Joint Optimization of Circuit Transformation and Qubit Mapping for Distributed Quantum Computing

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

***Abstract*—Distributed Quantum Computing (DQC) scales quantum computing capabilities by interconnecting multiple Quantum Processing Units (QPUs) for collaborative computation, but faces the critical challenge of high entanglement costs associated with remote gate operations. Circuit transformation and qubit mapping are critical components of quantum circuit compilation for minimizing entanglement cost, which are interdependent in DQC. Existing approaches that treat circuit transformation and qubit mapping as separate optimization problems fail to account for their fundamental interdependence, resulting in suboptimal entanglement utilization and degraded performance. This paper presents the first optimization framework for circuit transformation and qubit mapping tailored for DQC. To address this joint optimization problem, we propose *Qmactr*, a two-phase reinforcement learning approach. The first phase trains a qubit mapping agent, and the second phase trains a circuit transformation agent that utilizes the mapping agent as a reward oracle. Extensive evaluation results demonstrate that the proposed approach reduces entanglement consumption cost by up to 50.84% compared to state-of-the-art sequential methods while achieving superior circuit fidelity.**

I. Introduction

Quantum computing leverages quantum mechanical phenomena to solve problems intractable for classical computers across diverse fields, including cryptography [1], optimization [2], and machine learning [3]. However, current quantum processors face significant challenges such as limited qubit capacity, high noise levels, and short coherence times, severely constraining their practical utility [4]. To address these limitations, Distributed Quantum Computing (DQC) has emerged as a promising approach, where multiple Quantum Processing Units (QPUs) are interconnected to execute quantum circuits collaboratively [5]–[7]. For example, IBM Quantum's bicycle architecture leverages modular designs distributed across a set of replaceable modules connected quantumly [8], trapped-ion modules can be photonically interconnected to enable distributed quantum computing [9], and neutral-atom processors can be interconnected to facilitate distributed processing [10].

Despite its promise, DQC introduces unique challenges that do not exist in monolithic quantum computing. The most critical challenge is to implement remote gate operations between qubits distributed across different QPUs. These remote operations require the generation and consumption of entangled states shared between processors [11], which are orders of magnitude more costly than local gate operations. For instance, entanglement generation exhibits low success probability due to high photon loss rates, requiring tens of thousands of attempts to establish a single entangled pair, with costs up to 300 times higher than local operations [12]. Consequently, minimizing entanglement consumption emerges as the key optimization objective for efficient DQC implementation.

Circuit transformation represents a fundamental optimization strategy for enhancing circuit execution performance on Noisy Intermediate-Scale Quantum (NISQ) devices [13]–[15]. This technique systematically replaces sub-circuits with functionally equivalent alternatives, reducing the overall gate count to improve circuit execution time and fidelity. Circuit transformation becomes even more significant in DQC, as it helps reduce two-qubit gates that could otherwise require remote operations and consume costly entanglement. Therefore, optimizing quantum circuit transformation becomes critical for efficient DQC.

Equally important, qubit mapping determines the allocation of logical qubits from a quantum circuit to physical qubits distributed across multiple QPUs [16]–[18], directly influencing the number of remote gate operations required. Advanced techniques such as cat-entanglement protocols can enable multiple remote CX gates to be realized using a single shared entanglement, significantly improving resource efficiency [11]. However, such protocols place structural constraints on the circuit, as maintaining the cat-like state requires control qubits to remain unchanged across operations. This results in complex interdependencies between qubit mapping and circuit transformation.

Existing approaches treat circuit transformation and qubit mapping as independent, sequential optimization problems. This decoupled strategy fails to capture the fundamental interdependence between these two processes. As the motivational example in Section III-D demonstrates, a circuit transformation that increases gate count may actually reduce overall execution cost when it enables more efficient qubit mappings that minimize entanglement consumption through cat-entanglement protocols. Therefore, an approach for optimizing both circuit transformation and qubit mapping is desired to enhance distributed quantum circuit execution performance.

In this paper, we propose a novel reinforcement learning framework for the joint optimization of circuit transformation and qubit mapping in distributed quantum computing. The approach addresses the computational challenges posed by the large search space of equivalent circuits and the NP-

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

hard nature of the qubit mapping problem through a two-phase reinforcement learning strategy. In the first phase, we train a robust qubit mapping agent using Edge-Aware Graph Attention Networks that explicitly incorporates inter-QPU entanglement generation costs into the attention mechanism. In the second phase, we develop a circuit transformation agent with hierarchical action space decomposition that leverages the pre-trained qubit mapping agent as a reward oracle to guide the search efficiently in the large action space. The main contributions of this work are as follows. First, we formulate the joint optimization problem for entanglement cost minimization in DQC. Second, we propose *Qmactr*, a novel two-phase reinforcement learning framework that optimizes circuit transformation and qubit mapping. Third, we provide extensive evaluation results demonstrating significant improvements over state-of-the-art sequential optimization methods.

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

II. Related Work

This section reviews previous works related to quantum circuit transformation and qubit mapping.

Circuit Transformation. There have been many works using different methodologies for quantum circuit optimization. Rule-based strategies apply manually designed circuit transformations whenever possible, including Qiskit [19], t|ket⟩ [20], Quilc [21], and VOCQ [22]. Nam *et al.* developed automated heuristic methods that apply carefully chosen sequences of optimization subroutines to reduce gate counts [23]. In addition to rule-based approaches, several search-based methods have been proposed. Xu *et al.* introduced Quartz [13], a backtracking search-based circuit transformation algorithm, and later proposed QUESO [14], a beam search-based approach for quantum circuit synthesis and optimization. Most recently, Li *et al.* proposed a reinforcement learning-based quantum circuit optimizer named Quarl [15]. Quarl employs a hierarchical action space decomposition and graph neural networks to leverage the locality of circuit transformations, achieving superior performance compared to the above rule-based and search-based optimizers.

Qubit Mapping. The qubit mapping problem in DQC has emerged as a critical challenge distinct from single-processor mapping. Several approaches have formulated the qubit mapping in DQC as graph partitioning problems [24]–[26], but they did not consider the heterogeneous communication costs between different QPU pairs in realistic DQC clusters. Mao *et al.* formulated the general qubit allocation problem for DQC and proposed an approach using multistage hybrid simulated annealing [27]. Zhan *et al.* proposed QuPEP using genetic simulated annealing for offline data qubit placement and Lagrange relaxation for online entangled qubit provisioning [16] to minimize task completion time. Liu developed LarQucut, a cutting and mapping approach for large-sized quantum circuits that introduces the "hemicut" technique to avoid cutting circuits into completely independent sub-circuits, thereby reducing classical post-processing overhead [17]. Burt *et al.* proposed a multilevel framework for partitioning quantum circuits over distributed quantum processing units, adapting the Fiduccia-Mattheyses heuristic with a custom objective function [18].

However, these existing methods primarily focus on either circuit transformation or qubit mapping, without jointly considering circuit transformation and qubit mapping optimization in a unified framework.

III. Background and Motivation Example

In this section, we will present some background on quantum circuit transformation, distributed quantum computing, and qubit mapping. Besides, we will use an example to demonstrate the necessity of joint quantum circuit transformation and qubit mapping for distributed quantum computing.

A. Circuit Transformation

Circuit transformation replaces sub-circuits with functionally equivalent alternatives, which can improve the performance of quantum circuit execution, such as fidelity and circuit depth [28]. For example, as shown in Fig. 1, the original circuit on the left, containing four CX gates and one single-qubit gate, is functionally equivalent to the optimized circuit on the right, which has only two CX gates and one single-qubit gate. Replacing the original circuit with the optimized circuit

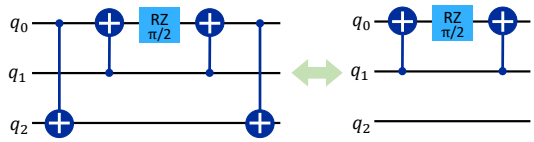

Fig. 1: A quantum circuit transformation example.

reduces both the gate count and cumulative error, thereby improving the overall fidelity. This paper will leverage circuit transformation to optimize the performance of distributed quantum computing.

