consumption reduction in distributed quantum computing, and does not explicitly model crosstalk or directly optimize execution fidelity. Other approaches directly synthesize optimal equivalent circuits from unitary matrix representations. For example, Tan *et al.* proposed OLSQ in [20], which leverages Satisfiability Modulo Theories to find globally optimal circuit structures, but its scalability is limited. To handle scalability issues, Younis *et al.* introduced BQSKit [21], which partitions the input circuit into small blocks and performs independent resynthesis on each block. However, these methods do not consider qubit routing for addressing hardware constraints. The most relevant work to ours is the PAM heuristic proposed in [11], which integrates block-based circuit transformation into the qubit mapping and routing loop. However, PAM minimizes the proxy metric of two-input gate count and fails to directly optimize overall circuit execution fidelity.

However, none of the above works consider crosstalk between couplers or directly optimize execution fidelity. To address these limitations, this paper focuses on the joint optimization of qubit mapping, routing, and circuit transformation, while explicitly modeling crosstalk and directly maximizing fidelity.

III. Preliminaries and Motivating Examples

This section presents the foundational concepts of quantum computing and QCC. We begin with an overview of quantum circuits and the constraints imposed by NISQ devices, and then introduce qubit mapping, routing, and circuit transformation, highlighting the limitations of traditional SWAP-based approaches through examples.

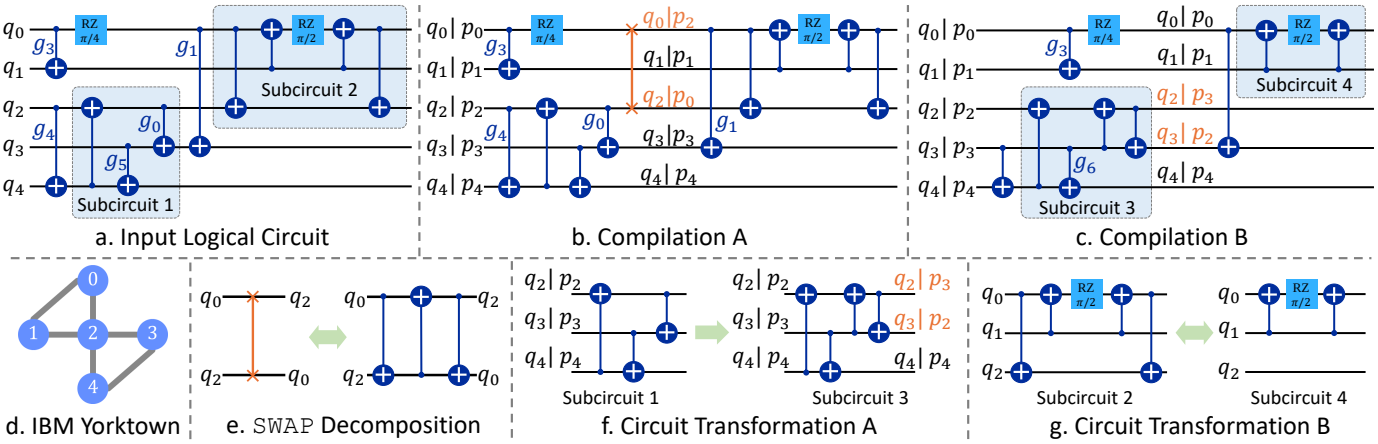

Fig. 1: A Motivational Example. (a) Input logical circuit. (b) Compilation A: traditional approach, resulting in 13 `CX` gates, 13 layers, and severe crosstalk. (c) Compilation B: crosstalk-aware joint optimization, resulting in 9 `CX` gates, 9 layers, and reduced crosstalk. (d) Qubit coupling topology of IBM Yorktown processor. (e) Decomposition of the non-native `SWAP` into three `CX` gates. (f) Circuit transformation A changing qubit locations. (g) Circuit transformation B reducing the gate count.

A. *Quantum Circuits*

A quantum circuit is an ordered sequence of single- and multi-qubit gate operations acting on logical qubits, which transforms an initial quantum state into a desired output state. Fig. 1(a) shows an example of a quantum circuit, where the vertical lines represent logical qubits, and gates are applied sequentially to their corresponding qubits, analogous to applying classical logic gates to classical bits. The circuit is represented in a layered form, where gates that appear in the same vertical column act in parallel and are said to belong to the same layer. The circuit can equivalently be represented as a directed acyclic graph (DAG), where nodes represent quantum gates and edges encode their data dependencies and execution order. From a mathematical perspective, the quantum state is represented as a complex vector, each gate is represented as a unitary matrix, and applying a gate corresponds to multiplying the state vector by the corresponding matrix [22]. The circuits considered here are logical circuits, which are hardware-agnostic and assume ideal execution. Quantum compilers take such a logical circuit as their input. In the input logical circuit, the gates can be arbitrary unitary operations and are not restricted by hardware constraints.

B. *Noisy Intermediate-Scale Quantum Processors*

In the near term, NISQ processors are the available quantum hardware on which quantum programs are executed. However, NISQ devices impose several fundamental constraints on the execution of quantum circuits.

One key constraint is that NISQ processors support only a limited number of physical qubits for realizing the logical qubits in quantum circuits. In addition, due to physical constraints of the hardware, qubits on an NISQ processor exhibit limited connectivity, and two-qubit gates can be applied only between directly connected qubits. For example, Fig. 1(d) shows the qubit coupling topology of the 5-qubit IBM Yorktown processor, where nodes represent qubits and edges denote couplers. Two-qubit gates can be applied between qubits directly connected by couplers, such as p_0 and p_1, and cannot be applied between qubits without a direct connection, such as p_0 and p_3. Therefore, logical qubits must be mapped onto physical qubits, and their locations must be routed accordingly during the circuit execution to satisfy the connectivity constraints.

In addition, NISQ processors also restrict the types of gate operations that can be directly executed. Each processor supports only a limited, predefined set of gate operations, known as the *native gate set*. Different quantum processors, built on different hardware platforms, may have different native gate sets. For example, IBM's Nighthawk processors using superconducting qubits support native gate set {`CZ, ID, RX, RZ, SX, X`} [23], where `CZ` is a two-qubit gate and the others are single-qubit gates. In contrast, IonQ's Aria Systems based on trapped-ion technology support a different native gate set of {`GPI, GPI2, MS`}, in which `MS` is the only two-qubit gate [24]. These native gate sets are universal, meaning that the gates in each universal gate set are sufficient to realize any logical circuit. As a result, logical gates in the input circuit must be transformed and decomposed into sequences of native gates supported by the target NISQ processor.

