JustQ: Automated Deployment of Fair and Accurate Quantum Neural Networks

Ruhan Wang[1], Fahiz Baba-Yara[2], Fan Chen[1]

[1]*Luddy School of Informatics, Computing, and Engineering*, [2]*Kelley School of Business*

Indiana University, Bloomington, IN, USA

E-mail: {ruhwang, fababa, fc7}@iu.edu

Abstract—Despite the success of Quantum Neural Networks (QNNs) in decision-making systems, their fairness remains unexplored, as the focus primarily lies on accuracy. This work conducts a design space exploration, unveiling QNN unfairness, and highlighting the significant influence of QNN deployment and quantum noise on accuracy and fairness. To effectively navigate the vast QNN deployment design space, we propose JustQ, a framework for deploying fair and accurate QNNs on NISQ computers. It includes a complete NISQ error model, reinforcement learning-based deployment, and a flexible optimization objective incorporating both fairness and accuracy. Experimental results show JustQ outperforms previous methods, achieving superior accuracy and fairness. This work pioneers fair QNN design on NISQ computers, paving the way for future investigations.

Index Terms—Quantum neural networks, fairness, accuracy, noisy intermediate-scale quantum, reinforcement learning

I. INTRODUCTION

Motivation. Quantum Neural Networks (QNNs) [1]–[5] have become powerful tools for automated decision-making, spanning finance [6], healthcare [7], and drug discovery [8]. While classical machine learning models are known to exhibit susceptibility to social biases [9], current works on QNNs predominantly emphasize accuracy [1]–[4] with limited discussion on fairness [5]. We summarize related works in Table I and identify three challenges. *First*, none of the prior work performed a holistic evaluation of the accuracy and fairness of QNNs. *Second*, existing work lacks a comprehensive method for modeling and measuring quantum noises in Noisy Intermediate-Scale Quantum (NISQ) computers [10], despite their significant impact on accuracy and fairness. *Third*, prior efforts solely focused on noise-aware training [2]–[4]. However, our preliminary studies show that accuracy gains from training can be compromised by imprudent and often unknown synthesis settings during the deployment phase.

Contributions. This work introduces a novel framework, JustQ, which addresses fairness and accuracy in QNNs simultaneously. Our contributions are as follows:

- We explored the design space using IBM quantum computers and found that: (1) QNN deployment has a greater impact on accuracy and fairness than training, and (2) QNN accuracy and fairness are closely linked during deployment, highlighting the need for automated exploration methods.
- We present the JustQ framework with three components: (1) an NISQ error model and measurement method; (2) a reinforcement learning-based approach for automated, fair, and accurate QNN deployment; and (3) a flexible optimization objective combining fairness and accuracy in a customizable reward function for diverse application goals.

TABLE I
A SUMMARY OF RELATED WORKS.

Scheme	Metric		Error Model		Design Phase	
	Acc.	Fairness	Real_Device	Meas_Method	Train.	Deploy.
C14 [1]	✔	✗	✗	✗	✔	✗
QMLP [2]			✔	✗		
DATE22 [3]			✔	✗		
DAC22 [4]			✔	✗		
CAV22 [5]	✗	✔	✗	✗		
JustQ	✔	✔	✔	✔	✗	✔

- We evaluate JustQ on various QNNs and show that JustQ surpasses prior deployment methods in both accuracy and fairness. Furthermore, JustQ's adaptable optimization objective empowers users to tailor it to specific application needs.

Limitations and Scope. QNNs have demonstrated advantages over their classical counterparts in theory [11], [12] and small-scale practical implementations [13], [14]. In this work, we initiate research on fair and accurate QNNs using IBM quantum computers, paving the way for future investigations in this domain. Machine learning fairness is an evolving research area lacking a widely agreed-upon definition. We adopt the concept of *individual* fairness [9], which advocates treating similar individuals equitably for similar outcomes. The limitation on the implementation scale (up to 27 qubits) is due to current NISQ technology capabilities. As technology continues to advance [10], the implementation scale and performance enhancement of JustQ will also improve.