To effectively apply circuit transformation techniques, it is essential to establish the foundational gate set that will serve as the target for optimization. A universal quantum gate set is a collection of quantum gates that can approximate any quantum computation to arbitrary precision, such as {CX, X, H, RZ} [23] and {arbitrary single-qubit gates, CX} [29]. During quantum circuit compilation, arbitrary input circuits are first converted into an intermediate representation using a chosen universal gate set [30] before proceeding to hardware-specific compilation and optimization. For example, some IBM processors use RZ, SX, and CX, and IonQ's quantum cloud platform transpiles input circuits to a representation using RX, RY, RZ, and CX gates and then converts CX gates into XX gates using a given decomposition [31]. The CX gate (or gates easily convertible to CX, such as XX) appears prevalent across most quantum computing architectures. Therefore, following the literature, this paper focuses on circuits composed exclusively of arbitrary single-qubit gates and CX gates.

B. Distributed Quantum Computing Framework

Since current quantum processors have a limited number of qubits, they cannot execute large-scale quantum circuits and solve real-world practical problems. Distributed quantum computing, which involves quantumly interconnecting quantum processors and performing computing collaboratively, is a promising paradigm for scaling the capabilities of quantum computing systems [8]–[10]. However, DQC faces the fundamental challenge of implementing remote gate operations. Under the DQC paradigm, the logical qubits of a quantum circuit are mapped to physical qubits distributed across multiple quantum processors, which creates a natural distinction between different types of operations. Gate operations can be classified into two categories: local gates and remote gates. Local gates encompass both single-qubit gates and multi-qubit gates applied to qubits mapped within the same processor, where they can be executed using the conventional monolithic quantum computing paradigm. In contrast, remote gates are multi-qubit operations applied to qubits distributed across two or more quantum processors, which differ fundamentally from monolithic quantum computing and require specialized inter-processor communication mechanisms.

C. Qubit Mapping and Remote Gate Operations

Qubit mapping for DQC involves allocating logical qubits from a logical circuit to physical qubits distributed across multiple QPUs. For instance, Circuit 1 in Fig. 2(b) can be executed in a distributed manner by mapping the qubits to two processors, QPU 0 and QPU 1, each with two computing qubits. If logical qubits q_0, q_1 of Circuit 1 are assigned to QPU 0 and q_2, q_3 to QPU 1, G_1, G_2, and G_3 become remote CX gates applied to qubits on different processors.

Several papers in the literature [27], [32] assume that remote gate operations are implemented through protocols where each shared Bell state realizes exactly one remote CX gate, leading to inefficient resource utilization. For example, Circuit 1 in Fig. 2(b) can be realized in a distributed manner as shown in Fig. 2(d), where q_0, q_1 are assigned to QPU 0 and q_2, q_3 to QPU 1. Sub-circuits A, B, and C with the same structure in Fig. 2(d) are functionally equivalent to remote CX gates G_1, G_2, and G_3 in Fig. 2(b), and each consumes one entanglement.

This paper considers a method introduced in [11], which utilizes cat-entanglement to potentially enable multiple CX gates by consuming a single entanglement. The example in Fig. 2(e) demonstrates how this approach can realize two remote CX gates (G_1 and G_2 of Circuit 2 in Fig. 2(c)) by consuming only a single shared Bell state between communication qubits c_0 and c_1, significantly reducing the entanglement overhead compared to conventional methods. The key insight behind this approach lies in creating a proxy relationship between the logical control qubit and a local communication qubit. Initially, a shared Bell state is established between communication qubits c_0 and c_1 on different processors. The state of $|q_1\rangle$ at the beginning can be an arbitrary state, denoted by $|q_1\rangle = \alpha|0\rangle + \beta|1\rangle$. Then, we perform the operations in sub-circuit A in Fig. 2(e), which will result in qubits q_1 and c_1 forming a cat-like state $\alpha|00\rangle + \beta|11\rangle$ [11]. This generalized Bell state establishes a fundamental equivalence relationship where $\mathrm{CX}(q_1, q)$ becomes equivalent to $\mathrm{CX}(c_1, q)$ for any target qubit q. As a result, we can achieve the function of sub-circuit A of Circuit 2 in Fig. 2(c) by using sub-circuit B in Fig. 2(e), where we substitute the gates G_1 and G_2 by CX gates controlled by the qubit c_1, which is located in the same processor as the target qubits. Finally, sub-circuit C in Fig. 2(e) disentangles q_1 and c_1, which frees the communication qubit c_1 for subsequent operations.

It should be noted that when using this cat-entanglement method, it is necessary to ensure that the cat-like state formed via remote entanglement remains unchanged before disentanglement. Therefore, the qubits in the cat-like state can only participate in circuit operations as control qubits. If the state of the qubits in the cat-like state needs to be changed, the system must first disentangle them, update the quantum state, and then consume a new entangled pair to reform the cat-like state before it can participate in remote CX operations. For example, if q_1 and q_3 in Fig. 2(b) are mapped to different QPUs, G_1 and G_3 cannot be realized by consuming a single entanglement because the state of control qubit q_1 changes between the two gates. In contrast, for the circuit in Fig. 2(c), if q_1 and q_3 are mapped to different QPUs, control qubit q_1 does not change between G_1 and G_2, so gates G_1 and G_2 can be realized by consuming one entanglement.

D. Motivation Example

Next, we will present a motivational example to demonstrate that qubit mapping and circuit transformation are coupled and required to be optimized jointly. In particular, consider transforming the logical circuit with 11 `CX` gates and six single-qubit gates in Fig. 2(a) and mapping the logical qubits to two QPUs, namely QPU 0 and QPU 1, each with 2 physical computing qubits and at least one communication qubit.

Using the state-of-the-art Quarl circuit transformation approach proposed in [15], the original circuit can be significantly simplified to the functionally equivalent Circuit 1 shown in Fig. 2(b), which has six CX gates and four single-qubit gates. Then, in terms of minimizing the number of remote gate operations, the optimal qubit mapping scheme is to map qubit 0 and qubit 1 to QPU 0 and qubit 2 and qubit 3 to QPU 1, which leads to three remote gates, i.e., G_1, G_2, and G_3 in Fig. 2(b). However, the three remote gates are controlled by two qubits, and the two of them controlled by the same qubit are interrupted by the CX and $\mathrm{RZ}(\pi/2)$ gate. Therefore, one entanglement shared by the two QPUs can only enable one of the three remote gates. As shown in Fig. 2(d), the distributed implementation of Circuit 1 on two QPUs consumes three shared entanglements.