Beyond these architectural constraints, NISQ processors are also fundamentally limited by intrinsic noise in both qubits and gate operations. In practice, low circuit fidelity caused by qubit decoherence and gate errors is one of the main obstacles preventing NISQ devices from solving large-scale, practically relevant quantum problems [6]. First, qubits gradually decohere over time due to unavoidable interactions with their surrounding environment [22]. Each gate operation takes a given amount of time. Therefore, since the number of layers (circuit depth) is roughly proportional to the execution time, reducing the number of layers shortens the circuit duration and improves its operational fidelity. Second, gate operations can introduce non-negligible errors [25]. Consequently, decreasing the total

number of gate operations can further enhance the circuit execution fidelity. Third, gates executed on spatially adjacent couplers may interfere with each other and experience reduced fidelity due to crosstalk [8]. In particular, the error rates of two-qubit gates can increase by nearly an order of magnitude when they are executed simultaneously with neighboring two-qubit gates, compared to isolated execution [9]. Accordingly, minimizing circuit depth, gate count, and crosstalk is crucial for achieving high-fidelity circuit execution.

Therefore, efficient quantum compilers that can transform logical circuits into hardware-executable implementations while improving execution fidelity on NISQ devices are highly desirable.

C. Quantum Circuit Compilation

The QCC process comprises qubit mapping, routing, and circuit transformation, whose joint optimization is critical for achieving high-fidelity execution.

First, qubit mapping and routing are the processes of assigning logical qubits to physical qubits and adjusting their locations through `SWAP` operations to satisfy hardware connectivity constraints. For example, in Fig. 1(b), logical qubits $q_0, q_1, \cdots, q_4$ of the input circuit from Fig. 1(a) are initially mapped to physical qubits $p_0, p_1, \cdots, p_4$ on the 5-qubit IBM Yorktown processor, where $q_i|p_j$ represents that logical qubit q_i is hosted by the j-th physical qubit. In Fig. 1(b), the gates before g_0 can be executed directly without violating the qubit connectivity constraints. However, after executing gate g_0, `CX` gate g_1 applied to q_1 and q_3 cannot be executed directly because there is no direct connection between p_0 and p_3. To resolve this, it is necessary to change the locations of logical qubits through `SWAP` operations. For example, we can swap the locations of q_2 and q_3 using a `SWAP` operation, shown as the orange gate in Fig. 1(b). The `SWAP` operation is not natively supported on most quantum processors and must be decomposed into native gates, such as `CX` gates, as shown in Fig. 1(e). Each `SWAP` operation increases both the gate count and the circuit execution time. Therefore, qubits must be routed carefully to minimize the gate count and circuit execution time and ultimately improve the circuit execution fidelity.

Second, circuit transformation is also crucial for both satisfying hardware constraints and improving circuit execution fidelity. In addition to adapting logical circuits to the native gate set of the target hardware, circuit transformation can also be used to replace sub-circuits with functionally equivalent but more efficient alternatives [18]. For example, as shown in Fig. 1(g), subcircuit 2 on the left is functionally equivalent to subcircuit 4 on the right, but uses fewer gates. Therefore, applying circuit transformations provides opportunities to reduce circuit depth and gate count. Moreover, circuit transformation is closely intertwined with qubit routing, as certain transformations can implicitly change the locations of logical qubits. For example, the circuit transformation shown in Fig. 1(f) also swaps the locations of qubits q_2 and q_3. This side effect can be leveraged to reduce the number of explicitly inserted and costly `SWAP` operations. For instance, in Fig. 1(c), applying the transformation from Fig. 1(f) also swaps the locations of q_2 and q_3, thereby enabling the subsequent gate g_1 to satisfy the connectivity constraints without additional `SWAP` insertions. Compared with the compilation shown in Fig. 1(b), the compilation in Fig. 1(c), which explicitly leverages circuit transformation for joint optimization, reduces both gate count and execution time. As a result, jointly optimizing qubit mapping, routing, and circuit transformation enables more improvement of circuit execution fidelity.

Beyond the necessity of joint optimization, explicitly considering crosstalk can further enhance circuit execution fidelity. In general, crosstalk strength decays with the spatial separation between couplers [26]. On the IBM Yorktown processor, for example, the simultaneous execution of $\texttt{CX}(p_0, p_1)$ and $\texttt{CX}(p_2, p_3)$ is expected to introduce more error than that of $\texttt{CX}(p_0, p_1)$ and $\texttt{CX}(p_3, p_4)$, because the former pair operates on couplers that are closer in the device's coupling topology. Therefore, the compilation can be further improved by avoiding the simultaneous execution of gates on nearby couplers and instead scheduling parallel operations on spatially well-separated couplers to mitigate crosstalk-induced errors. For example, Compilation A shown in Fig. 1(b) includes the simultaneous execution of g_3 and g_4, which can introduce large crosstalk errors, whereas Compilation B in Fig. 1(c) mitigates crosstalk by delaying g_3, thereby scheduling it to execute simultaneously with g_6 and resulting in lower crosstalk. The delay can be achieved by inserting an identity operation `I`, which functions as a no-operation and effectively introduces idle time on the qubit, thereby shifting the execution time of subsequent gates and avoiding harmful concurrent operations. In this paper, we account for crosstalk in the quantum circuit compilation process.

IV. System Model and Problem Formulation

This section presents the system model for crosstalk-aware circuit compilation on NISQ devices and formulates the compilation problem as a joint optimization of mapping, routing, and transformation to maximize execution fidelity.

System Model. We model a NISQ device as an undirected coupling graph $\mathcal{G} = (\mathcal{P}, \mathcal{E})$, where $\mathcal{P} = \{p_0, p_1, \ldots, p_{m-1}\}$ represents the set of m physical qubits and $\mathcal{E} \subset \mathcal{P} \times \mathcal{P}$ denotes the set of edges corresponding to available physical couplers. A two-qubit gate can only be executed on physical qubits p_u and p_v if $\{p_u, p_v\} \in \mathcal{E}$. To capture crosstalk between couplers, i.e., edges, we denote the set of couplers that can induce crosstalk with the coupler $\{u, v\}$ as $\mathcal{N}(\{u, v\})$. In addition, we use $\mathcal{S}$ to denote the collection of native gates supported by the NISQ processor. Let $C = (g_1, g_2, \ldots, g_k)$ denote an input circuit consisting of a sequence of gates acting on the logical qubit set $Q_C = \{q_0, q_1, \ldots, q_{n-1}\}$, where $n \leq m$.