II. PRELIMINARY

A. NISQ Hardware

NISQ noises [15]–[17] lead to *readout*, *gate*, and *crosstalk* errors. *Readout* errors refer to incorrect qubit measurement, e.g., reading $|1\rangle$ while the qubit is in the $|0\rangle$ state and vice versa. They are modeled as a 2×2 probability matrix and mitigated with readout-error correction [15]. *Gate* errors are dominated by 2-qubit gates and modeled as depolarizing noise [16]: $\text{Err}(\rho) = (1\text{-}p)\rho + p\frac{I}{2^n}$, where p is the system error rate depending on NISQ computers and the circuit. $\text{Err}(\rho)$, ρ, and n refer to the noise channel, the noiseless density matrix, and the qubit count, respectively. *Crosstalk* errors result from non-ideal qubit interactions in systems with over 10 qubits, causing deviations from the ideal behavior of quantum gates and circuits. The occurrence and impact of *crosstalk* are influenced by the total number and spatial arrangement of 2-qubit gates within the circuit [17].

Limitations in Prior Models. Prior noise-aware QNNs [2]–[4] focused on *gate* errors but ignored *crosstalk* errors. Recent
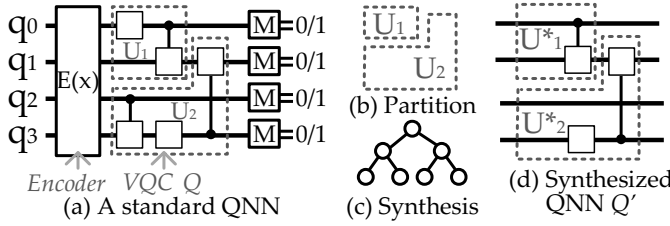
Fig. 1. A standard QNN and its approximate synthesis.

(a) A standard QNN   (b) Partition   (c) Synthesis   (d) Synthesized QNN Q′

Fig. 2. (a) Output distance and (b) accuracy of synthesized QNNs on `IBM_Almaden` (HPW: hours-per-week, Edu: education; CapG: capital-gain).

work [17] shows that *crosstalk* can cause a 20% increase in phase-flip error and a 33% decrease in gate coherence in a small 9-qubit circuit on the `IBM_Melbourne` computer. Moreover, none of the previous work [2]–[4] provides details on error estimation and measurement, which are time-consuming and can introduce significant latency overhead.

B. Quantum Neural Networks

QNNs [1]–[4] are a sequence of *quantum gates* that operate on a set *qubits*. As Figure 1(a) shows, a QNN comprises a data encoder, $E(x)$, embeding a classical input $\mathbf{x}$ into a quantum state $|\mathbf{x}\rangle$, a Variational Quantum Circuit (VQC), $\mathbf{Q}$, generating the output state, and a measurement layer, M, mapping the output quantum state to a classical vector.

QNN Fairness. We consider *individual* fairness [5] for equitable treatment of similar inputs, ensuring unbiased outcomes. A QNN is *fair* if and only if no (ϵ, δ)-*bias* pairs exist in the dataset $\mathcal{T}$. Such a pair is characterized by the input state distance between $E(x_i)$ and $E(x_j)$ being within a threshold ϵ, while the difference between their output $\mathbf{Q}(E(x_i))$ and $\mathbf{Q}(E(x_j))$ exceeds a threshold δ, formulated as:

$$[\mathbf{D}(E(x_i), E(x_j)) \le \epsilon] \wedge [\mathbf{d}(\mathbf{Q}(E(x_i)), \mathbf{Q}(E(x_j))) \ge \delta] \quad (1)$$

where $1 \ge \epsilon, \delta > 0$ and $\mathbf{D}(\cdot)$, $\mathbf{d}(\cdot)$ are distance metrics applied to the input and output quantum spaces. [5] has proved that if $\mathbf{D}(\cdot)$ and $\mathbf{d}(\cdot)$ are respectively specified as trace distance and measurement outcome distribution, then the following holds

$$\underbrace{\mathbf{d}(\mathbf{Q}(E(x_i)), \mathbf{Q}(E(x_j)))}_{\text{output measurement distance}} \le \mathbf{K} \cdot \underbrace{\mathbf{D}(E(x_i), E(x_j))}_{\text{input trace distance}} \quad (2)$$