As stated in Section III-C, consuming one entanglement is possible to realize multiple remote gates. Jointly optimizing the circuit transformation and qubit mapping can minimize the entanglement consumption by leveraging the power of cat-entanglement. For example, the original circuit in Fig. 2(a) can be transformed to functionally equivalent Circuit 2 in

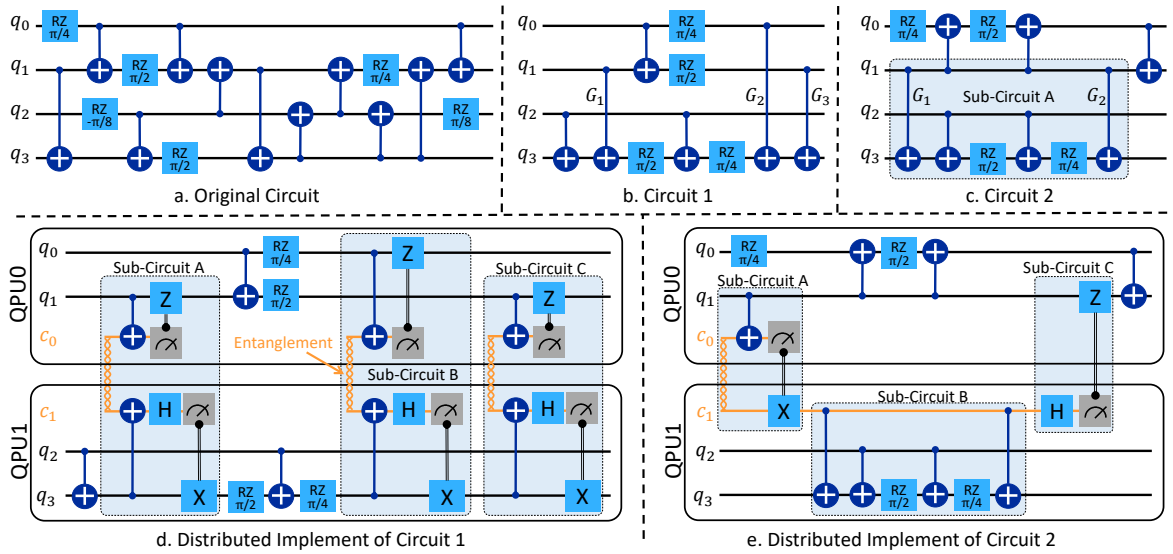

Fig. 2: A Motivational Example. (a) Original logical circuit. (b) Circuit 1: A functionally equivalent circuit given by Quarl [15]. (c) Circuit 2: Another functionally equivalent transformation. (d) Distributed implementation of Circuit 1 requiring three shared entanglements across QPU 0 and QPU 1. (e) Distributed implementation of Circuit 2 requiring only one shared entanglement.

Fig. 2(c), which has seven CX gates and four single-qubit gates. Although Circuit 2 has more gates than Circuit 1, it can be implemented in two QPUs by consuming only one shared entanglement. Specifically, mapping qubit 0 and qubit 1 to QPU 0, and qubit 2 and qubit 3 to QPU 1 causes two remote gates, i.e., G_1 and G_2 in Fig. 2(c). In addition, the two gates are controlled by the same qubits and can be realized by consuming only one entanglement, as shown in Fig. 2(e).

Compared with Circuit 1, Circuit 2 under joint optimization has one more gate, but the joint optimization approach can realize the circuit by consuming two fewer shared entanglements, where generating shared entanglement is much more costly than local gate operations. Therefore, it is desired to optimize circuit transformation and qubit mapping jointly. In this paper, we focus on joint circuit transformation and qubit mapping to minimize entanglement consumption for DQC.

IV. System Model and Problem Formulation

This section introduces the system model for DQC and the joint circuit transformation and qubit mapping problem.

System Model. Consider a quantum circuit C to be executed on a DQC cluster consisting of N QPUs connected by E quantum links. The DQC cluster is represented by a graph $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is the collection of QPUs and $\mathcal{E}$ is the collection of quantum links. Each QPU $n \in \{1, 2 \cdots, N\}$ is equipped with Q_n computing qubits and at least one communication qubit. Communication qubits on different QPUs utilize the quantum links to generate entanglements that enable remote gate operations between QPUs. We assume sufficient communication qubits are available (either through adequate provisioning or temporal multiplexing), and this paper focuses on joint circuit transformation and qubit mapping, which mainly involves computing qubits. The cost of generating an entanglement between QPU n and QPU m is denoted as $p_{n,m}$, which depends on several infrastructure parameters. For example, when QPU n and QPU m are not directly connected, entanglement generation requires selecting a path between them, generating entanglement on each link along the path, and performing entanglement swapping on intermediate QPUs [33]–[35]. The cost increases with path length. Additionally, different quantum links may have varying photon collection and loss rates, requiring different numbers of attempts [36].

Problem Definition. Let τ be a circuit transformation that converts the original circuit C into an equivalent circuit $C' = \tau(C)$, where $\texttt{equiv} : C \times C \rightarrow \{0, 1\}$ is the equivalence function that returns 1 if two circuits are functionally equivalent. The transformed circuit $\tau(C)$ contains $I_{\tau(C)}$ qubits. Let π be a qubit mapping function where $\pi(n) \subseteq \{1, 2, \ldots, I_{\tau(C)}\}$ represents the collection of qubits mapped to QPU n. Given these definitions, the problem seeks to find: (1) a transformation τ such that $\texttt{equiv}\,(\tau(C), C) = 1$, and (2) a qubit mapping π that assigns the $I_{\tau(C)}$ qubits of $\tau(C)$ to QPUs in the cluster.

Objective Function. Generating high-fidelity shared Bell states across processors is expensive due to complex infrastructure requirements, low success rates that necessitate multiple attempts, and rapid decoherence from environmental noise. For example, entanglement generation has a low success probability per attempt, e.g., 4×10^{-5} as documented in [37], requiring tens of thousands of attempts to generate one entanglement. Consequently, remote Bell state generation between QPUs is orders of magnitude more costly than local gate operations within a single processor, with entanglement generation costs up to 300 times higher than local operations [12]. As a result, we aim to minimize the number of entanglements consumed for the distributed execution of quantum circuits. The cost of distributed quantum computing is dominated by the cost of generating entanglements, and this paper focuses on minimiz-

ing entanglement consumption for DQC. Let $\mathcal{R}_\pi(m,n)$ denote the number of entanglements required to enable all remote gates between QPU m and QPU n under mapping π. The value of $\mathcal{R}_\pi(m,n)$ depends on whether control qubits can be reused according to the cat-entanglement-based remote gate operation described in Section III-C. The total entanglement cost is $\sum_{n=1}^{N}\sum_{m=1}^{n} p_{n,m}\mathcal{R}_\pi(m,n)$. Note that the approach proposed in this paper can also optimize performance metrics used in monolithic quantum computing, such as the total gate count and CX gate count considered in [13], [15].

Problem Formulation. The joint circuit transformation and qubit placement problem can be formulated as follows:

$$\begin{aligned} \min_{\tau,\pi} \quad & \sum_{n=1}^{N}\sum_{m=1}^{n} p_{n,m}\mathcal{R}_\pi(m,n) \\ \text{subject to} \quad & \texttt{equiv}\,(\tau(C), C) = 1 \qquad (1) \\ & |\pi(n)| \le Q_n \quad \forall n \in \{1,2,\ldots,N\} \qquad (2) \end{aligned}$$

The objective function minimizes the total entanglement cost. Constraint (1) ensures the transformed circuit remains functionally equivalent to the original circuit. Constraint (2) limits the number of qubits assigned to each QPU to remain within its available capacity.

Problem Complexity. This joint optimization problem poses substantial computational challenges due to the inherent complexity of both subproblems. The circuit transformation task alone confronts an enormous search space, e.g., a circuit with merely 58 gates from the gate set {CX, X, H, RZ} admits over 162 million functionally equivalent representations [15]. Furthermore, even after obtaining an optimized circuit transformation, the subsequent qubit mapping problem remains NP-hard [18]. Consequently, the joint optimization of circuit transformation and qubit mapping presents formidable computational complexity.

V. A Two-Phase RL Framework: *Qmactr*

This section presents the proposed RL-based approach for jointly optimizing circuit transformation and qubit mapping.

