Empowering Quantum Serverless Circuit Deployment Optimization via Graph Contrastive Learning and Learning-to-Rank Co-designed Approaches

Tingting Li , **Ziming Zhao*** , **Jianwei Yin***
Zhejiang University
{litt2020, zhaoziming}@zju.edu.cn, zjuyjw@cs.zju.edu.cn

Abstract

With the rapid advancements in quantum computing, cloud-based quantum services have gained increasing prominence. However, due to quantum noise, optimizing the deployment of quantum circuits remains an NP-hard problem with an expansive search space. Existing methods usually use heuristic algorithms to approximate the solution, such as the representative IBM Qiskit. On the one hand, they often find suboptimal deployment solutions. On the other hand, prior technologies do not consider user-specific requirements and can only provide a single deployment strategy. In this paper, we propose QCDeploy that can provide a ranked list of effective deployment strategies to optimize quantum serverless circuit deployment. Specifically, we model quantum circuits as Directed Acyclic Graph (DAG) representations and utilize graph contrastive learning for vector embedding. Then, a tailored list-aware learning-to-rank architecture is employed to generate a list of candidate strategies (prioritizing better strategies). We conduct extensive evaluations involving 45 prevalent quantum algorithm circuits across 3∼5 qubits, utilizing 3 IBM quantum physical devices with three types of chip topologies. The results demonstrate that our proposed framework significantly outperforms IBMQ's default deployment scheme, *e.g.,* achieving 17.95% overhead reduction and increasing the execution success rate by 20%∼40%.

1 Introduction

Quantum computing offers a significant advantage over classical computing in addressing computationally intensive problems, owing to its inherent properties of quantum superposition and entanglement [Zhao *et al.*, 2025; Li *et al.*, 2024c]. Some application scenarios involve physical simulation [Zhang *et al.*, 2023b; Deng *et al.*, 2022], quantum machine learning [Saravanan and Saeed, 2023; Ren *et al.*, 2022], optimization problems [Li *et al.*, 2023; Tan *et al.*, 2023]. The broader quantum computing ecosystem

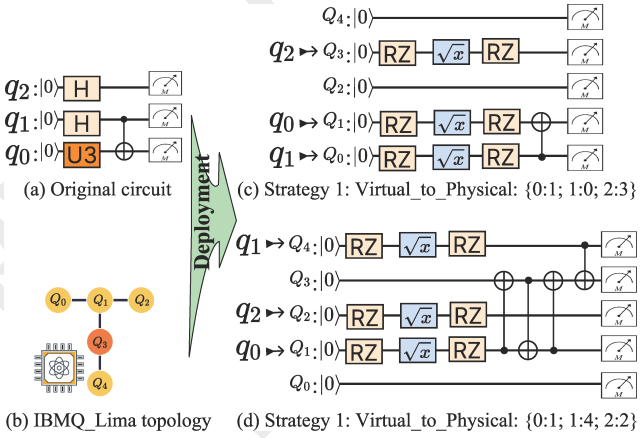

Figure 1: Illustrative explanation for circuit deployment.

is currently constrained by the limited availability of quantum hardware resources, which has led to the emergence of Quantum-as-a-Service (QaaS) platforms [Li and Zhao, 2024]. QaaS marks a pivotal shift in the provisioning of specialized quantum computing resources, following the industry-wide trend of delivering technology through service-oriented models. It parallels established paradigms such as Function-as-a-Service (FaaS) [Chopra *et al.*, 2021] and Platform-as-a-Service (PaaS) [Mei *et al.*, 2015].

In QaaS, quantum serverless architecture lowers the barrier to accessing quantum hardware [Li and Zhao, 2024; Nguyen *et al.*, 2024b], but deploying quantum algorithms is a complex and multifaceted process. Particularly, Noisy Intermediate-Scale Quantum (NISQ) hardware [Preskill, 2018; Li *et al.*, 2024d; Li *et al.*, 2024a] is characterized by limited coherence times, noisy operations, and restricted qubit connectivity. Unlike logical qubits, physical qubits on NISQ devices can typically interact only with their immediate neighbors, making two-qubit gate operations challenging. Consequently, algorithm deployment necessitates a series of transpilation steps, including qubit mapping, circuit optimization, gate decomposition, topology-aware gate insertion, and eventual execution on physical hardware [Li *et al.*, 2019]. We provide the illustrative explanation for circuit deployment in Figure 1. Consider deploying the original circuit (involving 3 qubits) onto the *IBM_Lima* [IBM, 2022a] (the quantum device contains 5

*Co-corresponding authors.

qubits), the right part of Figure 1 displays two types of qubit mapping methods. It is clear that the transpiled circuit obtained using the *Virtual_to_Physical:* $\{0{:}1;\ 1{:}0;\ 2{:}3\}$ strategy will use fewer gates and have a shorter execution time than the *Virtual_to_Physical:* $\{0{:}1;\ 1{:}4;\ 2{:}2\}$ strategy. A detailed explanation of the factors leading to these differences in transpiled circuits is provided in § 3.2.

Deploying quantum algorithms on NISQ devices faces key challenges in qubit mapping and layout optimization, both NP-hard problems with large search spaces [Lao *et al.*, 2021]. To this end, existing methods use heuristic algorithms for approximations, and a representative solution is IBM Qiskit [Aleksandrowicz and others, 2019]. Nevertheless, previous techniques have some limitations. On the one hand, they often provide suboptimal solutions, resulting in additional quantum gate overhead and reduced execution efficiency. On the other hand, existing solutions only provide a single deployment scheme, which may fail to satisfy specific functional requirements (*e.g.,* constraints on fidelity).

In this paper, we propose QCDeploy that can provide a ranked list of effective deployment strategies to optimize quantum serverless circuit deployment. To the best of our knowledge, this is the first solution that considers user requirements of function execution results and provides a list of candidate strategies. Specifically, we model quantum circuits as Directed Acyclic Graph (DAG) representations and utilize graph contrastive learning for vector embedding. Subsequently, a tailored list-aware learning-to-rank architecture is employed to generate a list of candidate strategies (prioritizing better deployment strategies). As a result, QCDeploy not only delivers low-overhead deployment strategies but also offers multiple alternatives to accommodate diverse user-specific constraints and preferences.