where $\mathbf{K}$ ($0 < \mathbf{K} \le 1$) is the Lipschitz constant that characterizes a QNN circuit on a NISQ device. The smallest $\mathbf{K}$, denoted as $\mathbf{K}^*$, sets the upper limit on the ratio of δ to ϵ for a fair QNN. *A smaller $\mathbf{K}^*$ indicates a more fair QNN*. A QNN is *(ϵ, δ)-fair* if and only if $\delta \ge \mathbf{K}^*\epsilon$. Consider a noiseless QNN, $\mathbf{Q}$, and its noisy counterpart $\mathbf{Q_{Err}}$, which includes only depolarizing noise with an error rate of p. Let $\mathbf{K}^*$ and $\mathbf{K}^*_{\mathbf{Err}}$ respectively represent the Lipschitz constants of $\mathbf{Q}$ and $\mathbf{Q_{Err}}$, we have

$$\mathbf{K}^*_{\mathbf{Err}} = (1 - p) \cdot \mathbf{K}^* \quad (3)$$

Based on Equations 2 and 3, we can utilize p as a proxy for fairness if noises in a synthesized QNN circuit on a particular NISQ device can be unified as depolarizing noise. Note that $\mathbf{K}^*$ can be calculated based on p [5] to quantify the unfairness of a QNN. For simplicity, this work only presents the measured p as a proxy for fairness and omits the $\mathbf{K}^*$ calculations.

Training & Deployment. QNNs are implemented using multi-qubit gates and trained via hybrid quantum-classical methods. Optimized QNNs are then deployed onto NISQ devic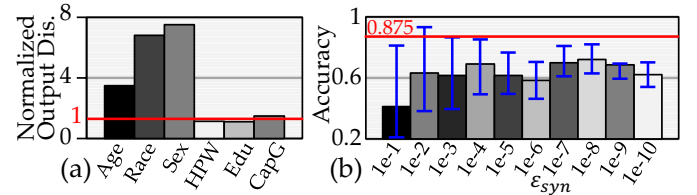es, such as IBM computers, which support a specific native gate set consisting of two 1-qubit gates (`U2`, `U3`) and one 2-qubit gate (`CNOT`). Current 2-qubit gates, with an error rate of 10^{-3} and a coherence time of ten to a hundred microseconds [18], lead to considerable performance degradation. To address this, quantum compilers [19], [20] use approximate synthesis to reduce 2-qubit gate number (i.e., N_{2q}) and circuit depth in the synthesized QNN (i.e., $\mathbf{Q}'$) while ensuring that the unitary difference between $\mathbf{Q}'$ and the uncompiled $\mathbf{Q}$ remains within a predefined budget denoted as ϵ_{syn}. As shown in Figure 1, approximate synthesis has three steps: (1) The original $\mathbf{Q}$ is partitioned into blocks (i.e., $U_1, \cdots, U_N$) with each block having at most S_{blk} (e.g., 3) qubits. (2) The compiler searches for circuit candidates with a minimal N_{2q} within an ϵ_{syn} budget over a tree [19]. It generates a list of acceptable compiled circuits for each individual partition, e.g., $\{U_1^1, U_1^2, \cdots, U_1^{n_1}\}$ for U_1. Note that the number of synthesized circuits can vary across different partitions. (3) One compiled circuit is selected from the candidate list of each partition and sequentially recombined to form the synthesized QNN $\mathbf{Q}'$.

Limitations in Prior QNNs. We conducted a preliminary study to evaluate fairness/accuracy of the best QNN circuit generated by noise-aware training [4] using *Adult Income* [21] dataset. We compiled the circuit using default settings in `BQSKit` [19], [20]. We then selected data pairs (x_i, x_j) with a unit input distance $\mathbf{D}(E(x_i), E(x_j))$, resulting from differences in a single feature group, e.g., *age* or *hour-per-week (HPW)*. We normalized the output distance to that of *HPW* and presented the results in Figure 2(a). *The results reveal unfairness between different feature groups: the same unit difference in Sex and HPW results in a 7.8× difference in output distance.* We also compile the QNN with various $\boldsymbol{\epsilon_{syn}}$ and report the averaged accuracy in Figure 2(b) where the blue line denotes the maximal/minimal accuracy achieved for each $\boldsymbol{\epsilon_{syn}}$, while the red line is the expected ideal accuracy. *Our observation reveals significant reductions and variations in accuracy after QNN synthesis, emphasizing the critical importance of QNN deployment over training in achieving model accuracy.* This is due to the potential compromise of accuracy gains from training by imprudent synthesis settings, which are often unknown to average end-users.