As discussed in the previous section, the extremely large search space of equivalent circuits and the NP-hard nature of the mapping subproblem make the joint optimization problem intractable for deterministic algorithms. Recent advances in reinforcement learning (RL) for quantum compilation, particularly Quarl [15] for circuit transformation, have demonstrated state-of-the-art performance. Fig. 3 shows an overview of the proposed two-phase approach. In the first phase, *Qmactr* trains a robust RL-based qubit mapping agent π tailored for the cat-entanglement-based remote gate protocol, as shown in Fig. 3(a). In the second phase, a new RL-based circuit transformation agent τ is learned, as shown in Fig. 3(b), where the mapping agent from the first phase serves as the primary reward signal provider. Both agents are formulated as Markov Decision Processes (MDPs), with corresponding RL approaches proposed and tailored to each problem.

Algorithm 1: RL-based Qubit Mapping Training

Input: DQC Cluster G, Circuit C
Output: Trained parameters θ

Initialize Actor-Critic agent with policy π_θ;
for *episode* $e = 1$ **to** $N_{\max}$ **do**
  Initialize trajectory buffer $\mathcal{B} \leftarrow \emptyset$;
  Initialize state s_0 based on G and C;
  for $t = 0, 1, 2, \ldots$ ***until*** *mapping complete* **do**
    Extract $\mathbf{h}_G, \mathbf{h}_C$ from s_t; `// GNN used`
    Sample action $a_t \sim \pi_\theta(a_t \mid \mathbf{h}_G, \mathbf{h}_C, s_t)$;
    Execute a_t and get s_{t+1}; `// apply action`
    Compute reward r_t ; `// evaluate action`
    Store $(s_t, a_t, r_t, s_{t+1}, \log \pi_\theta(a_t \mid s_t))$ in $\mathcal{B}$;
  end
  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\text{PPO}}(\mathcal{B})$; `// update weights` θ
end
return *Optimized parameters* θ

A. RL-based Qubit Mapping Agent

MDP Formulation for Qubit Mapping. To apply RL to the qubit mapping problem, we formulate it as a MDP where at each time step t, the agent observes state s_t^π, takes action a_t^π, and receives reward r_t^π.

The state s_t^π provides a comprehensive snapshot of the mapping process, including three components: the DQC cluster G with QPU capacities and inter-QPU entanglement generation costs, the connection matrix of qubits in circuit C, and the mapping status tracking current logical qubit assignments and remaining QPU capacities.

The action space at each step t consists of all valid (QPU,logical-qubit) pairs. The policy $\pi_\theta(a_t^\pi \mid s_t^\pi)$ determines the probability of selecting a specific action given the current state. To ensure feasibility, invalid actions, such as remapping already placed logical qubits or assigning logical qubits to QPUs without free physical qubits, are masked during action selection.

The reward function is strategically designed to guide the agent towards mappings that minimize entanglement consumption. Upon executing action a_t^π and transitioning to state s_{t+1}^π, the environment provides a reward r_t^π that encourages the placement of frequently communicating logical qubits on physically proximate QPUs, with the reward increasing inversely with the inter-QPU distance.

Architecture of the Mapping Agent. The qubit mapping agent employs an Actor-Critic model based on Proximal Policy Optimization (PPO). Next, we present the details of the qubit mapping agent.

To effectively represent the characteristics of the DQC cluster G and the circuit C, the agent employs a dual Graph Neural Network (GNN) architecture. This architecture generates tailored node embeddings, $\mathbf{h}_G$ and $\mathbf{h}_C$, where this generation occurs in steps ① and ④ in Fig. 3(a). The GNNs take the current state s_t as input and output embeddings (or features). Unlike conventional Graph Attention Networks that derive attentions solely from node features, the GNNs in the

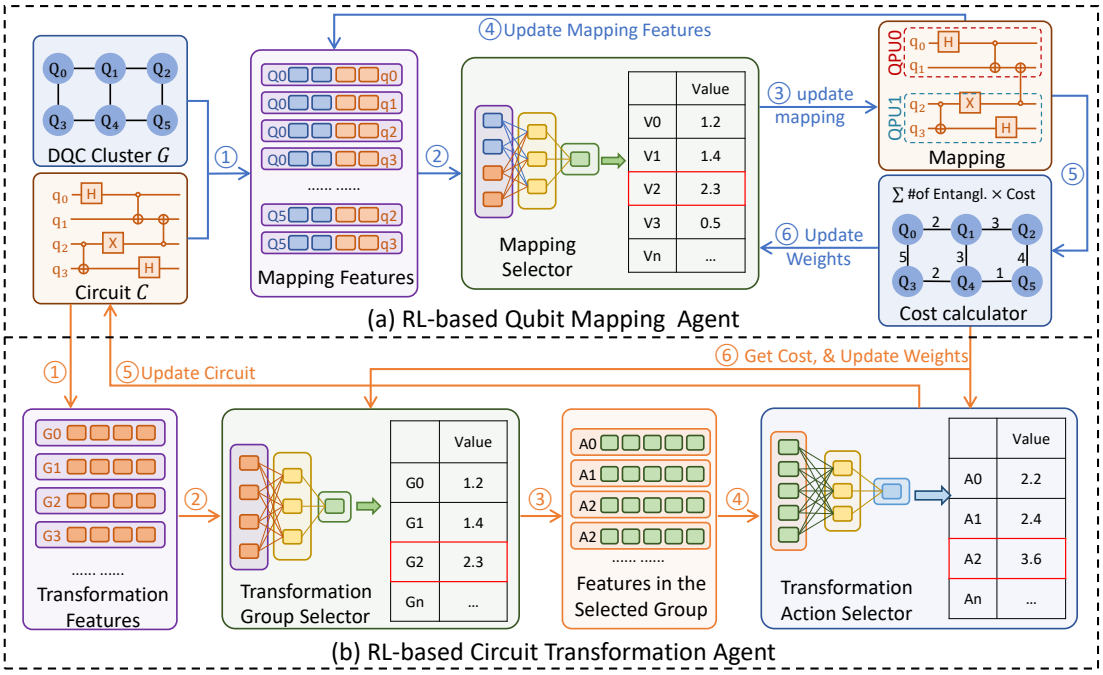

Fig. 3: The architecture of *Qmactr*. The system employs (a) a Qubit Mapping Agent and (b) a Circuit Transformation Agent.

qubit mapping agent exploit Edge-Aware Graph Attention (EA-GAT) layers. The EA-GAT layer explicitly incorporates quantitative edge attributes, such as the inter-QPU entanglement generation costs, directly into the attention computation. This allows the model to have a better understanding of the DQC cluster's weighted topology. Specifically, the attention score e_{ij} for an edge from node j to node i with weight w_{ij} is formulated as follows:

$$e_{ij} = \text{LeakyReLU}\left(\vec{a}_{\text{src}}^T \mathbf{W} h_j + \vec{a}_{\text{dst}}^T \mathbf{W} h_i + a_{\text{edge}} \cdot w_{ij}\right)$$

where $\mathbf{W}$ is a learnable linear transformation, h_i and h_j are the node features, and $\vec{a}_{\text{src}}$, $\vec{a}_{\text{dst}}$, and a_{edge} are learnable attention parameters. The key innovation is the $a_{edge} \cdot w_{ij}$ term, which provides a direct, learnable pathway for edge weights to influence inter-node attention. These scores are then normalized into attention weights α_{ij} via a softmax function across node i's neighborhood. The final layer output for node i is a multi-head aggregation of transformed neighbor features, weighted by these learned coefficients:

$$h_i' = \|_{k=1}^{K} \sum_{j \in N(i) \cup i} \alpha_{ij}^{(k)} \mathbf{W}^{(k)} h_j$$

where K is the number of attention heads and $\|$ denotes concatenation. By stacking these EA-GAT layers with residual connections, the GNN architecture produces highly informative embeddings that capture not only the graph's topology but also the critical, explicit relationships defined by its edge weights, providing a rich feature set for the policy network.