In summary, this paper makes three key contributions.

- We carefully examine the challenges of function deployment when using current serverless FaaS technologies for quantum computing. Then, we propose QCDeploy, to provide a list of effective deployment strategies to optimize quantum serverless circuit execution.
- In QCDeploy, we employ the Graph Convolutional Network (GCN) to extract the embedding features of the circuit DAG, and leverage graph contrastive learning to capture structural patterns. The circuit graph embedding vector is then fed to the learning to rank architecture and outputs a list of candidate deployment strategies.
- We implement a prototype of QCDeploy and evaluate it substantially on the IBM Quantum platform with 3 physical devices. The experiments involve 100 groups of function sets based on 45 prevalent quantum circuits, encompassing 15 algorithms each configured with 3 different qubit counts. The results demonstrate that QCDeploy significantly outperforms the IBMQ default deployment scheme in terms of circuit execution overhead and execution success rate.

2 Background and Related Work

Quantum Computing and QaaS. Quantum computing has shown great potential in solving computational problems that are intractable for classical computers [Li *et al.*, 2025b], such as factoring large numbers [Shor, 1999], cryptography [Pirandola and others, 2020], and simulating quantum systems [Zhang *et al.*, 2023b]. Quantum as a Service (QaaS) is a cloud-based paradigm that provides remote access to quantum computing resources via the Internet [Garcia-Alonso *et al.*, 2021], such as IBM Quantum [IBM, 2022a] and Amazon Braket [Gonzalez, 2021]. It allows users to leverage the power of quantum processors for various computational tasks without owning or maintaining the sophisticated and costly quantum hardware themselves. In the Noisy Intermediate-Scale Quantum (NISQ) [Preskill, 2018] era, characterized by the availability of quantum devices with a limited number of qubits that are prone to noise and error, has sparked a surge in quantum computing research and development [Li *et al.*, 2025a]. Effectively utilizing NISQ-era quantum processors depends heavily on the efficient management of quantum resources. Therefore, optimizing the orchestration and choreography of circuit deployment is essential, as it can enhance resource efficiency, improve algorithm performance, and scalability.

Quantum Circuit Deployment. Deploying quantum algorithms onto physical hardware requires bridging the gap between high-level logical circuits and low-level hardware implementations [Preskill, 2018; Li and Zhao, 2024]. This deployment process typically involves compiling the algorithm into a sequence of quantum gates compatible with the target hardware [Smith *et al.*, 2016], optimizing the circuit to reduce noise and resource consumption [Hietala *et al.*, 2023], and mapping logical qubits to physical qubits [Zulehner *et al.*, 2018]. The physical implementation is constrained by the hardware topology, which dictates how qubits can interact [Wille *et al.*, 2019]. To accommodate these constraints, additional operations like SWAP gates may be required [Svore *et al.*, 2018], complicating the deployment process. The above operations (such as mapping and routing) are NP-hard problems [Lao *et al.*, 2021], making efficient and scalable solutions a critical area of research. Current solutions mainly include mathematical optimization and heuristic algorithms. Mathematical methods, such as integer linear programming (ILP) [Tan and Cong, 2020], aim for globally optimal solutions but often struggle with scalability due to the exponential size of the search space. Heuristic approaches, like genetic algorithms and simulated annealing [Zhou *et al.*, 2020], provide approximate solutions that prioritize efficiency over optimality. In light of recent advances in artificial intelligence, we intend to promote a data-driven scheme for generating deployment strategies via deep learning techniques.

3 Preliminaries and Problem Formulation

In this section, we formalize the system model and problem space of quantum serverless circuit deployment optimization.

3.1 System Model

We consider a system represented by the tuple $\mathcal{S}_{\mathcal{M}} = (\mathcal{D}, \mathcal{F})$, where $\mathcal{D}$ denotes the available quantum devices (*i.e.,* quantum computers) and $\mathcal{F}$ represents the set of functions that

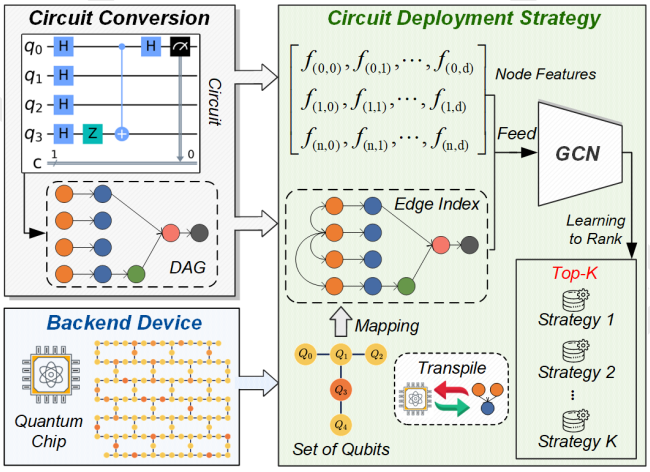

Figure 2: The overall workflow of QCDeploy.

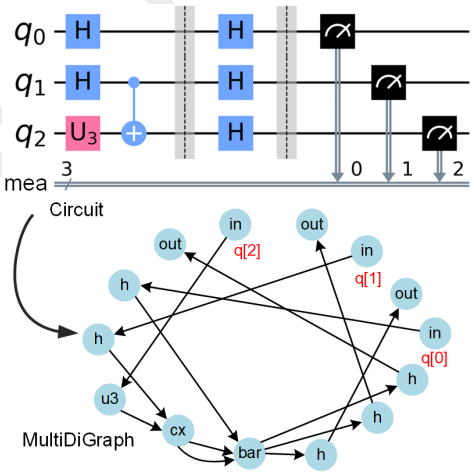

Figure 3: The DAG conversion of the quantum circuit.

need to be executed. Formally, $\mathcal{D} = \{d_1, d_2, ...\}$ and $\mathcal{F} = \{f_1, f_2, f_3, ...\}$. The objective is to execute all functions in $\mathcal{F}$ using the devices in $\mathcal{D}$. Note that for any function $f_i \in \mathcal{F}$ requiring n_c qubits, it can only be executed on a device $d_j \in \mathcal{D}$ that provides at least $n_d \geq n_c$ qubits.