C. Reinforcement Learning

We use deep Q-learning (DQL) [22] that combines a lightweight reinforcement learning (RL) model with deep neural networks (DNNs) to search for the twin objectives of fairness and accuracy in QNN deployment, following its recent success in solving multi-constrained optimization problems in resource-limited scenarios [23].
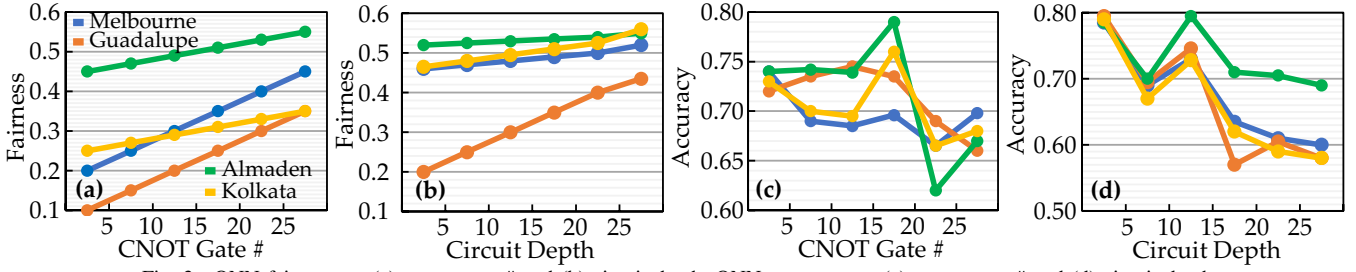
Fig. 3. QNN fairness vs. (a) `CNOT` gate # and (b) circuit depth; QNN accuracy vs. (c) `CNOT` gate # and (d) circuit depth.

Figure 4 shows a standard DQL model. A current state s_t guides the selection of action a_t from the action space $A(s_t)$ using a policy DNN. A target DNN, mirroring the policy DNN, aids in updating policy network parameters.

Fig. 4. A standard DQL model.

The policy DNN takes s_t as input and outputs action values for all possible actions, with $Q_{max}(s_t, a_t)$ indicating the best choice. Upon taking action a_t, the agent receives a reward r_t and transitions to next state s_{t+1}. Accurate prediction of $Q(s_t, a_t)$ by the policy DNN satisfies Equation 4. The loss function in Equation 5 is employed for updating the parameters of both the policy and target DNNs.
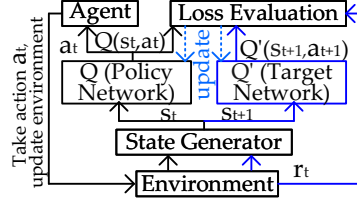
$$Q_{max}(s_t, a_t) = Q'_{max}(s_{t+1}, a_{t+1}) + r_t \qquad (4)$$

$$Loss = [r + \gamma \cdot Q'_{max}(s_{t+1}, a_{t+1}) - Q_{max}(s_t, a_t)]^2 \qquad (5)$$

Challenges in RL-based QNN Deployment. We are the first to apply DQL in QNN deployment. Our work identifies the following challenges and provides corresponding solutions.

1) **States Specification**. The agent's state tensors must contain essential information to determine block connections that meet fairness and accuracy requirements. However, generating a comprehensive and high-quality state representation for QNN circuit deployment is a non-trivial task.

2) **Action Specification**. Each agent action requires an exhaustive search in the complete action space for the current state. In addition, the action specification must align with the input/output structure of the policy/target DNN, ensuring a constant size for the state representation and action specification throughout the design process.

3) **Reward Quantification**. Balancing fairness and accuracy for different QNN designs is challenging, particularly when the QNN is only a partial circuit in the synthesis process.