Following the GNN for feature extraction, the qubit mapping agent has an Actor network and a Critic network. The Actor implements the policy π_θ. Instead of a monolithic network, it uses a pairwise scoring mechanism. To select an action a_t, it evaluates the value of each valid (QPU, logical-qubit) pair. Its input is a concatenated feature vector containing the qubit embedding $\mathbf{h}_C$, the QPU embedding $\mathbf{h}_G$, and a vector containing information about the current qubit mapping situation derived from s_t^π. The agent outputs the scores for each pair, shown in the mapping selector in Fig. 3(a). These scores are then converted into a probability distribution from which an action is sampled. The Critic estimates the state-value function. It takes a global perspective, using aggregated embeddings of all qubits and QPUs, along with the vector containing information about the current qubit mapping situation from s_t^π, to predict the expected future reward.

Mapping model training. The training follows a standard episodic reinforcement learning loop. For each episode, the agent interacts with the environment, generating a sequence of transitions $(s_t^\pi, a_t^\pi, r_t^\pi, s_{t+1}^\pi)$. The $\log \pi_\theta(a_t^\pi \mid s_t^\pi)$ of each action is also recorded. These transitions are stored in a temporary buffer $\mathcal{B}$. After collecting a trajectory, the entire buffer is used to update the model parameters θ by optimizing the PPO objective function, which happens in step ⑥ of Fig. 3(a). This involves calculating advantages using Generalized Advantage Estimation (GAE) and performing a clipped policy update, which ensures stable and efficient learning. Upon completion of training, the algorithm returns the optimized parameters θ of the mapping model. Algorithm 1 presents the training of the proposed qubit mapping agent.

B. RL-based Circuit Transformation Agent

The circuit transformation agent in the second phase addresses the joint optimization problem by intelligently searching the space of functionally equivalent circuits. During this process, the pre-trained qubit mapping agent serves as a fast

oracle, estimating the entanglement cost for each candidate circuit after mapping.

MDP Formulation for Circuit Transformation. The circuit transformation is formulated as a hierarchical MDP to effectively manage the complex and dynamic action space.

The state at step t, denoted as s_t^τ, represents the current quantum circuit. Following Qural [15], state s_t^τ is represented as a Directed Acyclic Graph (DAG), C_t. An opportunity is defined as a subgraph within the current DAG that corresponds to a source pattern in a library of equivalent circuit transformations. Each opportunity may have multiple alternative transformations, collectively referred to as an action group. The process of identifying these opportunities is dynamic, where old opportunities may vanish, and new ones may emerge as the circuit undergoes transformations.

The action space at each step t is defined hierarchically to ensure computational traceability. In particular, the agent breaks the decision down into two simpler, sequential steps. Instead of navigating a compound action in an enormous action space, the agent first selects a matched subgraph (an opportunity) in the circuit to modify and then chooses a specific transformation from the corresponding action group. This decomposition is necessary because a typical circuit can contain thousands of such opportunities, each with multiple replacement options. The compound action space combining all possibilities would be computationally infeasible and prevent effective learning.

The reward function is the key link between the two-phase framework. After executing an action a_t^τ, the circuit C_t is transformed into a new circuit C_{t+1}. The reward is not based on simple heuristics like gate count but on the estimated real-world entanglement cost. The new circuit C_{t+1} is fed into the pre-trained qubit mapping agent, which returns a cost estimation of the total entanglement cost denoted as $\text{Cost}(\pi(C_{t+1}))$, shown in step ⑥ in Fig. 3(b). The reward r_t^τ is defined to quantify the normalized reduction in the estimated cost following an action, as follows:

$$r_t^\tau = \frac{\text{Cost}(\pi(C_t)) - \text{Cost}(\pi(C_{t+1}))}{\text{Cost}(\pi(C_t))}.$$

Architecture of the Circuit Transformation Agent. The agent consists of three main components: a GNN for feature extraction, a Meta-Critic network, and an Intra-Group Actor network.

The GNN is used to generate feature embeddings for circuit subgraphs, which happens in step ① of Fig. 3(b). For each action group (opportunity) and each potential transformation, we generate a feature vector by concatenating its GNN embedding with key features like its CX gate count and total gate count.

The Meta-Critic network (the transformation group selector in Fig. 3(b)) evaluates the value of choosing a particular action group g. It takes the feature vector of the subgraph corresponding to g as input and outputs a scalar value $V(s_t, g)$, which represents the expected future reward if we decide to operate on this subgraph. This value guides the gate group selection policy τ_g.

Algorithm 2: RL-based *Qmactr*

Input: Circuit C and pretrained mapping agent π
Output: Optimized circuit C^* and mapping $\pi(C^*)$

$C^* \leftarrow C$; `// best circuit found so far`
$\text{cost}^* \leftarrow \text{Cost}(\pi(C^*))$;
for *episode* $e = 1$ **to** $N_{\max}$ **do**
  $C_0 \leftarrow C$; `// reset to input circuit`
  for *step* $t = 0$ **to** $T_{\max} - 1$ **do**
    `// select action and env. react.`
    Sample action $a_t^\tau = (g^*, x^*) \sim \pi_\phi(a_t^\tau \mid C_t)$;
    Apply transformation a_t^τ to C_t and get C_{t+1};
    $\text{cost}_{\text{new}} \leftarrow \text{Cost}(\pi(C_{t+1}))$;
    $r_t^\tau \leftarrow (\text{Cost}(\pi(C_t)) - \text{cost}_{\text{new}})/\text{Cost}(\pi(C_t))$;
    Store transition $(C_t, a_t^\tau, r_t^\tau, C_{t+1})$ in buffer $\mathcal{B}$;
    if $cost_{new} < cost^*$ **then**
      $\text{cost}^* \leftarrow \text{cost}_{\text{new}}$;
      $C^* \leftarrow C_{t+1}$; `// update` C^*
    end
    `// batch learning`
    if *buffer full* ***or*** *episode terminates* **then**
      $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_{\text{PPO}}(\mathcal{B})$; `// update` ϕ
      $\mathcal{B} \leftarrow \emptyset$; `// reset buffer`
    end
  end
end
return C^*, $\pi(C^*)$

The Intra-Group Actor network (the transformation action selector in Fig. 3(b)) is responsible for the circuit transformation selection policy τ_x. Its input is concatenated vectors for the special transformation x chosen from the selected group g. It outputs a logit, $l(g, x)$, scoring the desirability of this specific transformation. The policy $\tau_x(x \mid s_t^\tau, g; \phi)$, parameterized by ϕ, defines a softmax distribution over the logits of all feasible transformations within the selected group g:

$$\tau_x(x|s_t, g; \phi) = \frac{\exp(l(g, x; \phi))}{\sum_{x' \in X(g)} \exp(l(g, x'; \phi))}.$$

Transformation model training. The agent is trained using a PPO algorithm adapted for the hierarchical structure.

To collect experience, the agent interacts with the quantum circuit environment to generate trajectories. For each step in these trajectories, we store the feature of the chosen group, the features of all possible replacement actions within that group, the chosen action index, the log-probability of that action, and the received reward.

The advantage function, a key aspect of the training, is $A_t = r_t + \gamma V(s_{t+1}) - V(s_t)$. $V(s_t)$ is the value of the chosen action group, provided by the Meta-Critic network. $V(s_{t+1})$ is estimated by identifying all new and influenced opportunities in the resulting circuit C_{t+1}. We pass the feature vectors of all these new opportunities through the Meta-Critic network and take the maximum value as the estimate for the next state's value. This provides a forward-looking evaluation of the quality of the next state.