3.2 Optimization Goals

The optimization of quantum function deployment primarily involves two aspects: circuit deployment and candidate strategies. On the one hand, circuit deployment relies on the qubit mapping method to transpile the virtual circuit into a physical one. This process involves decomposing virtual circuit gates into basis gates supported by the target quantum hardware and inserting *SWAP* gates to satisfy the qubit connectivity constraints imposed by the device topology [Li *et al.*, 2019]. Specifically, qubit mapping refers to assigning the n_c qubits of the original circuit to the n_d physical qubits of the selected quantum device, where $n_d \geq n_c$. As shown in Figure 1, different qubit mapping configurations yield distinct transpiled circuits, which vary in terms of gate count and execution overhead. The primary reason for this variability lies in the physical connectivity constraints of the quantum chip, not all two-qubit operations can be directly executed between arbitrary pairs of qubits. In Figure 1, the original circuit includes a two-qubit gate (*i.e.,* the CX gate involved q_0 and q_1). We analyze two different qubit mapping strategies for circuit deployment based on IBMQ_Lima [IBM, 2022a]. For the *Virtual_to_Physical:* {0:1; 1:0; 2:3} strategy, q_0 (corresponding to q_1 in original circuit) and q_1 (corresponding to q_0 in original circuit) in IBMQ_Lima are directly connected so as the CX gate can be directly implemented. For the *Virtual_to_Physical:* {0:1; 1:4; 2:2} strategy, q_4 (corresponding to q_1 in original circuit) and q_1 (corresponding to q_0 in original circuit) in IBMQ_Lima are not directly connected so that the CX gate needs to be implemented with a series of additional *SWAP* gates to exchange the quantum states of two qubits. Therefore, we can see that more gates are introduced in Figure 1 (d) to achieve transpilation in the physical device.

On the other hand, considering that quantum computers are affected by quantum noise, when a given deployment scheme cannot meet user requirements (*e.g.,* realize greater than 0.9 fidelity), we tend to provide a list of candidate deployment strategies to ensure the successful execution of user functions as much as possible.

4 Design Details of QCDeploy

4.1 Architecture and Overview

In Figure 2, we depict a high-level architecture of QCDeploy, including key components of quantum circuit representation and learning to rank module. Among them, the functions (*i.e.,* quantum circuits) arrive sequentially over time. Then, the function circuit will be converted into a Directed Acyclic Graph (DAG) to feed the Graph Convolutional Network (GCN) [Kipf and Welling, 2017] for embedding vector extraction. To enhance the representation, a tailored graph contrastive learning module is used to train the GCN parameters. Subsequently, the learning to rank module is used to generate a list of candidate deployment strategies.

4.2 Quantum Circuit Representation

A quantum circuit can be converted into a Directed Acyclic Graph (DAG), as illustrated in Figure 3. In this representation, each quantum gate corresponds to a node in the graph, and the qubits serve as edges connecting these nodes. For instance, a two-qubit gate results in a node with both in-degree and out-degree equal to 2, such as the *CX* gate in the BV_3 example shown in Figure 3. Note that the resulting DAG from a quantum circuit may be a MultiDiGraph, which allows multiple (parallel) edges between the same pair of nodes. Formally, a quantum circuit DAG (*i.e.,* graph) can be defined as $\mathcal{G} = \{\mathcal{V}, A\}$, where $\mathcal{V}$ is the gate set consisting of n nodes $\{v_1, \cdots, v_n\}$, and $A \in \mathbb{R}^{n \times n}$ is a symmetric matrix used to characterize the adjacency relationship for nodes. Specifically, $a_{ij} = 1$ indicates an edge between nodes v_i and v_j, and 0 otherwise. To record the edge number of nodes, the node degree matrix is defined as $D = \mathbf{diag}(d_1, \cdots, d_n)$ with

Algorithm 1 Contrastive Learning for Circuit Graph

Input: The initialized GCN-based graph embedding model $\mathcal{M}$, the quantum circuit graph dataset $\mathcal{S}$
Output: The well-trained model $\mathcal{M}$

1: # Generate graph view with random perturbations
2: **procedure** GENE_GRAPH_VIEWS(g)
3: $g' \leftarrow$ copy(g)
4: $edges \leftarrow$ list(g'.edges(keys=true)) ▷ Multiple edges
5: Initialize the perturbation probability $prob$
6: **for** $edge \in edges$ **do**
7: **if** rand().item() $< prob$ **then**
8: g'.remove_edge($edge$)
9: **end if**
10: **end for**
11: **return** $g'.feature, g'.edges$
12: **end procedure**
13: # The training pipeline
14: Initialize the optimizer
15: Initialize the self-supervised contrastive loss $criterion$
16: **for** $g \in \mathcal{S}$ **do**
17: optimizer.zero_grad()
18: $feature_1, edge_1 \leftarrow$ Gene_graph_views(g)
19: $feature_2, edge_2 \leftarrow$ Gene_graph_views(g)
20: $z_i \leftarrow \mathcal{M}(feature_1, edge_1)$
21: $z_j \leftarrow \mathcal{M}(feature_2, edge_2)$
22: $loss \leftarrow criterion(z_i, z_j)$
23: $loss.backward()$
24: optimizer.step()
25: **end for**
26: **return** The well-trained model $\mathcal{M}$

$d_i = \sum_{j=1}^{n} a_{ij}$.

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}, \; D = \begin{pmatrix} d_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & d_n \end{pmatrix} \tag{1}$$

In addition, we generate one-hot vectors for each node in the graph and stack them to form the feature matrix. Specifically, the nodes considered include 10 types, *i.e.,* {'barrier', 'in', 'out', 'cx', 'cz', 'h', 'rz', 'u1', 'u3', 'x'}. Among them, 'barrier', 'in', and 'out' belong to circuit control nodes, corresponding to circuit logic partitioning, quantum state initialization, and quantum state measurement, respectively. And {'cx', 'cz', 'h', 'rz', 'u1', 'u3', 'x'} correspond to various single-qubit gates and two-qubit gates. Therefore, each node is finally mapped into a 10-dimensional one-hot vector, and the feature matrix of the circuit graph is represented as an $n \times 10$ matrix.