A compilation of the input circuit C is represented by τ that converts C into a hardware-compliant circuit $C' = \tau(C)$. This process involves: (1) Initial Mapping M_0, which assigns logical qubits in Q to physical qubits in P; (2) Routing, which inserts `SWAP` operations to satisfy connectivity constraints; and (3) Transformation, which replaces sub-circuits with functionally

equivalent realizations to optimize performance. To ensure correctness, we define $\texttt{eqv}(C', C) = 1$ if the unitary matrix of the transformed circuit $U(C')$ is identical to the original $U(C)$. For each gate $g \in C'$, it must be in the native gate set $\mathcal{S}$ of the focused device. In addition, for any gate $g \in C'$, we define $\texttt{cpl}(g)$ as the coupler acted on by g, i.e.,

$$\texttt{cpl}(g) = \begin{cases} \varnothing, & \text{if } g \text{ is a single-qubit gate acting on } u, \\ \{u, v\}, & \text{if } g \text{ is a two-qubit gate acting on} (u, v). \end{cases}$$

To capture gate concurrency, we define a layer indicator $\ell(g)$ that assigns each gate $g \in \tau(C)$ to its execution layer. In addition, we define the set of gates that are executed in the same layer as gate $g \in \tau(C)$ as

$$\mathcal{L}_\tau(g) \triangleq \{\, g' \in \tau(C) \mid \ell_\tau(g') = \ell_\tau(g) \,\}.$$

Objective Function. Maximizing circuit fidelity is essential for reliable quantum computation. It is primarily limited by gate errors and qubit decoherence [6]. First, quantum gates are inherently imperfect. Each gate g is affected by noise from control errors and environmental interactions. For a given compiled circuit $C' = \tau(C)$, effective fidelity of a gate $g \in \tau(C)$ is defined as

$$f_\tau(g) = \begin{cases} f_g^c, & \text{if } \exists\, g' \in \mathcal{L}_\tau(g) \text{ s.t. } \texttt{cpl}(g') \in \mathcal{N}(\texttt{cpl}(g)), \\ f_g^i, & \text{otherwise,} \end{cases}$$

where f_g^c is the fidelity of gate g when it is affected by crosstalk due to the simultaneous execution of another gate on a nearby coupler, and f_g^i represents the isolated fidelity of gate g when it is executed without any crosstalk interference. These gate errors accumulate as the circuit depth increases. Second, qubits degrade over time due to decoherence, including relaxation and dephasing caused by coupling to the environment. Even without operations, this process reduces the stored quantum information. Let $T_\tau(q)$ denote the total time during which qubit q participates in the execution of the compiled circuit (including both gate operations and idle periods), and let $\bar{T}$ be the average coherence time of the device. The decoherence-induced fidelity factor for qubit $q \in Q$ is given by

$$f_\tau(q) = \exp\left(-\frac{T_\tau(q)}{\bar{T}}\right),$$

where $\bar{T}$ denotes the effective coherence time of qubits on the target NISQ processor [22].

Following the literature [27], this paper formulates the circuit fidelity under compilation τ as the product of gate fidelities and decoherence factors as follows

$$\mathcal{F}(\tau(C)) = \prod_{g \in \tau(C)} f_\tau(g) \cdot \prod_{q \in Q_{\tau(C)}} f_\tau(q).$$

Problem Formulation. Given the definitions above, the crosstalk-aware quantum circuit compilation problem is defined as finding an optimal compilation policy τ that maximizes the execution fidelity $\mathcal{F}(\tau(C))$. The problem denoted by P can be formally expressed as:

$$\text{P} \quad \begin{aligned} \max_{\tau} \quad & \mathcal{F}(\tau(C)) \\ \text{subject to} \quad & \texttt{eqv}(\tau(C), C) = 1, \\ & g \in \mathcal{S}, && \forall g \in \tau(C), \\ & \texttt{cpl}(g) \in \mathcal{E}, && \forall g \in \tau(C). \end{aligned}$$

The first constraint enforces logical equivalence between the compiled and original circuits. The second constraint restricts the compiled circuit to use only gates from the native gate set $\mathcal{S}$. The last constraint imposes hardware compliance, ensuring that every two-qubit gate in the output circuit operates on physically connected qubits.

Problem Complexity. Solving this optimization problem is extremely challenging for two main reasons. First, the search space is massive. We are not just deciding where to place qubits (which is already an NP-hard problem [28]), but also choosing among many ways to transform the circuit logic. This combination creates an exponential number of possibilities that are impossible to enumerate. Second, the fidelity objective involves complex trade-offs. For example, routing a gate through a high-fidelity path might take a longer route, increasing the circuit depth and causing more decoherence. Traditional rule-based heuristics often look only one step ahead and fail to balance these conflicting factors. To navigate this complex landscape effectively, we propose to leverage RL to learn a global optimization policy.

V. RL-Based Quantum Compiler: CARL

In this section, we present a RL-based approach to solve the joint optimization problem.

A. Framework Overview

To address the critical challenge of fidelity degradation in quantum computing, we propose CARL, a RL framework that redefines the quantum circuit compilation pipeline. Fig. 2 illustrates the overall workflow of CARL, which introduces composite state construction, policy network, bimodal action selector, and environment updates.

In the initialization phase, we determine an initial mapping of logical qubits to physical qubits. We first partition the input circuit into subcircuit blocks and generate a library of hardware-compliant equivalent variants for each block. To overcome the initialization sensitivity of RL, we employ Simulated Annealing (SA) to construct a high-quality initial mapping. This static mapping explicitly accounts for temporal circuit dependencies by prioritizing the connectivity of operations scheduled earlier in the execution timeline over those scheduled later.

In the learning phase, we treat the dynamic compilation process as a Markov Decision Process (MDP) and employ a Deep Reinforcement Learning (DRL) agent to learn the compilation policy. The learning phase handles both qubit swapping and circuit transformation. The agent navigates this expanded search space via a joint decision-making strategy. For qubit