TABLE I: Characteristics of Tested Quantum Circuits

Name	# of qubits	# of gates	# of CXs
qft_6	6	92	39
qft_9	9	203	84
barenco_tof_5	9	170	72
gf2^4_mult	12	225	99
rc_adder_6	14	200	93
qft_16	16	513	240
tof_10	19	255	102
qft_20	20	1000	410
gf2^7_mult	21	669	300
gf2^8_mult	24	883	405
adder_8	24	900	409
qcla_mod_7	26	884	382
gf2^10_mult	30	1347	609
qcla_adder_10	36	450	233
gf2^16_mult	48	3435	1581

To train the policy, the PPO objective function with a clipped ratio is used to update the parameters of both the Intra-Group Actor and the Meta-Critic networks. The Actor is updated to favor actions with high advantage, while the Critic is updated to provide more accurate state-value estimates, thus creating a virtuous cycle of learning.

During training, the trained agent executes a two-stage decision process. First, the Meta-Critic network evaluates all available transformation opportunities in the circuit to select the best transformation opportunity g^*. Then, for this chosen opportunity, the Intra-Group Actor network assesses all possible circuit transformation actions and selects the best transformation x^*. The resulting hierarchical action (g^*, x^*) is then applied to modify the circuit. We continuously train and record the circuit with the lowest entanglement cost so far. After the training process, we select the circuit with the lowest entanglement cost we have ever observed as the output circuit. The detailed training and decision-making process of *Qmactr* is shown in Algorithm 2.

VI. Performance Evaluation

This section evaluates the performance of *Qmactr*. The source code for our evaluation is publicly available on GitHub at https://github.com/deltapolyu/Qmactr.

A. Evaluation Settings

All experiments were conducted on a system running Ubuntu 22.04 LTS, equipped with a 12-core Intel Xeon Platinum 8352V processor, 90 GB of system memory, and an NVIDIA RTX 4090 GPU with 24 GB of dedicated memory. The reinforcement learning framework was implemented using Python 3.12 and PyTorch 2.3.0, with GPU acceleration provided by CUDA 12.1.

Following the experimental setup of Quarl [15], we adopt a standard benchmark suite primarily composed of arithmetic and reversible circuits [13], [23], [38]–[40]. Table I summarizes the characteristics of the tested circuits. To thoroughly assess the adaptability of *Qmactr* to different hardware constraints, we evaluate all methods using four network topologies: star [27], line, grid [41], and random networks. The

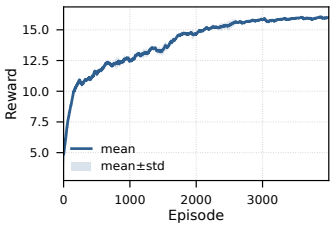

Fig. 4: Convergence of the Qubit Mapping Agent.

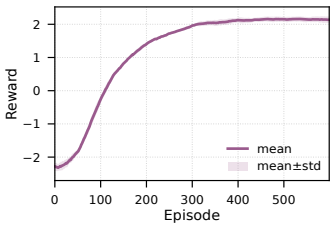

Fig. 5: Convergence of the Circuit Transformation Agent.

random network topology is generated following the Jellyfish network model [42]. The network topology scales adaptively with circuit size to match the computational requirements of each benchmark circuit listed in Table I. For example, for smaller circuits like `gf2^4_mult` with 12 qubits, we employ a 2×3 grid. As the circuit size increases, such as for `gf2^16_mult` with 48 qubits, we use a larger 3×3 grid to accommodate the higher qubit count and communication needs. Note that each qubit can be represented by multiple physical qubits using quantum error-correcting codes, and the quantum capacity of each QPU refers to the number of logical qubits it can accommodate. Similarly, the length of the line topology is adjusted based on the qubits in the circuit under test. The total qubit capacity is randomly allocated among QPUs. The star topology is evaluated across all benchmarks as a common architecture with a central hub. Due to space limitations, additional implementation details are omitted. For further details, please refer to our source code.

B. Baseline Methods

To demonstrate the efficacy of the proposed *Qmactr*, we compare its performance against three baselines. The first baseline, named Direct+Map, employs a mapping-only strategy where the original quantum circuit is directly input to the mapping algorithm proposed in [27] without any circuit transformation. The second baseline, named Quartz+Map, combines Quartz [13], a prominent circuit transformation tool, with subsequent mapping. The third baseline, named Quarl+Map, implements a state-of-the-art sequential optimization pipeline where circuits are first optimized using the RL-based Quarl [15], then mapped. These baselines enable us to assess whether the proposed joint optimization approach can outperform traditional sequential processing methods.

C. Evaluation Results

Convergence. We first evaluate the convergence behavior of the proposed *Qmactr*. Fig. 4 shows the training rewards of the qubit mapping agent versus episodes under the `gf2^4_mult` circuit on the star topology. The reward exhibits a consistent upward trend over 4000 episodes, increasing from a relatively low initial value and gradually approaching a stable high-reward region. This improvement indicates that the qubit mapping agent, equipped with GNN-based feature representation and guided by the proposed reward function, can learn the relationship between the logical circuit and the physical DQC cluster and make high-quality qubit mapping decisions.

TABLE II: Entanglement cost comparison across different optimization methods. The best result is highlighted in bold.

Circuit	Star				Line				Grid				Random			
	Direct	Quartz	Quarl	*Qmactr*	Direct	Quartz	Quarl	*Qmactr*	Direct	Quartz	Quarl	*Qmactr*	Direct	Quartz	Quarl	*Qmactr*
adder_8	438	432	410	**354**	762	738	716	**352**	578	550	536	**294**	414	356	352	**274**
barenco_tof_5	188	182	170	**108**	188	182	170	**108**	276	252	242	**230**	168	155	141	**120**
gf2^10_mult	3872	3670	3516	**3072**	7010	6882	6854	**5954**	5332	5212	4924	**4416**	4402	4302	4104	**3714**
gf2^16_mult	13068	13120	12810	**10502**	27216	26672	26150	**22910**	14510	14362	13918	**11862**	13000	12900	12546	**11012**
gf2^4_mult	408	388	372	**330**	454	446	420	**358**	670	636	634	**582**	408	398	376	**328**
gf2^7_mult	1924	1912	1870	**1640**	3024	2960	2882	**2426**	2250	2232	2198	**1710**	1764	1752	1702	**1518**
gf2^8_mult	2498	2292	2234	**2078**	4838	4708	4630	**3952**	3292	3262	2990	**2668**	2186	2058	1988	**1710**
qcla_adder_10	480	468	478	**406**	648	632	618	**574**	472	460	446	**390**	520	514	528	**442**
qcla_mod_7	886	854	792	**560**	1226	1182	1045	**916**	1148	1116	1046	**968**	648	638	598	**530**
qft_16	1308	1296	1284	**1092**	2428	2392	2352	**2034**	2292	2218	2192	**1916**	1528	1516	1474	**1200**
qft_20	2296	2274	2218	**2016**	4672	4540	4514	**3750**	3312	3288	3218	**2610**	2410	2200	2146	**1756**
qft_6	206	198	188	**162**	226	202	192	**184**	390	378	362	**312**	218	208	202	**158**
qft_9	416	408	392	**312**	456	448	432	**328**	676	642	632	**572**	360	354	348	**296**
rc_adder_6	222	212	198	**152**	180	176	176	**112**	250	236	226	**184**	204	192	184	**160**
tof_10	184	176	172	**128**	230	224	220	**156**	212	202	196	**144**	178	172	158	**120**