4.3 Graph Vector Embedding

We employ a Graph Convolutional Network (GCN) [Kipf and Welling, 2017] model to extract the embedding representations of the circuit DAG. Specifically, GCN transforms each node vector of size f (*i.e.,* the dimension of node input) into a vector of size c (*i.e.,* the embedding dimension) through multiple graph convolutional layers. At the final

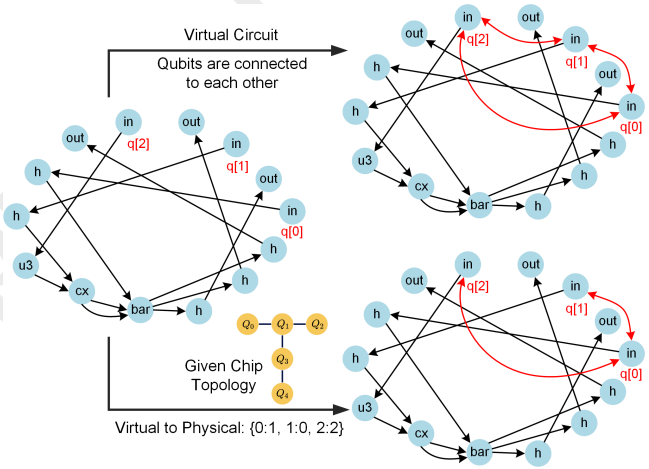

Figure 4: Mapping node connections based on chip and strategy.

layer, the embedding vectors of all nodes are averaged to obtain the embedding representation of the DAG. Formally, let $X^{(l)} = \{x_1^{(l)}, \cdots, x_n^{(l)}\} \in \mathbb{R}^{n \times c}$ ($l \in \{1, \cdots, L\}$) denotes the node feature matrix after layer l and the row feature vector of node i represented as $x_i^{(l)} = \{h_0, \cdots, h_{c-1}\} \in \mathbb{R}^c$ (especially, $x_i^{(0)} \in \mathbb{R}^f$). These vectors are updated at every layer, so as to construct a hierarchical profile with higher-level vectors representing broader and more abstract properties.

At each GCN layer, the vector of each node is first transformed by a learned matrix $W^{(l)} \in \mathbb{R}^{f \times c}$ as $\tilde{x}_i^{(l)} = x_i^{(l-1)} W^{(l)}$. Then, each node's representation vector is averaged with that of its direct neighbors, which allows this node to include its neighbor qubit information, effectively capturing the topological structure of the quantum circuit.

$$x_i^{(l)} \leftarrow \sum_{j=1}^{n} \frac{a_{ij}}{\sqrt{d_i d_j}} \tilde{x}_j^{(l)}, \quad \forall i \in \{1, 2, \ldots, n\} \tag{2}$$

Furthermore, a non-linear function σ is applied at the end to complete updating the node representation vectors for the current layer. More compactly, we can express the above update in matrix form as $X^{(l)} \leftarrow \sigma(\bar{A} X^{(l-1)} W^{(l)})$, where σ usually employs ReLU activation function (*i.e.,* $\sigma(x) = \max(0, x)$), and $\bar{A} = D^{-1/2}(A + I)D^{-1/2}$ is the normalized adjacency matrix (I refers the identity matrix).

Next, we will adopt self-supervised contrastive learning [You *et al.*, 2020] to train the GCN embedding model parameters. The contrastive learning workflow is summarized in Algorithm 1. Given quantum circuit graph dataset $\mathcal{S}$, take out a specific circuit g, and call the $Gene\_graph\_views$ function to generate two views of g after random perturbations. Particularly, perturbation refers to randomly removing edge connections with a certain probability (lines 1∼12 in Algorithm 1). Subsequently, a tailored self-supervised contrastive learning loss is defined as follows:

$$\mathcal{L} = -\log \frac{\exp(\tilde{z}_i \cdot \tilde{z}_j / \tau)}{\sum_{k=1}^{n} \exp(\tilde{z}_i \cdot \tilde{z}_k / \tau) + \epsilon} \tag{3}$$

where $\tilde{z}_i$ and $\tilde{z}_j$ are the normalized results of the embedding

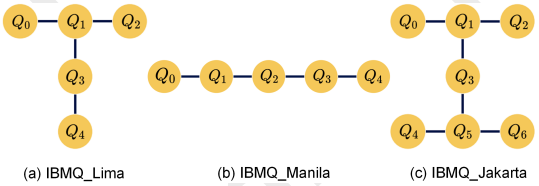

Figure 5: Three types of quantum chip topologies.

vectors (z_i and z_j at lines 20∼21 in Algorithm 1). In Eq. (3), the numerator calculates the positive-pair score, while the denominator represents the negative-pair score. Moreover, $\cdot$ represents the dot product, τ is the temperature hyperparameter [He *et al.*, 2020], and ϵ is a small value to prevent numerical instability. The above contrastive learning loss is used to calculate the embeddings from both graph views. Then, the loss is backpropagated through the network, and the optimizer updates the model's parameters. Finally, we can obtain a well-trained model $\mathcal{M}$, which enables effectively extracting embedding representations of quantum circuit graphs.

4.4 Candidate Deployment Strategy Generation

Given a quantum circuit graph, its virtual circuit (before deployment) assumes that all qubits are interconnected, as shown in the upper part of Figure 4. However, on a physical quantum chip, there is a certain connection relationship between qubits (*i.e.,* chip topology). In addition, the deployment strategy will directly affect the connection relationship between qubits in the physical quantum circuit. A specific example is shown in the lower part of Figure 4, the qubit mapping from virtual to physical reference {0:1, 1:0, 2:2}. Thus, $q[0]$ and $q[1]$ are connected to each other in virtual circuits (due to $q[1]$ is connected with $q[0]$ in physical circuit); $q[0]$ and $q[2]$ are connected to each other in virtual circuits (due to $q[1]$ is connected with $q[2]$ in physical circuit).