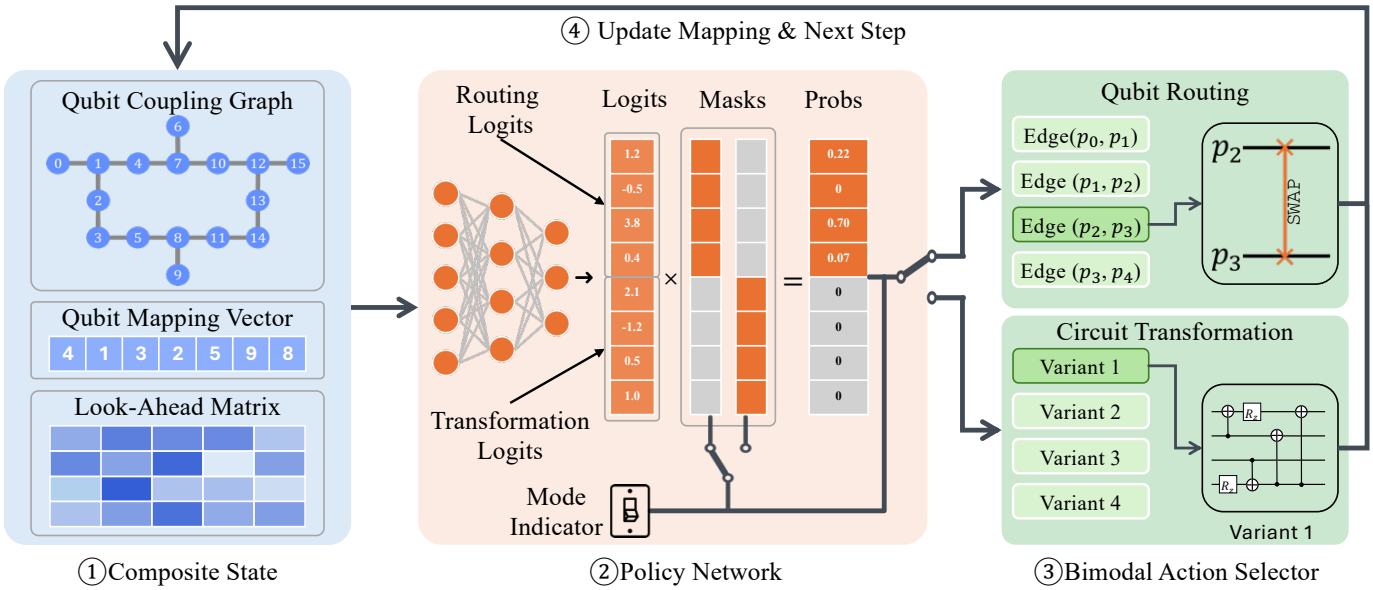

Fig. 2: Overview of the proposed CARL framework. The workflow operates in four stages: ① **Composite State Construction**, which encodes hardware topology, current mapping, and look-ahead dependencies; ② **Policy Network**, utilizing an action masking mechanism to filter invalid moves based on the current mode; ③ **Bimodal Action Selector**, where the agent executes either physical *qubit routing* (SWAP insertion) or logical circuit transformation (equivalent variant selection); and ④ **Environment Update**, which updates the logical to physical mapping and advances the compilation step.

routing, when connectivity constraints are not met, the agent performs `SWAP` operations. This dynamically adjusts the qubit locations to bring interacting qubits to adjacent physical qubits. For circuit transformation, when connectivity allows, the agent selects the optimal precomputed equivalent variant for the current logical block. This step serves as a local optimization mechanism within the compilation flow; crucially, because different equivalent variants yield distinct output permutations, selecting a specific variant implicitly routes the logical qubits for subsequent operations while reducing gate counts.

The core innovation lies in the crosstalk-aware joint optimization. Rather than greedily choosing the shortest path or the smallest circuit block, the agent utilizes a decay-based look-ahead mechanism. By evaluating a weighted future-interaction matrix and potential crosstalk, the policy balances the immediate fidelity gain of circuit transformation against the long-term overhead of qubit routing, ensuring that current decisions not only facilitate downstream execution but also actively steer circuit operations away from crosstalk.

B. Initialization

We initialize the compilation by precomputing equivalent variants of subcircuit blocks and finding a high-quality starting mapping.

Dependency-Aware Partitioning. We first represent the input circuit as a DAG to capture the dependency relationships between quantum operations. Based on the topological ordering of the DAG, the circuit is sliced into a sequence of disjoint logical blocks $\mathcal{B} = \{B_1, B_2, \ldots, B_N\}$. The partitioning algorithm employs a greedy, layer-wise aggregation strategy. Iterating through the DAG layers, we accumulate quantum gates into the current block B_i. The expansion of a block terminates when the number of unique logical qubits involved reaches a predefined threshold k (e.g., $k = 3$ in our experiments), or when a dependency violation is detected. This ensures that each block B_i represents a locally isolatable unitary transformation U_i acting on a small subset of qubits, making computationally expensive subcircuit transformation feasible.

Topology-Aware Subcircuit Transformation Preprocessing. A core innovation of CARL is that it does not treat a circuit block B_i as a fixed sequence of gates. Instead, for each block, we precompute a library of equivalent variants, denoted as $\mathcal{V}_i$. Since the physical connectivity of qubits on a NISQ device varies (e.g., linear, triangular, or star-shaped subgraphs), a single equivalent subcircuit variant may be optimal for one region of the NISQ processor but suboptimal for another. To address this, we leverage the BQSKit compiler [21] to perform subcircuit resynthesis. For each block B_i, we generate multiple hardware-compliant implementations under different connectivity constraints. For each block, we precompute a small library of hardware-compliant variants under different connectivity constraints, and store each variant together with its output permutation. By precomputing these variants, we decouple the expensive circuit transformation overhead from the real-time decision loop, allowing the RL agent to select a candidate variant instantly during the compilation process.

Dependency-Aware Initial Mapping. To facilitate efficient RL convergence and avoid the cold-start problem, we generate an initial mapping M_0 via SA rather than a random mapping.

Algorithm 1: RL-based CARL

Input: Circuit C and Policy π
Output: Optimized Circuit C^*

// Phase 1: Preprocessing
Partition C into dependency-aware blocks $\mathcal{B} = \{B_1, \ldots, B_N\}$;
Precompute subcircuit variants $\mathcal{V}_i$ for each $B_i \in \mathcal{B}$;
// Phase 2: RL-Based Compilation
for *episode* $e = 1$ **to** E_{max} **do**
  Initialize mapping M via SA($\mathcal{B}$);
  Reset: $C^* \leftarrow \emptyset$, mode $\omega \leftarrow 0$, block idx $i \leftarrow 1$;
  while $i \leq N$ **do**
    Get state S_t and validity mask m_t;
    Sample action $a_t \sim \pi(a_t|S_t, m_t)$;
    if $\omega = 0$ **then** // Qubit Routing
      if a_t *is SWAP* **then**
        Update M via SWAP(a_t);
        $r_t \leftarrow$ Eq. (1)(Mapping Reward);
      else // Commit to Opt.
        $\omega \leftarrow 1$;
      end
    else // Subcircuit Transformation
      Choose variant $v \in \mathcal{V}_i$ via a_t;
      Append v to C^* and Update M based on v;
      $r_t \leftarrow$ Eq. (2) (Optimization Reward);
      $i \leftarrow i + 1$;
      $\omega \leftarrow 0$;
    end
    Store transition and update π via PPO;
  end
end
return C^*

We first construct a weighted interaction graph from the partitioned blocks, where logical qubit pairs that interact earlier or more frequently receive larger weights. The SA algorithm then minimizes the weighted physical distance on the device coupling graph, yielding an initial mapping M_0 that favors early dependencies.

C. Compilation Via Reinforcement Learning

This section frames the problem as an MDP and presents the proposed RL framework, following the workflow shown in Fig. 2.