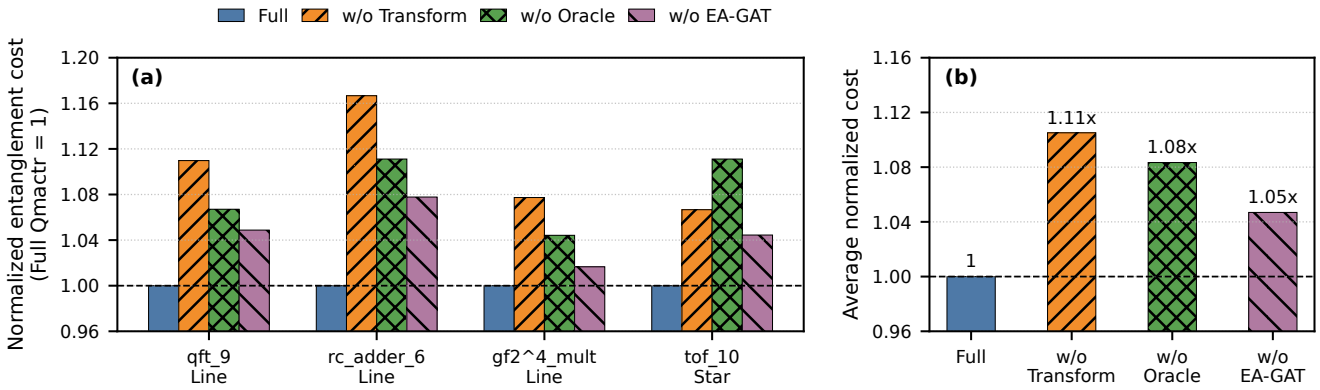

Fig. 6: Ablation study of Qmactr components.

In addition, Fig. 5 shows the training rewards of the circuit transformation agent under the same circuit-topology setting. The transformation reward increases steadily over 600 episodes, rising from negative values to positive rewards and gradually flattening in the later stage of training. This trend indicates that the transformation agent learns to select circuit rewrites that are increasingly favorable to the downstream mapping objective. Together, these results demonstrate that both stages of *Qmactr* exhibit stable learning behavior and converge toward effective optimization policies.

Performance Comparison. Next, we evaluate *Qmactr* against the three baselines across multiple circuits and hardware topologies. Table II presents a comprehensive performance comparison between the proposed method, *Qmactr*, and the three baselines across eight different circuits and four distinct QPU interconnection topologies, i.e., star, grid, and line, random networks. As is evident across all subplots in Table II, *Qmactr* consistently achieves the lowest entanglement cost, significantly outperforming all baselines in every tested scenario. Specifically, *Qmactr* achieves a performance improvement of up to 50.84% compared to the next best approach, with average improvements of 22.82%, 20.05%, and 17.33% over Direct+Map, Quartz+Map, and Quarl+Map, respectively. The results also demonstrate that simply applying existing circuit transformation and qubit mapping methods in serial yields limited benefits. For instance, both Quartz+Map and Quarl+Map often show only marginal improvements over the Direct+Map baseline, and in some cases, can even lead to slightly higher costs. These findings highlight the inherent limitations of decoupled optimization approaches and demonstrate the necessity of the proposed joint optimization approach.

Ablation Study. We further conduct ablation studies to quantify the contribution of the key components in *Qmactr*. We compare the full design with three ablated variants. First, *w/o Transformation* removes the circuit transformation agent and directly applies the learned mapping agent to the original circuit, which evaluates whether circuit rewriting is necessary beyond mapping alone. Second, *w/o Oracle Reward* keeps the transformation agent but replaces the mapping-oracle-guided reward with structural rewards based on gate-count and CX-count reduction, testing whether transformation decisions should be optimized for downstream mapping cost rather than

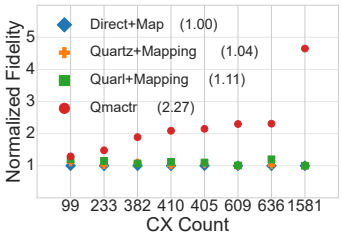

Fig. 7: Fidelity Comparison.

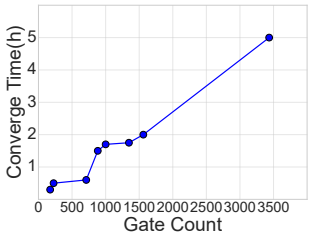

Fig. 8: Scalability Analysis.

local circuit-size metrics. Third, *w/o EA-GAT* removes the edge-weight term from the graph attention computation, reducing the mapping network to a topology-aware but edge-weight-agnostic GAT and evaluating the importance of explicitly modeling inter-QPU entanglement costs.

Fig. 6 reports the normalized entanglement cost of each variant relative to the full *Qmactr* design on representative circuit-topology pairs, where values larger than 1 indicate performance degradation. The results show that all three components contribute to the final optimization quality. Removing the transformation phase increases the average cost by about 11%, confirming that mapping alone cannot fully exploit equivalent circuit structures. Replacing the mapping-oracle reward with structural rewards increases the average cost by about 8%, indicating that reducing gate count or CX count does not necessarily minimize the final distributed execution cost. Removing EA-GAT increases the average cost by about 5%, showing that incorporating edge-weight information into the attention mechanism helps the mapping agent distinguish heterogeneous inter-QPU communication costs.

Scalability and Fidelity. We also evaluate the circuit execution fidelity under *Qmactr* and baseline methods. Following the evaluation methodology established in the literature [13], [15], we adopt the absolute circuit fidelity metric, calculated as $\prod_{\rho \in C}(1 - \varepsilon(\rho))$, where $\varepsilon(\rho)$ represents the error rate of operation ρ in the circuit execution. For entanglement operations, we set the error rate to $1 - F_{\text{ent}}$, where F_{ent} is set to 0.90 as reported in [43]. The error rates for local quantum operations follow the same parameters used in [13], [15]. For operations that share one cat-entanglement, we model decoherence effects by assuming the error rate increases with the operation sequence number. Specifically, the error rate $\varepsilon(\rho)$ of the n-th operation ρ using the entanglement is set to $1 - (1 - \varepsilon_0(\rho)) \cdot 0.9^{n-1}$, where $\varepsilon_0(\rho)$ represents the initial error rate of operation ρ, and the factor 0.9 represents the cumulative degradation of the shared entanglement resource due to decoherence over successive operations. Fig. 7 shows the absolute circuit fidelity of approaches normalized by that of the Direct+Map baseline. The results demonstrate that *Qmactr* consistently outperforms baselines. Moreover, the performance gap between *Qmactr* and the baselines increases as the circuit scale grows. We also evaluate the scalability of *Qmactr*. Fig. 8 demonstrates that the convergence time of *Qmactr* scales linearly with circuit size in terms of gate count, showing excellent scalability. Moreover, our approach converges faster than the baselines due to the high computational cost of their mapping algorithms, with Quarl+Map's execution time dominated by the Quarl optimizer.

VII. Conclusion

This paper formulated and studied the joint circuit transformation and qubit mapping problem in DQC, where both subproblems are difficult to solve. This paper introduced *Qmactr*, the first optimization framework for the formulated problem. *Qmactr* is a two-phase reinforcement learning approach, which enables intelligent exploration of functionally equivalent circuits optimized for distributed quantum circuit execution. Evaluation results on benchmark circuits across multiple QPU interconnection topologies demonstrate that *Qmactr* consistently outperforms state-of-the-art methods under different settings, achieving up to 50.84% reduction in overall entanglement cost. In addition, *Qmactr* achieves higher circuit execution fidelity than the sequential baselines.

References

[1] S. Pirandola, U. L. Andersen, L. Banchi, M. Berta, D. Bunandar, R. Colbeck, D. Englund, T. Gehring, C. Lupo, C. Ottaviani *et al.*, "Advances in quantum cryptography," *Advances in optics and photonics*, vol. 12, no. 4, pp. 1012–1236, 2020.