For a specific deployment strategy mapping into a circuit graph, we can extract its graph vector embedding (in § 4.3), and the top node representations $X^{(L)}$ (after L layers) are cascaded with the average function, and the embedding vector representation of the circuit DAG can be denoted as follows.

$$\mathbf{E} = \mathbf{average}(x^{(L)}) = \frac{e^{h_k}}{\sum_{k=0}^{c-1} e^{h_k}} \tag{4}$$

To generate the candidate strategies, we use RankFormer architecture [Buyl *et al.*, 2023] in QCDeploy, a list-aware Transformer designed for Learning-to-Rank (LTR) tasks. Let $\mathbf{E}_i$ represent the embedding vector for the i-th deployment strategy. The embedding vectors $\mathbf{E}_1, \mathbf{E}_2, \ldots, \mathbf{E}_n$ are fed into a Transformer encoder. The Transformer processes these vectors to capture interdependencies among items within the list.

$$\mathbf{H} = \text{Transformer}(\mathbf{E}_1, \mathbf{E}_2, \ldots, \mathbf{E}_n) \tag{5}$$

Each item's features $\mathbf{h}_i$ are passed through a scoring head to produce a raw score s_i. This scoring head typically consists of a linear layer followed by an activation function.

$$s_i = \text{ScoreHead}(\mathbf{h}_i) \tag{6}$$

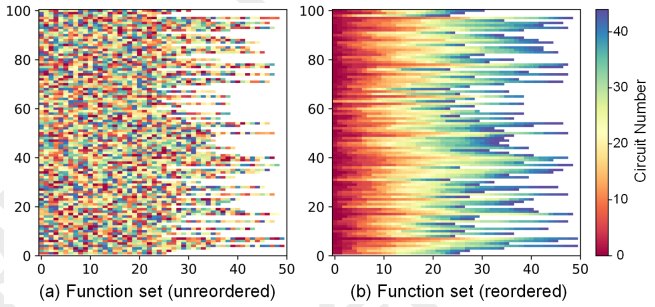

Figure 6: Visualization of 100 testing groups of function sets.

In addition to individual item scores, QCDeploy also models the overall quality of the list using the $[CLS]$ token's output from the Transformer, denoted as $\mathbf{h}_{\text{CLS}}$.

$$q = \text{ListAssessmentHead}(\mathbf{h}_{\text{CLS}}) \tag{7}$$

The final score for each item is a combination of its raw score s_i and the list's quality score q. This combination ensures that per-item scores are adjusted based on the overall list quality.

$$\mathbf{s}' = \mathbf{s} + \alpha q \tag{8}$$

where α is a learnable parameter that controls the influence of the listwide quality on individual scores. The items are then ranked based on their final scores $\mathbf{s}'$. The ranking can be formally represented as sorting the indices i based on $\mathbf{s}'_i$:

$$\text{Ranking} = \text{argsort}(\mathbf{s}') \tag{9}$$

Overall, QCDeploy will produce the candidate strategy list based on embedding vectors of deployment strategies.

5 Evaluation

In this section, we comprehensively evaluate the quantum serverless circuit deployment effect by QCDeploy, with code available in the online repository[1].

5.1 Experiment Setup

Testbeds. We use OpenWhisk [Apache, 2021] to implement the quantum serverless framework, given that OpenWhisk is an open-source, distributed serverless computing platform that provides a flexible and scalable environment for running code snippets or functions without the need to manage infrastructure. Specifically, we deploy the IBMQ as the backend and employ IBM Qiskit [Aleksandrowicz and others, 2019] to conduct quantum experiments. Three quantum machines *['IBMQ_Lima', 'IBMQ_Manila', 'IBMQ_Jakarta']* are utilized, and these machines consist of three distinct topological structures of quantum chips, as shown in Figure 5. For the quantum circuit deployment baseline, we use the default scheme of IBM [IBM, 2022a; IBM, 2022b] quantum computing platform as the competing algorithm, which is consistent with existing work [Wang *et al.*, 2022].

Basic Quantum Circuits. For serverless function orchestration experiments, we generate a series of basic quantum

[1]Repository https://github.com/Secbrain/QCDeploy.

Candidate Qubit	Number of strategies = 10			Number of strategies = 20			Number of strategies = 30		
	$qubit = 3$	$qubit = 4$	$qubit = 5$	$qubit = 3$	$qubit = 4$	$qubit = 5$	$qubit = 3$	$qubit = 4$	$qubit = 5$
IBMQ_Lima	87.47±0.09	83.41±0.12	92.03±0.08	73.30±0.11	73.53±0.18	78.15±0.15	68.67±0.09	69.12±0.21	75.83±0.15
IBMQ_Manila	89.22±0.08	88.23±0.09	86.32±0.13	72.11±0.15	74.29±0.14	75.01±0.19	70.29±0.12	72.03±0.16	71.62±0.21
IBMQ_Jakarta	82.42±0.15	89.12±0.10	84.47±0.18	73.04±0.15	75.90±0.16	74.45±0.18	66.96±0.15	71.22±0.17	68.51±0.22
Candidate Qubit	**Number of strategies = 40**			**Number of strategies = 50**			**Number of strategies = 60**		
	$qubit = 3$	$qubit = 4$	$qubit = 5$	$qubit = 3$	$qubit = 4$	$qubit = 5$	$qubit = 3$	$qubit = 4$	$qubit = 5$
IBMQ_Lima	65.66±0.12	69.03±0.24	72.65±0.16	60.16±0.18	66.49±0.27	72.78±0.18	60.03±0.21	65.80±0.28	71.51±0.19
IBMQ_Manila	67.95±0.10	71.59±0.20	69.31±0.22	63.68±0.12	69.33±0.22	68.43±0.24	61.48±0.13	67.64±0.23	65.17±0.28
IBMQ_Jakarta	66.34±0.09	68.40±0.18	64.83±0.25	65.96±0.09	66.54±0.20	59.68±0.26	65.36±0.10	66.54±0.19	60.75±0.24

Table 1: The averaged NDCG results for different numbers of candidate strategies, under various qubit settings and different devices.

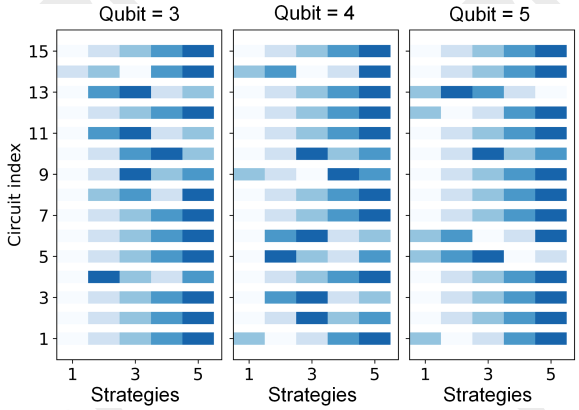

Figure 7: Visualization of generated strategies by QCDeploy for IBMQ_Manila. Lighter colors represent better deployment schemes.