State Representation. To enable the agent to perceive both the static hardware topology and the dynamic circuit dependencies, we design a comprehensive state representation $S_t = \{A_G, W_t, M_t, \omega_t\}$. The static adjacency matrix $A_G \in \{0,1\}^{m\times m}$ represents the physical coupling graph $\mathcal{G}$, where $A_{ij} = 1$ indicates a physical connection between qubits i and j. To capture long-term dependencies, we construct a weighted matrix $W_t \in \mathbb{R}^{n\times n}$ representing the interaction demand between logical qubits. Unlike strictly local observations, W_t aggregates dependencies from the current block and future look-ahead blocks, scaled by a discount factor. This allows the agent to anticipate future connectivity requirements. In addition, we use a binary matrix $M_t \in \{0,1\}^{m\times n}$ to encode the current logical-to-physical assignment, where $M_{ij} = 1$ if physical qubit i holds logical qubit j. $\omega_t \in \{0,1\}$ is a discrete variable indicating the current decision mode (0 for mapping adjustment, 1 for circuit transformation), which conditions the agent's action space. This composite state $S_t = \{A_G, W_t, M_t, \omega_t\}$ serves as the input to the agent's policy network, enabling it to correlate physical constraints with logical requirements for optimal decision-making. For each candidate action, we also attach an action-feature vector that summarizes the estimated routing distance, variant cost, and crosstalk risk, so the policy can compare legal actions more directly.

Action Space. To accommodate the interleaved decision process, we define a unified discrete action space $\mathcal{A}$ that encompasses both qubit routing and subcircuit transformation choices. The validity of actions at time step t is strictly conditioned on the mode indicator ω_t via an action masking mechanism. The action space is structured as the union of two subsets: $\mathcal{A} = \mathcal{A}_\mathrm{m} \cup \mathcal{A}_\mathrm{g}$. In particular, $\mathcal{A}_\mathrm{m}$ is the collection of qubit routing actions, and $\mathcal{A}_\mathrm{g}$ represents the collection of circuit transformation actions.

When the agent is in the qubit routing mode (indicated by state $\omega_t = 0$), the valid actions correspond to the physical edges of the device coupling graph $\mathcal{G} = (\mathcal{P}, \mathcal{E})$. For each edge $\{p_u, p_v\} \in \mathcal{E}$, there exists a corresponding action $a_{uv} \in \mathcal{A}_\mathrm{m}$. Executing a_{uv} triggers a SWAP operation between physical qubits u and v, updating the mapping state M_t and incurring a gate cost penalty. We also include a special transition action a_c. This action becomes valid if and only if the current mapping M_t satisfies the connectivity constraints for the current logical block. Executing a_c terminates the qubit routing mode and transitions the state to $\omega_{t+1} = 1$ (circuit transformation mode). In contrast, when the agent is in the circuit transformation ($\omega_t = 1$), the action space corresponds to the index of the precomputed equivalent variants for the current subcircuit block. Let $\mathcal{V}_i$ be the library of generated variants for the current block B_i. The agent selects an action $a_k \in \{0, \ldots, |\mathcal{V}_i| - 1\}$. Executing a_k appends the k-th synthesized circuit variant to the compilation output. As defined in our preprocessing step, this action implicitly updates the logical-to-physical mapping, and advances the compilation process to the next logical block B_{i+1}, resetting the mode to $\omega_{t+1} = 0$.

Since $\mathcal{A}_\mathrm{m}$ and $\mathcal{A}_\mathrm{g}$ are mutually exclusive based on the system state, we utilize an action masking scheme. At each step t, when $\omega_t = 0$, all actions in $\mathcal{A}_\mathrm{g}$ are masked as illegal; conversely, when $\omega_t = 1$, all actions in $\mathcal{A}_\mathrm{m}$ are masked. This mechanism enforces strict adherence to the hierarchical compilation flow.

Reward Function. To guide the agent toward global optimality, we design a composite reward function that explicitly balances gate count reduction, circuit depth minimization, and crosstalk reduction. The objective is to penalize operations that extend the critical path or induce crosstalk, while incentivizing

actions that improve fidelity.

Routing Reward (R_t). In the qubit routing mode, the reward minimizes the performance overhead introduced by qubit routing. Since SWAP operations are typically decomposed into multiple two-qubit gates and must be executed sequentially, they are the primary contributor to depth expansion. Furthermore, inserting SWAP operations may induce crosstalk. We formulate the reward as:

$$R_{\mathrm{t}} = -C_{\mathrm{SWAP}} \cdot (1 + \eta \cdot N_{\mathrm{c}}) + \alpha \cdot (D_t - D_{t+1}), \tag{1}$$

where C_{SWAP} is a fixed penalty reflecting the impact of a SWAP operation on both error accumulation and circuit depth. The term N_{c} quantifies the number of detected crosstalk caused by the inserted SWAP operation, scaled by the coefficient η. The second term $(D_t - D_{t+1})$ represents the reduction in physical distances between logical qubits in the current block, scaled by α. This potential-based shaping accelerates convergence by guiding the agent to resolve connectivity violations via the shortest path.

Transformation Reward (R_o). In the circuit transformation mode, the reward captures the trade-off between execution cost, the potential benefit of qubit permutation (i.e., changes in qubit locations), and crosstalk errors. Since equivalent variants of a subcircuit block differ in gate composition and topological structure, they serve as the primary determinant of local circuit depth and coherence preservation. Furthermore, gate scheduling within a specific variant may lead to crosstalk errors between neighboring qubits. We formulate the reward as:

$$R_{\mathrm{o}} = \underbrace{(\bar{C} - C)}_{\text{Cost Reduction}} + \lambda \cdot \underbrace{(P(M) - P(M'))}_{\text{Potential Gain of Qubit Permutation}} - \gamma \cdot N_{\mathrm{c}} \tag{2}$$

The first term encourages the selection of cost-efficient equivalent variants of the current block. Here, C represents the execution cost of the selected variant, defined as a weighted combination of its CZ count and local depth. By maximizing the advantage over the average cost $\bar{C}$, the agent minimizes the duration qubits are exposed to decoherence. The second term evaluates how the output permutation of the selected variant impacts future operations. We quantify this impact using the potential function $P(M)$, defined as the weighted sum of physical distances between logical qubits that are scheduled to interact in downstream blocks:

$$P(M) = \sum_{q_u, q_v \in Q_c} W_t[q_u, q_v] \cdot \mathrm{dist}_{\mathcal{G}}(M[q_u], M[q_v]). \tag{3}$$

In Eq. (3), M and M' represent the mapping state before and after applying the permutation induced by the chosen equivalent variant, respectively. $W_t[q_u, q_v]$ denotes the entry in the look-ahead matrix representing the aggregated, decay-weighted interaction demand between logical qubits q_u and q_v in future blocks. A high value indicates these qubits need to interact soon. In addition, $M[q_u]$ and $M[q_v]$ are the physical qubits occupied by logical qubits q_u and q_v under mapping M. And $\mathrm{dist}_{\mathcal{G}}(M[q_u], M[q_v])$ is the shortest-path distance on the qubit coupling graph $\mathcal{G}$ between physical qubits $M[q_u]$ and $M[q_v]$.