[2] A. Abbas, A. Ambainis, B. Augustino, A. Bärtschi, H. Buhrman, C. Coffrin, G. Cortiana, V. Dunjko, D. J. Egger, B. G. Elmegreen *et al.*, "Challenges and opportunities in quantum optimization," *Nature Reviews Physics*, pp. 1–18, 2024.

[3] M. Schuld, I. Sinayskiy, and F. Petruccione, "An introduction to quantum machine learning," *Contemporary Physics*, vol. 56, no. 2, pp. 172–185, 2015.

[4] Z. Yang, M. Zolanvari, and R. Jain, "A survey of important issues in quantum computing and communications," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 2, pp. 1059–1094, 2023.

[5] T. Häner, D. S. Steiger, T. Hoefler, and M. Troyer, "Distributed quantum computing with qmpi," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–13.

[6] 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.

[7] Y. Mao, Y. Liu, X. Shang, and Y. Yang, "Network topology design for distributed quantum computing," in *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2024, pp. 1213–1223.

[8] T. J. Yoder, E. Schoute, P. Rall, E. Pritchett, J. M. Gambetta, A. W. Cross, M. Carroll, and M. E. Beverland, "Tour de gross: A modular quantum computer based on bivariate bicycle codes," *arXiv preprint arXiv:2506.03094*, 2025.

[9] D. Main, P. Drmota, D. Nadlinger, E. Ainley, A. Agrawal, B. Nichol, R. Srinivas, G. Araneda, and D. Lucas, "Distributed quantum computing across an optical network link," *Nature*, pp. 1–6, 2025.

[10] J. P. Covey, H. Weinfurter, and H. Bernien, "Quantum networks with neutral atom processing nodes," *npj Quantum Information*, vol. 9, no. 1, p. 90, 2023.

[11] A. Yimsiriwattana and S. J. Lomonaco Jr, "Generalized ghz states and distributed quantum computing," *arXiv preprint quant-ph/0402148*, 2004.

[12] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim, "Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects," *Physical Review A*, vol. 89, no. 2, p. 022317, 2014.

[13] 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.

[14] A. Xu, A. Molavi, L. Pick, S. Tannu, and A. Albarghouthi, "Synthesizing quantum-circuit optimizers," *Proceedings of the ACM on Programming Languages*, vol. 7, no. PLDI, pp. 835–859, 2023.

[15] 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.

[16] F. Zhan, Y. Zhao, and C. Qiao, "Towards high-performance distributed quantum computing with qubit placement and provisioning," in *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*. IEEE, 2024, pp. 1–10.

[17] L. Liu, "Larqucut: A new cutting and mapping approach for large-sized quantum circuits in distributed quantum computing (dqc) environments," *ACM Transactions on Architecture and Code Optimization*, 2025.

[18] F. Burt, K.-C. Chen, and K. K. Leung, "A multilevel framework for partitioning quantum circuits," *arXiv preprint arXiv:2503.19082*, 2025.

[19] 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 *et al.*, "Quantum computing with qiskit," *arXiv preprint arXiv:2405.08810*, 2024.

[20] 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.

[21] R. S. Smith, E. C. Peterson, M. G. Skilbeck, and E. J. Davis, "An open-source, industrial-strength optimizing compiler for quantum programs," *Quantum Science and Technology*, vol. 5, no. 4, p. 044001, 2020.

[22] K. Hietala, R. Rand, L. Li, S.-H. Hung, X. Wu, and M. Hicks, "A verified optimizer for quantum circuits," *ACM Transactions on Programming Languages and Systems*, vol. 45, no. 3, pp. 1–35, 2023.

[23] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov, "Automated optimization of large quantum circuits with continuous parameters," *npj Quantum Information*, vol. 4, no. 1, p. 23, 2018.

[24] M. Zomorodi-Moghadam, M. Houshmand, and M. Houshmand, "Optimizing teleportation cost in distributed quantum circuits," *International Journal of Theoretical Physics*, vol. 57, no. 3, pp. 848–861, 2018.

[25] P. Andres-Martinez and C. Heunen, "Automated distribution of quantum circuits via hypergraph partitioning," *Physical Review A*, vol. 100, no. 3, p. 032308, 2019.

[26] Z. Davarzani, M. Zomorodi-Moghadam, M. Houshmand, and M. Nouri-Baygi, "A dynamic programming approach for distributing quantum circuits by bipartite graphs," *Quantum Information Processing*, vol. 19, no. 10, p. 360, 2020.

[27] Y. Mao, Y. Liu, and Y. Yang, "Qubit allocation for distributed quantum computing," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[28] T. Fösel, M. Y. Niu, F. Marquardt, and L. Li, "Quantum circuit optimization with deep reinforcement learning," *arXiv preprint arXiv:2103.07585*, 2021.

[29] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical review A*, vol. 52, no. 5, p. 3457, 1995.

[30] E. Younis and C. Iancu, "Quantum circuit optimization and transpilation via parameterized circuit instantiation," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2022, pp. 465–475.

[31] IonQ, "Getting started with native gates," https://docs.ionq.com/guides/getting-started-with-native-gates#general-algorithm, 2025, accessed: June 26, 2025.

[32] D. Cuomo, M. Caleffi, K. Krsulich, F. Tramonto, G. Agliardi, E. Prati, and A. S. Cacciapuoti, "Optimized compiler for distributed quantum computing," *ACM Transactions on Quantum Computing*, vol. 4, no. 2, pp. 1–29, 2023.

[33] S. Shi and C. Qian, "Concurrent entanglement routing for quantum networks: Model and designs," in *Proceedings of the Annual conference of the ACM SIGCOMM*, 2020, pp. 62–75.

[34] L. Yang, Y. Zhao, H. Xu, L. Huang, and C. Qiao, "Dynamic entanglement routing based on stream processing for quantum networks," *IEEE/ACM Transactions on Networking*, 2024.

[35] L. Yang, Y. Zhao, L. Huang, and C. Qiao, "Asynchronous entanglement provisioning and routing for distributed quantum computing," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[36] J. Hannegan, J. D. Siverns, J. Cassell, and Q. Quraishi, "Improving entanglement generation rates in trapped-ion quantum networks using nondestructive photon measurement and storage," *Physical Review A*, vol. 103, no. 5, p. 052433, 2021.

[37] V. Krutyanskiy, M. Galli, V. Krcmarsky, S. Baier, D. Fioretto, Y. Pu, A. Mazloom, P. Sekatski, M. Canteri, M. Teller *et al.*, "Entanglement of trapped-ion qubits separated by 230 meters," *Physical Review Letters*, vol. 130, no. 5, p. 050803, 2023.

[38] M. Amy, D. Maslov, and M. Mosca, "Polynomial-time t-depth optimization of clifford+ t circuits via matroid partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 10, pp. 1476–1489, 2014.

[39] K. Hietala, R. Rand, S.-H. Hung, X. Wu, and M. Hicks, "A verified optimizer for quantum circuits," *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021.

[40] A. Kissinger and J. Van De Wetering, "Pyzx: Large scale automated diagrammatic reasoning," *arXiv preprint arXiv:1904.04735*, 2019.

[41] M. Pant, H. Krovi, D. Towsley, L. Tassiulas, L. Jiang, P. Basu, D. Englund, and S. Guha, "Routing entanglement in the quantum internet," *npj Quantum Information*, vol. 5, no. 1, p. 25, 2019.

[42] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 225–238.

[43] A. Hashim, M. Yuan, P. Gokhale, L. Chen, C. Juenger, N. Fruitwala, Y. Xu, G. Huang, K. Nowrouzi, L. Jiang *et al.*, "Efficient generation of multi-partite entanglement between non-local superconducting qubits using classical feedback," *arXiv preprint arXiv:2403.18768*, 2024.