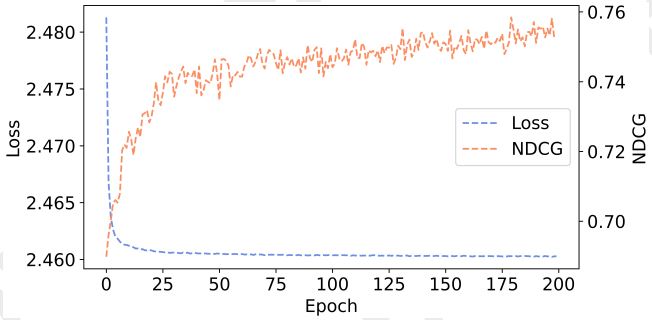

Figure 8: The loss and NDCG curves during model training.

circuits as the function to be executed. Specifically, 15 classical quantum algorithm circuits [Zhang *et al.*, 2023a; Li *et al.*, 2019; Li *et al.*, 2020b; Li *et al.*, 2020a] are used, including *['BV', 'Clifford', 'Grover', 'Ising', 'QFT', 'QKNN', 'QNN', 'QPE', 'QSVM', 'QuGAN', 'RB', 'Shor', 'Simon', 'VQC', 'XEB']*, that cover commonly used quantum algorithm circuits. For each algorithm, we set the used qubit ranging from $[3, 5]$, resulting in a total of 45 original circuits. These 45 (*i.e.,* 15 algorithms $\times$ 3 types of qubit settings) original circuits will be transpiled onto 3 quantum devices with three types of chip topologies.

Function Groups. For function sets, we randomly generate 200 groups from the original circuits, and each group comprises a random number of circuits ranging from $[20, 50]$. These 200 groups are divided into training and testing sets according to 1:1. For 100 testing groups of function sets, we visualize them in native-generated order in Figure 6 (a), and Figure 6 (b) corresponding to reordered by circuit index. Note that the groups with native-generated order in Figure 6 (a) are the test data for subsequent experiments.

5.2 Generated Strategy Analysis

We analyze the generated strategies based on the Normalized Discounted Cumulative Gain (NDCG) metric [Buyl *et al.*, 2023], which is widely used to evaluate ranking quality.

Different Number of Candidate Strategies. Under three quantum devices and three types of qubit settings, we consider different numbers of candidate strategies (*i.e.,* $[10, 20, 30, 40, 50, 60]$), calculate the NDCG results of the generation strategies by QCDeploy and summarize them in Table 1. It is clear that when the number of candidates is smaller, QCDeploy realizes better results, *e.g.,* under IBMQ_Lima and $qubit = 5$, the NDCG is 92.03% for 10 candidate strategies. Even when considering 60 candidate strategies, QCDeploy can still achieve more than 60% NDCG results. This demonstrates the effectiveness of QCDeploy because we can usually narrow the candidate deployment strategies to less than a dozen by combining the topological connections between circuits and quantum chips.

Deployment Strategy Visualization. Furthermore, we visualize the generated strategies to provide more in-depth insights. Among them, the lighter the color of the block, the better the deployment strategy. Figure 7 shows 15 circuits with three types of qubit settings on IBMQ_Manila (results of other devices can be found in the online repository), QCDeploy almost generates excellent deployment strategy lists, in which colors are arranged from light to dark. In addition, even if the first one is a suboptimal deployment strategy in some cases, the candidate strategy list generated by QCDeploy can support the optimal deployment strategy within a limited number of times.

Convergence Process during Training. We record the training loss and NDCG logs and plot them in Figure 8. It can be seen that the model converges quickly. At about $Epoch = 25$, the loss value drops to a lower level, and then NDCG gradually approaches the highest value at about $Epoch = 75$.

5.3 Deployment Effectiveness Evaluation

Next, we evaluate the practical deployment effect with IBMQ physical machines as the backend.

Circuit Execution Time Analysis. For three quantum machines, we randomly select 10 sets of functions and ana-

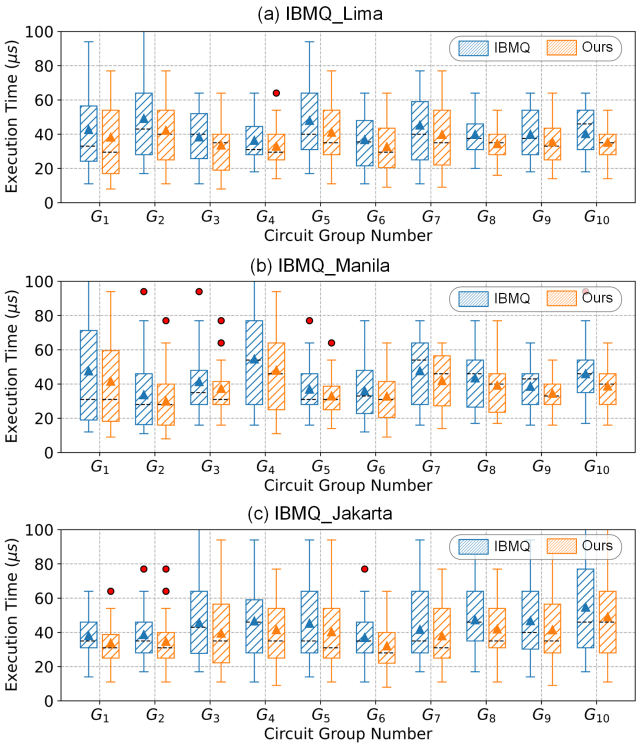

Figure 9: The circuit execution time analysis.