Maximizing Eq. (2) ensures that logically interacting qubits in downstream layers are placed physically close. This spatial locality mitigates the SWAP insertion problem and, combined with the crosstalk penalty, ensures that the resulting circuit achieves high fidelity by avoiding crosstalk.

Policy Optimization. To learn the complex mapping between the multi-input state and the hierarchical action space, we employ the Maskable Proximal Policy Optimization algorithm [29].

Standard RL algorithms typically assume a fixed action space. However, quantum compilation imposes strict validity constraints, e.g., a SWAP gate can only be applied to physically connected qubits, and local gate optimization can only be triggered when the requisite connectivity is satisfied.

Action Masking Mechanism. We explicitly integrate an action masking filter into the policy network to ensure validity. Let $\mathcal{M}(s_t)$ be the set of legal actions at state s_t. We define a binary mask vector m_t where $m_{t,a} = 1$ if $a \in \mathcal{M}(s_t)$ and 0 otherwise. During the qubit routing mode ($\omega_t = 0$), the mask enables only actions related to physical SWAP operations and disables all circuit transformation actions. Conversely, during the circuit transformation mode ($\omega_t = 1$), only valid equivalent variants are enabled. The probability of selecting an invalid action is forced to zero by modifying the policy logits before the softmax operation:

$$\pi(a_t|s_t) = \frac{m_{t,a} \cdot \exp(l_a)}{\sum_{a' \in \mathcal{A}} m_{t,a'} \cdot \exp(l_{a'})} \tag{4}$$

where l_a are the raw logits output by the actor network. This mechanism allows the agent to learn efficiently within the valid manifold of the optimization landscape without wasting training epochs on illegal transitions.

Network Architecture and Training. We utilize an Actor-Critic architecture where both networks share a feature extractor that processes the composite state inputs. The Actor head outputs the policy distribution $\pi_\theta(a|s)$, while the Critic head estimates the value function $V_\phi(s)$ to compute the generalized advantage estimation.

To enhance training stability and accelerate convergence, we employ a Reward Normalization technique using the VecNormalize wrapper [30]. This component continuously estimates the running mean and standard deviation of the received rewards and normalizes them to have zero mean and unit variance. This ensures that the reward scale remains consistent throughout training, preventing large fluctuations that can destabilize learning. Additionally, to encourage exploration and prevent premature convergence to suboptimal decisions, we implement a dynamic entropy regularization schedule. The entropy coefficient is initialized at a high value to promote diverse trajectory sampling and is linearly decayed to a lower bound as training progresses, allowing the agent to exploit the learned policy for fidelity maximization in later stages.

At each time step t, the agent receives the composite state S_t and generates action probabilities via the masked policy. Ac-

TABLE I: Characteristics of benchmark circuits.

Circuit	N_q	N_{gate}	N_{CZ}	Depth
qft_5	5	60	20	30
qaoa_5	5	89	24	41
grover_5	5	502	168	314
qnn_5	5	34	4	14
ae_5	5	88	20	48
hhl_12	12	650	231	253
qft_16	16	544	107	249
full_adder_16	16	390	113	253
full_adder_28	28	710	209	452

Note: N_q, N_{gate}, N_{CZ}, and Depth represent the number of logical qubits, the total number of quantum gates, the number of two-qubit gates, and depth of the logical circuit, respectively.

tion selection follows a stochastic sampling strategy throughout the learning process to ensure extensive exploration of the hierarchical search space. Meanwhile, a best-output tracking mechanism continuously records and retains the compiled circuit instance C^* that achieves the highest fidelity during training.

VI. Performance Evaluation

In this section, we provide a comprehensive evaluation of the performance of the proposed CARL. The source code for our evaluation is publicly available on GitHub at https://github.com/deltapolyu/Carl.

A. Evaluation Settings

All evaluations were conducted on a system running Ubuntu 22.04 LTS, equipped with a 96-core AMD Ryzen 7795WX processor, 256 GB of system memory, and an NVIDIA RTX 3090 GPU with 24 GB of memory. The RL framework was implemented using Python 3.12 and PyTorch 2.9.1, with GPU acceleration provided by CUDA 12.7.

To evaluate the execution fidelity, we utilize real-world data from the IBM Quantum computer `ibm_miami` [31]. The simulation parameters are set based on the device's average properties: relaxation time $T_1 = 348\mu s$, dephasing time $T_2 = 263\mu s$, two-qubit error rate $E_{2q} = 2.865 \times 10^{-3}$, single-qubit error rate $E_{1q} = 2.283 \times 10^{-4}$, single-qubit gate duration $T_{1q} = 32$ns, and two-qubit gate duration $T_{2q} = 138$ns. The supported native gates are $\mathcal{S} = \{$`CZ, RX, RZ, SX, X, ID`$\}$. Regarding crosstalk, following the experimental observations in [9], we set the error rate of simultaneous `CZ` gates on neighboring couplers to be ten times higher than that of isolated operations. To evaluate the scalability and adaptability of CARL across varying circuits, we selected a diverse set of quantum algorithms from the widely adopted MQT Bench [32] suites. Detailed characteristics of the selected benchmark circuits are summarized in Table I.

B. Baseline Methods

To comprehensively evaluate the efficacy of CARL, we compare its performance against four representative baselines. The first baseline, Qiskit [13], serves as the industrial standard, employing the stochastic SABRE heuristic with the highest optimization level. The second baseline, BQSKit [21], represents a state-of-the-art multi-step optimization pipeline that partitions and synthesizes logical blocks prior to mapping. The third baseline, PAM [11], implements a heuristic strategy that specifically integrates permutation-aware synthesis into the routing loop. Finally, t|ket⟩ [15] acts as a high-performance compiler baseline, utilizing advanced structural optimization passes and efficient routing algorithms. These baselines enable us to assess whether our proposed joint optimization outperforms both traditional sequential flows and specialized heuristics.

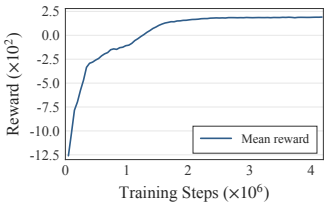

Fig. 3: Training convergence of the mean reward.

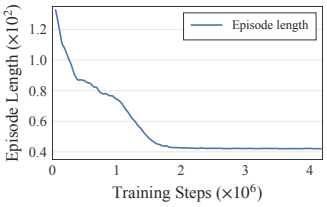

Fig. 4: Training convergence of the episode length.