III. DESIGN SPACE EXPLORATION

A. Design Space Complexity

We explore the design space complexity by performing approximate synthesis [19], [20] using four state-of-the-arts QNNs [1]–[4] onto `IBM_Almaden`. We adjust the block size S_{blk} and threshold ϵ_{syn} to control the number of generated synthesis blocks and the size of the search space. Increasing S_{blk} and decreasing ϵ_{syn} result in a smaller search space. Table II shows the results using settings of S_{blk}=2, ϵ_{syn}=10^{-2}, and S_{blk}=3, ϵ_{syn}=10^{-5}. It is worth mentioning that the results

TABLE II
COMPARISON ON DESIGN SPACE COMPLEXITY.

QNNs	Config.		Space	Acc.	Fairness
	S_{blk}	ϵ_{syn}			
C14 [1]	2	1E-2	43,046,721	0.764	0.3488
	3	1E-5	729	0.724	0.4801
QMLP [2]	2	1E-2	43,046,721	0.772	0.5126
	3	1E-5	729	0.756	0.4923
DATE22 [3]	2	1E-2	98,304	0.624	0.4992
	3	1E-5	6,561	0.764	0.4697
DAC22 [4]	2	1E-2	43,046,721	0.748	0.3293
	3	1E-5	6,561	0.742	0.4985

of DATE22 [3] deviate from the other three QNN circuits due to its unique circuit, using `CNOT` gate instead of parameterized entangled gates found in the other QNNs [1], [2], [4]. Overall, we observe significant differences in the search space for varying S_{blk} and ϵ_{syn}. While reducing the search space effectively reduces the search time, circuits obtained through a smaller search space generally perform worse than those obtained through a larger search space. *Therefore, the vast complexity of the design space in practical QNN synthesis poses a significant challenge, requiring automated and efficient exploration strategies.*

B. Fairness and Accuracy in Synthesized QNNs

We perform noise-aware training [4] and approximate synthesis [19], [20] using a 8-qubit QNN [2]. The target devices are `IBM_Melbourne` and `IBM_Guadalupe`. To investigate the impact of `CNOT` numbers, we randomly selected a compiled circuit and iteratively reduced its `CNOT` count by one, while ensuring it remained within the ϵ_{syn} budget. Simultaneously, we recorded the corresponding fairness and accuracy measurements. We explore the impact of circuit depth caused by `CNOT` displacement by measuring fairness/accuracy for synthesized QNNs with varying depths. We repeated experiments 100 times and report the averaged results.

To evaluate QNN fairness, we utilize the measured noise error rate, p, on the target NISQ device for each synthesized circuit. Despite variations among NISQ computers, we observe a consistent monotonic increase in QNN fairness with higher `CNOT` count and circuit depth, as demonstrated in Figure 3(a)-(b). The QNN accuracy is intricately tied to both the `CNOT` count and circuit depth. Figure 3(c) highlights that the optimal accuracy occurs at a middle sweet spot: exact synthesized circuits with 32 `CNOT` gates introduce error-prone 2-qubit gates, while drastically reducing `CNOT` gates (e.g., to 5) enlarges the unitary difference, both leading to a decrease in accuracy. Figure 3(d) demonstrates that, in most cases, reducing circuit depth enhances accuracy by operating within
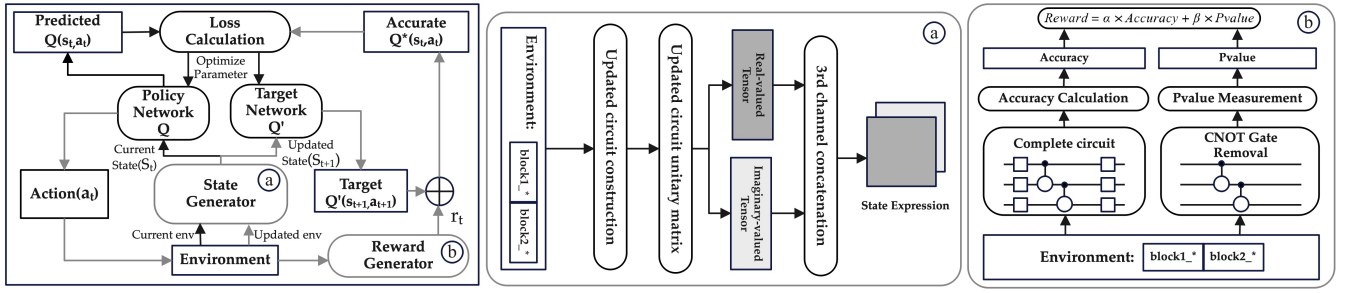
Fig. 5. The JustQ framework. Left: the overall architecture. Middle and Right: functional blocks in JustQ: (a) state generator; (b) reward generator.