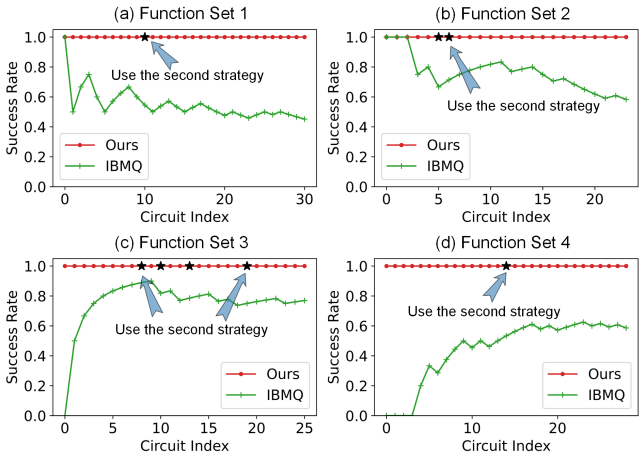

Figure 10: The execution success rate of function sets.

lyze the execution time of each quantum circuit. As shown in Figure 9, our solutions all outperform the default deployment strategy of IBMQ. For example, for G_{10} under IBMQ_Manila machine, the maximum execution time is reduced from $\sim 78\mu s$ to $\sim 64\mu s$, which realizes 17.95% overhead reduction. This demonstrates that vector embedding design and graph contrastive learning in QCDeploy can effectively extract quantum circuit representations, thereby enabling remarkable deployment strategy generation.

Function Execution Success Rate. Moreover, we consider a conditionally constrained setting in the real world where users have requirements on the execution overhead and fidelity of functions. Specifically, we set the execution fidelity greater than 0.9 as the success condition. We plot four groups of function sets from the testset in Figure 10, based on the IBMQ_Lima machine. We can observe that, at the end of each function set, the success rate of IBMQ is between 40% to 80%. In contrast, all our solutions achieve a 100% success rate. The black star mark indicates that this function succeeded after executing two candidate strategies. In other words, in the candidate list generated by QCDeploy, the first one cannot meet the fidelity condition, while the second strategy can meet the requirement. This highlights that the Learning-to-Rank design in QCDeploy is able to prioritize and recommend the optimal deployment strategy, thereby supporting low execution overhead and high fidelity within a limited number of times to meet user needs.

6 Discussion

Practicability of Quantum Serverless FaaS. In the realm of quantum computing, the efficient orchestration of quantum resources is paramount to enhancing the performance and scalability of quantum algorithms. Motivated by the current advancements in Function-as-a-Service (FaaS) and Quantum-as-a-Service (QaaS) [Nguyen *et al.*, 2024b], our research aligns with the ongoing technological convergence of these paradigms and explores their architectural integration. Furthermore, our proposed approach is also readily applicable to major quantum cloud service providers, such as IBM, Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform, and so on [Nguyen *et al.*, 2024a]. This compatibility underscores the practical extensibility of our method in real-world quantum computing infrastructures.

Limitations and Future Work. For different users, there may be some specific patterns in circuit architectures (or scale), so considering fine-grained customer-customized orchestration solutions may be an interesting and promising direction. As part of future work, we will explore more advanced neural network architectures that can be integrated into the QCDeploy framework to enhance its decision-making capabilities. Furthermore, quantum noise [Patel and others, 2020; Li *et al.*, 2024b] can also be considered and used as input for function orchestration action to build higher-fidelity circuits on physical machines. Finally, which components can be parallelized to run more efficiently will be considered in subsequent work.

7 Conclusion

In this paper, we present QCDeploy, to provide a list of effective deployment strategies to optimize quantum serverless circuit deployment. Given the properties of quantum circuits, QCDeploy designs the tailor-made circuit representation scheme, including DAG conversion and GCN embedding vector feature extraction. Moreover, QCDeploy leverages graph contrastive learning for mining node connection patterns and learning-to-rank architecture to generate a list of candidate strategies. Based on 3 IBM quantum physical devices, extensive experiments with 45 quantum algorithm circuits demonstrate that QCDeploy can effectively reduce circuit execution time and improve execution success rate.

References

[Aleksandrowicz and others, 2019] Gadi Aleksandrowicz et al. Qiskit: An open-source framework for quantum computing. *Accessed on: Mar*, 16, 2019.

[Apache, 2021] Apache. Open source serverless cloud platform. https://openwhisk.apache.org/, 2021.

[Buyl *et al.*, 2023] Maarten Buyl, Paul Missault, and Pierre-Antoine Sondag. Rankformer: Listwise learning-to-rank using listwide labels. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3762–3773, 2023.

[Chopra *et al.*, 2021] Amit K Chopra, Munindar P Singh, et al. Deserv: Decentralized serverless computing. In *2021 IEEE International Conference on Web Services (ICWS)*, pages 51–60. IEEE, 2021.

[Deng *et al.*, 2022] Jinfeng Deng, Hang Dong, Chuanyu Zhang, Yaozu Wu, Jiale Yuan, Xuhao Zhu, Feitong Jin, Hekang Li, Zhen Wang, Han Cai, et al. Observing the quantum topology of light. *Science*, 378(6623):966–971, 2022.

[Garcia-Alonso *et al.*, 2021] Jose Garcia-Alonso, Javier Rojo, David Valencia, Enrique Moguel, Javier Berrocal, and Juan Manuel Murillo. Quantum software as a service through a quantum api gateway. *IEEE Internet Computing*, 26(1):34–41, 2021.

[Gonzalez, 2021] Constantin Gonzalez. Cloud based qc with amazon braket. *Digitale Welt*, 5(2):14–17, 2021.

[He *et al.*, 2020] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[Hietala *et al.*, 2023] Kesha Hietala, Robert Rand, Liyi Li, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. A verified optimizer for quantum circuits. *ACM Transactions on Programming Languages and Systems*, 45(3):1–35, 2023.

[IBM, 2022a] IBM. IBMQ Quantum. https://quantum-computing.ibm.com/, 2022.

[IBM, 2022b] Qiskit IBM. IBM Qiskit. https://qiskit.org/, 2022.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR (Poster)*. OpenReview.net, 2017.

[Lao *et al.*, 2021] Lingling Lao, Hans Van Someren, Imran Ashraf, and Carmen G Almudever. Timing and resource-aware mapping of quantum circuits to superconducting processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(2):359–371, 2021.

[Li and Zhao, 2024] Tingting Li and Ziming Zhao. Moirai: Optimizing quantum serverless function orchestration via device allocation and circuit deployment. In *2024 IEEE International Conference on Web Services (ICWS)*, pages 707–717. IEEE, 2024.