C. Evaluation Results

Training Convergence We first evaluate the training stability and convergence behavior of CARL. Fig. 3 shows the training convergence of CARL. As shown in Fig. 3, the mean reward increases sharply from negative values and converges to a high positive plateau, indicating that the agent learns to balance fidelity maximization and constraint satisfaction. The transition from negative to positive values indicates that the agent evolves from random exploration, which is frequently penalized due to invalid actions and high crosstalk, to a high-quality policy that effectively mitigates interference and maximizes fidelity. Meanwhile, Fig. 4 shows that the episode length rapidly decreases and stabilizes. Since the episode length directly correlates with the number of operations in the compiled circuit, this reduction demonstrates that CARL efficiently explores the hierarchical action space to minimize compilation overhead in each episode. Together, these results confirm the training stability and convergence of CARL.

Performance Comparison. Next, we present the comprehensive evaluation results on the selected benchmark circuits in Table II. The evaluation considers four key metrics: execution fidelity, `CZ` gate count, crosstalk count (Xtalk), and circuit critical depth. We report execution fidelity in its normalized form, denoted as NF, to enable fair and consistent comparison across circuits of different scales. Specifically, all fidelities are normalized with respect to the fidelity achieved by our approach. As demonstrated in Table II, CARL consistently achieves superior execution fidelity across all the benchmark circuits. On average, CARL improves execution fidelity by 1.29×, 3.94×, 4.08×, and 3.44× over BQSKit, PAM, Qiskit, and t|ket⟩, respectively. In particular, the fidelity gain increases with circuit scale. For example, on `full_adder_28`, CARL achieves fidelity improvements of 1.25×, 3.81×, 10.20×, and

TABLE II: Performance comparison of different quantum compilation approaches across benchmark circuits.

Benchmark	**BQSKIT** [21]				**PAM** [11]				**Qiskit** [13]				**tket** [15]				**CARL (Ours)**			
	CZ	Depth	Xtalk	NF	CZ	Depth	Xtalk	NF	CZ	Depth	Xtalk	NF	CZ	Depth	Xtalk	NF	CZ	Depth	Xtalk	NF
qft_5	31	129	**0**	0.8946	**24**	150	2	0.8482	30	55	6	0.8368	38	78	10	0.7099	25	**53**	**0**	**1**
qaoa_5	**23**	101	**0**	0.9964	28	155	4	0.8285	30	**55**	12	0.7473	33	70	10	0.7644	30	70	**0**	**1**
grover_5	**216**	812	6	0.9679	222	1223	32	0.3113	331	746	44	0.2454	284	**656**	28	0.4944	272	734	**0**	**1**
qnn_5	**4**	26	**0**	0.9844	6	41	**0**	0.9784	**4**	**13**	**0**	0.9879	**4**	**13**	**0**	0.9987	**4**	**13**	**0**	**1**
ae_5	**25**	108	**0**	0.9701	26	155	2	0.8725	32	71	**0**	0.9803	32	83	4	0.8757	28	**70**	**0**	**1**
hhl_12	423	1208	**52**	0.3686	**319**	1403	143	0.0566	407	**613**	124	0.2870	450	739	119	0.1814	397	666	69	**1**
qft_16	416	1128	**53**	0.5136	**319**	1088	142	0.1680	379	**375**	165	0.4634	456	480	174	0.1424	452	569	85	**1**
full_adder_16	109	436	**2**	0.7141	**101**	541	24	0.3353	163	351	25	0.3243	161	331	23	0.3622	109	**251**	7	**1**
full_adder_28	211	834	**2**	0.8008	**185**	1049	26	0.2627	344	748	37	0.0980	361	745	34	0.0841	252	**563**	19	**1**

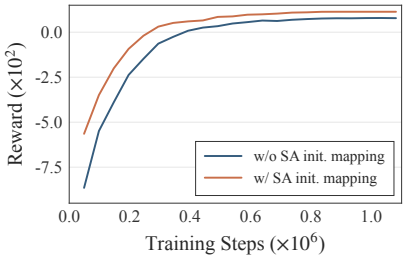

Fig. 5: Convergence w/ and w/o SA-based initial mapping.

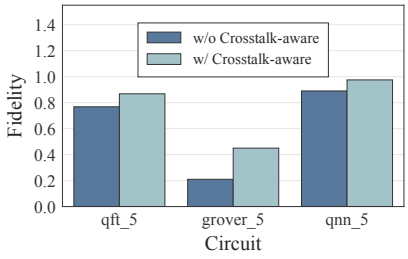

Fig. 6: Fidelity w/ and w/o Xtalk-aware.

11.89× compared to BQSKit, PAM, Qiskit, and t|ket⟩, respectively. This large improvement arises because execution fidelity decays exponentially with increasing circuit depth, gate count, and crosstalk, so even modest reductions in these factors lead to substantial fidelity gains.

We observe that a significant part of the fidelity improvement can be attributed to our crosstalk-aware optimization. By explicitly modeling interference between neighboring couplers, CARL avoids high-error parallel gate executions that are typically overlooked by existing compilers, leading to more reliable circuit execution. As shown in Table II, for most benchmark circuits, CARL achieves the lowest number of crosstalk operations among all compared methods, which demonstrates that our approach can explicitly avoid noisy crosstalk operations.

In addition, CARL benefits from its joint optimization of qubit mapping, routing, and circuit transformation while explicitly optimizing fidelity. Unlike existing approaches that treat these components independently or optimize only for a single objective, CARL balances gate errors, crosstalk, and decoherence in a unified framework, enabling it to discover compilation strategies that better preserve quantum state fidelity. For example, although BQSKit primarily optimizes the number of two-qubit gate operations and can achieve relatively low `CZ` counts, it produces circuits with significantly larger depth, which can severely degrade execution fidelity due to increased decoherence. Baselines may occasionally achieve slightly lower depths, but they fail to account for the noise induced by dense gate scheduling. In contrast, our approach explicitly accounts for this trade-off.

Overall, the evaluation results show that CARL yields significantly better fidelity than baselines that optimize proxy metrics.

Ablation Studies. Next, we conduct ablation experiments to evaluate the contribution of the key components in CARL. Specifically, we focus on the impact of the dependency-aware initial mapping and the crosstalk-aware reward function.

To investigate the necessity of the SA-based dependency-aware initial mapping, we compare the training performance of the full CARL framework against a variant that utilizes a random initial mapping. As illustrated in Fig. 5, the inclusion of the initial mapping provides several clear benefits. First, the agent starting with dependency-aware initial mapping begins with a significantly higher reward, indicating that our SA-based mapping effectively reduces initial connectivity violations and `SWAP` insertion overhead. Second, the curve with initial mapping exhibits a steeper learning slope, reaching a positive reward plateau earlier than the curve without initial mapping. Finally, CARL with the SA-based initial mapping converges to a higher total reward. These results demonstrate the importance of dependency-aware mapping in mitigating the cold-start problem.