[Li *et al.*, 2019] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[Li *et al.*, 2020a] Gushu Li, Yufei Ding, and Yuan Xie. Eliminating redundant computation in noisy quantum computing simulation. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[Li *et al.*, 2020b] Gushu Li, Yufei Ding, and Yuan Xie. Towards efficient superconducting quantum processor architecture design. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1031–1045, 2020.

[Li *et al.*, 2023] Junde Li, Mahabubul Alam, and Swaroop Ghosh. Large-scale quantum approximate optimization via divide-and-conquer. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2023.

[Li *et al.*, 2024a] Tingting Li, Liqiang Lu, Ziming Zhao, Ziqi Tan, Siwei Tan, and Jianwei Yin. Qust: Optimizing quantum neural network against spatial and temporal noise biases. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[Li *et al.*, 2024b] Tingting Li, Ziming Zhao, and Jianwei Yin. Minerva: Enhancing quantum network performance for high-fidelity multimedia transmission. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 3704–3712, 2024.

[Li *et al.*, 2024c] Tingting Li, Ziming Zhao, and Jianwei Yin. Qlsel: Demonstrating efficient high-fidelity link selection for quantum networks in the wild. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 1766–1768, 2024.

[Li *et al.*, 2024d] Tingting Li, Ziming Zhao, and Jianwei Yin. Task-driven quantum device fingerprint identification via modeling QNN outcome shift induced by quantum noise. In *WWW (Companion Volume)*, pages 557–560. ACM, 2024.

[Li *et al.*, 2025a] Tingting Li, Ziming Zhao, Liqiang Lu, Siwei Tan, and Jianwei Yin. Empowering quantum error traceability with moe for automatic calibration. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pages 1–7. IEEE, 2025.

[Li *et al.*, 2025b] Tingting Li, Ziming Zhao, Liqiang Lu, and Jianwei Yin. Quframe: A novel encoding ensemble framework for quantum neural networks. In *2025 International Conference on Quantum Communications, Networking, and Computing (QCNC)*, pages 583–590. IEEE, 2025.

[Mei *et al.*, 2015] Lijun Mei, Hao Chen, Shaochun Li, Qicheng Li, Guangtai Liang, and Jeaha Yang. A service-based framework for mobile social messaging in paas systems. In *2015 IEEE International Conference on Web Services*, pages 751–754. IEEE, 2015.

[Nguyen *et al.*, 2024a] Hoa T Nguyen, Prabhakar Krishnan, Dilip Krishnaswamy, Muhammad Usman, and Rajkumar Buyya. Quantum cloud computing: a review, open problems, and future directions. *arXiv preprint arXiv:2404.11420*, 2024.

[Nguyen *et al.*, 2024b] Hoa T Nguyen, Muhammad Usman, and Rajkumar Buyya. Qfaas: A serverless function-as-a-service framework for quantum computing. *Future Generation Computer Systems*, 154:281–300, 2024.

[Patel and others, 2020] Tirthak Patel et al. UREQA: leveraging operation-aware error rates for effective quantum circuit mapping on nisq-era quantum computers. In *USENIX ATC*, 2020.

[Pirandola and others, 2020] Stefano Pirandola et al. Advances in quantum cryptography. *Advances in optics and photonics*, 12(4):1012–1236, 2020.

[Preskill, 2018] John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2018.

[Ren *et al.*, 2022] Wenhui Ren, Weikang Li, Shibo Xu, Ke Wang, Wenjie Jiang, Feitong Jin, Xuhao Zhu, Jiachen Chen, Zixuan Song, Pengfei Zhang, et al. Experimental quantum adversarial learning with programmable superconducting qubits. *Nature Computational Science*, 2022.

[Saravanan and Saeed, 2023] Vedika Saravanan and Samah M. Saeed. Data-driven reliability models of quantum circuit: From traditional ML to graph neural network. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2023.

[Shor, 1999] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[Smith *et al.*, 2016] Robert S Smith, Michael J Curtis, and William J Zeng. A practical quantum instruction set architecture. *arXiv preprint arXiv:1608.03355*, 2016.

[Svore *et al.*, 2018] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q# enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the real world domain specific languages workshop 2018*, pages 1–10, 2018.

[Tan and Cong, 2020] Bochen Tan and Jason Cong. Optimal layout synthesis for quantum computing. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[Tan *et al.*, 2023] Siwei Tan, Mingqian Yu, Andre Python, Yongheng Shang, Tingting Li, Liqiang Lu, and Jianwei Yin. Hyqsat: A hybrid approach for 3-sat problems by integrating quantum annealer with cdcl. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 731–744. IEEE, 2023.

[Wang *et al.*, 2022] Hanrui Wang, Jiaqi Gu, Yongshan Ding, Zirui Li, Frederic T. Chong, David Z. Pan, and Song Han. QuantumNAT: quantum noise-aware training with noise injection, quantization and normalization. In *59th ACM/IEEE Design Automation Conference (DAC)*, 2022.

[Wille *et al.*, 2019] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC)*, 2019.

[You *et al.*, 2020] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33:5812–5823, 2020.

[Zhang *et al.*, 2023a] Hezi Zhang, Anbang Wu, Yuke Wang, Gushu Li, Hassan Shapourian, Alireza Shabani, and Yufei Ding. Oneq: A compilation framework for photonic one-way quantum computation. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

[Zhang *et al.*, 2023b] Pengfei Zhang, Hang Dong, Yu Gao, Liangtian Zhao, Jie Hao, Jean-Yves Desaules, Qiujiang Guo, Jiachen Chen, Jinfeng Deng, Bobo Liu, et al. Many-body hilbert space scarring on a superconducting processor. *Nature Physics*, 19(1):120–125, 2023.

[Zhao *et al.*, 2025] Ziming Zhao, Tingting Li, and Zhaoxuan Li. Qufm: Towards efficient quantum link fidelity measurements in quantum networks. In *2025 International Conference on Quantum Communications, Networking, and Computing (QCNC)*, pages 516–520. IEEE, 2025.

[Zhou *et al.*, 2020] Xiangzhen Zhou, Sanjiang Li, and Yuan Feng. Quantum circuit transformation based on simulated annealing and heuristic search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12):4683–4694, 2020.

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