To evaluate the impact of crosstalk awareness, we compare CARL with a variant that ignores crosstalk by setting the corresponding reward penalty to zero. This baseline optimizes only gate count and circuit depth, without accounting for interference between concurrent operations. As shown in Fig. 6, the crosstalk-aware strategy consistently outperforms this baseline across all benchmarks. Without crosstalk modeling, aggressive parallelization minimizes depth but simultaneously activates neighboring couplers, leading to severe error accumulation. In contrast, CARL identifies such high-risk patterns and trades

a small increase in depth for significantly higher fidelity. These results show that optimizing gate count or depth alone is insufficient on real hardware. In conclusion, the results demonstrate that crosstalk-aware optimization is essential to the performance of CARL, effectively mitigating error amplification in noisy quantum devices.

VII. Conclusion

This paper proposes CARL, a crosstalk-aware quantum compilation framework via RL that jointly optimizes qubit mapping, routing, and circuit transformation to maximize execution fidelity on NISQ devices. By explicitly modeling gate errors, decoherence, and crosstalk, CARL moves beyond proxy metrics such as gate count and depth, enabling fidelity-oriented compilation. CARL integrates hierarchical decision-making, dependency-aware partitioning, SA-based initial mapping, and a decay-based look-ahead mechanism. These components allow the agent to balance short-term routing costs with long-term fidelity. Evaluation results on diverse benchmarks show that CARL consistently outperforms state-of-the-art compilers, achieving significantly higher execution fidelity and substantially reducing crosstalk, especially for large-scale circuits.

References

[1] Y. Liu, Y. Mao, X. Xu, X. Shang, F. Ye, and Y. Yang, "A nonblocking multistage switching network for distributed quantum computing," *IEEE Transactions on Networking*, 2025.

[2] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature communications*, vol. 5, no. 1, p. 4213, 2014.

[3] Y. Cao, J. Romero, and A. Aspuru-Guzik, "Potential of quantum computing for drug discovery," *IBM Journal of Research and Development*, vol. 62, no. 6, pp. 6–1, 2018.

[4] R. Mandelbaum, "Scaling for quantum advantage and beyond," IBM Quantum Blog, Nov. 2025. [Online]. Available: https://www.ibm.com/quantum/blog/qdc-2025

[5] R. Acharya, L. Aghababaie-Beni, I. Aleiner, T. I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, N. Astrakhantsev, J. Atalaya, R. Babbush, D. Bacon, B. Ballard, ... *et al.*, "Quantum error correction below the surface code threshold," *Nature*, vol. 638, no. 8052, pp. 920–926, 2025.

[6] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[7] P. Krantz, M. Kjelhaug, C. K. Andersen, E. Kisley, S. Gustafsson, J. Bylander, W. D. Oliver, and C. M. Wilson, "A quantum engineer's guide to superconducting qubits," *Applied Physics Reviews*, vol. 6, no. 2, 2019.

[8] K. Rudinger, C. W. Hogle, R. K. Naik, A. Hashim, D. Lobser, D. I. Santiago, M. D. Grace, E. Nielsen, T. Proctor, S. Seritan, S. M. Clark, R. Blume-Kohout, I. Siddiqi, and K. C. Young, "Experimental characterization of crosstalk errors with simultaneous gate set tomography," *PRX Quantum*, vol. 2, p. 040338, Nov 2021.

[9] P. Zhao, K. Linghu, Z. Li, P. Xu, R. Wang, G. Xue, Y. Jin, and H. Yu, "Quantum crosstalk analysis for simultaneous gate operations on superconducting qubits," *PRX quantum*, vol. 3, no. 2, p. 020301, 2022.

[10] 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.

[11] 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)*, vol. 1. IEEE, 2023, pp. 745–756.

[12] E. Russo, M. Palesi, D. Patti, G. Ascia, and V. Catania, "Attention-based deep reinforcement learning for qubit allocation in modular quantum architectures," *arXiv preprint arXiv:2406.11452*, 2024.

[13] 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.

[14] S. Niu, A. Suau, G. Staffelbach, and A. Todri-Sanial, "A hardware-aware heuristic for the qubit mapping problem in the nisq era," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–14, 2020.

[15] 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, 2020.

[16] M. G. Pozzi, S. J. Herbert, A. Sengupta, and R. D. Mullins, "Using reinforcement learning to perform qubit routing in quantum compilers," *ACM Transactions on Quantum Computing*, vol. 3, no. 2, pp. 1–25, 2022.

[17] M. Xu, Z. Li, O. Padon, S. Lin, J. Pointing, A. Hirth, H. Ma, J. Palsberg, A. Aiken, U. A. Acar *et al.*, "Quartz: superoptimization of quantum circuits," in *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2022, pp. 625–640.

[18] Z. Li, J. Peng, Y. Mei, S. Lin, Y. Wu, O. Padon, and Z. Jia, "Quarl: A learning-based quantum circuit optimizer," *Proceedings of the ACM on Programming Languages*, vol. 8, no. OOPSLA1, pp. 555–582, 2024.

[19] X. Zhang, X. Xu, Y. Liu, Y. Mao, B. Xiao, and Y. Yang, "Joint optimization of circuit transformation and qubit mapping for distributed quantum computing," in *IEEE INFOCOM 2026-IEEE Conference on Computer Communications*. IEEE, 2026, pp. 1–10.

[20] B. Tan and J. Cong, "Optimal layout synthesis for quantum computing," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[21] E. Younis, C. C. Iancu, W. Lavrijsen, M. Davis, and E. Smith, "Berkeley quantum synthesis toolkit (bqskit) v1," Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA (United States), Tech. Rep., 2021.

[22] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[23] IBM Quantum, "Compute resources — nighthawk processors," https://quantum.cloud.ibm.com/computers?processorType=Nighthawk, 2025, accessed: 2025-10-27.

[24] IonQ, "Getting started with ionq's hardware-native gateset," https://docs.ionq.com/guides/getting-started-with-native-gates#introducing-the-native-gates, 2025, accessed: 2025-10-27.

[25] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, and D. J. Wineland, "Randomized benchmarking of quantum gates," *Phys. Rev. A*, vol. 77, p. 012307, Jan 2008.

[26] M. Alghadeer, S. P. Fors, S. Cao, S. D. Fasciati, H. Ishizaka, A. F. Kockum, P. Leek, and M. Bakr, "Crosstalk dispersion and spatial scaling in superconducting qubit arrays," 2025. [Online]. Available: https://arxiv.org/abs/2512.18148

[27] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, 2019, pp. 1015–1029.

[28] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 international symposium on code generation and optimization*, 2018, pp. 113–125.

[29] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *arXiv preprint arXiv:2006.14171*, 2020.

[30] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of machine learning research*, vol. 22, no. 268, pp. 1–8, 2021.

[31] IBM Quantum. (2026) Ibm quantum systems: ibm_miami. [Online]. Available: https://quantum.cloud.ibm.com/computers?system=ibm_miami

[32] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing," *Quantum*, vol. 7, p. 1062, 2023, MQT Bench is available at https://www.cda.cit.tum.de/mqtbench/.