the qubit coherence time. However, interestingly, we observe a suboptimal spot across all IBM computers, with some instances (e.g., the 20-qubit `IBM_Almaden`) achieving the best accuracy at certain circuit depths. *In conclusion, existing compilers prioritize minimizing the `CNOT` gate count under an ϵ_{syn} budget. However, our findings demonstrate that the compiled QNN circuits exhibit substantial fairness and accuracy variations, as they are intricately correlated with both the `CNOT` gate number and circuit depth. These critical factors, previously overlooked by compilers, significantly influence the final performance of the compiled circuits.*

IV. JUSTQ

The relationship between QNN accuracy and fairness is complex and influenced by synthesized circuit characteristics, sometimes yielding contradictory results. The intricate design space and limited quantum circuit synthesis knowledge pose challenges for deploying fair and accurate QNNs on NISQ devices. To address these challenges, we propose JustQ, a DQL-based framework for automated fair and accurate QNN deployment. It includes a complete NISQ noise model and measurement method for precise noise assessment. Our framework also offers a flexible optimization objective with a customizable reward function, integrating fairness and accuracy to accommodate diverse application goals.

A. The JustQ Framework

As illustrated in Figure 5, the proposed *JustQ* framework primarily consists of a state generator, a reward generator, and paired policy/target networks. The policy/target networks are initialized using random parameters without prior training. The framework undergoes multiple exploration cycles. Each cycle starts from a blank design, which represents a completely disconnected synthesized QNN with multiple partitions. Throughout these cycles, the framework iteratively refines the blank design by taking actions, gradually shaping it into an optimized QNN configuration.

Training Process. To initiate the exploration cycle, a random circuit from the first QNN partition, e.g., U_1^i, is selected as the initial action. Subsequently, the policy network produces a vector encompassing all possible actions that can be taken in that region, i.e., connecting U_1^i with each candidate synthesized circuit for U_2. The overall reward is calculated, and this information, along with state, action, and value estimates, trains the neural network and updates the policy/target networks. The exploration cycle iterates to optimize the design.

Once the search is complete, full system simulations are conducted to verify and evaluate the design. In general, the policy network generates coarse designs, while the target network efficiently refines them based on prior knowledge, continuously generating more optimal configurations. Unlike traditional supervised learning, this framework doesn't require a training dataset; instead, the policy/target networks gradually train themselves from past exploration cycles.

B. The JustQ Error Model and Measurement

Error Estimation. Prior research [2]–[5] focused on gate-level noise while neglecting *crosstalk* errors. To address this, we introduced a noise-estimation circuit [24] that integrates both *gate* errors and *crosstalk* errors. Enhancing the method further, we incorporated randomized compiling [25] to convert incoherent errors to coherent errors, and quantum circuit engineering [17] to account for the impact of `CNOT` displacement on crosstalk errors. The key steps of our approach are as follows: (1) *Estimation Circuit Construction*: We start with a given QNN partition circuit, referred to as the target circuit. Then, we construct an error estimation circuit by removing all 1-qubit gates from the target circuit. This simplifies the circuit, making it easily simulatable and measurable, as shown in [24]. (2) *Randomized Compiling and Circuit Engineering*: We utilize randomized compiling [25] to convert coherent errors into incoherent errors, which involves inserting a layer of randomized 1-qubit gates before and after each layer of 2-qubit gates. To incorporate the impact of `CNOT` displacement on crosstalk errors, we adopt the approach proposed in [17]. By varying the displacement of `CNOT` gates in each target circuit and creating multiple circuits for later measurement, we can effectively account for the impact of crosstalk errors and enhance the error modeling of our QNNs.

Error Measurement. For each estimation circuit we synthesize, we measure its error rate on the target NISQ computer. Note that we exclude readout errors from our error model because they are decoupled from the QNN circuits. However, we adopt the IBM calibration errors and mitigate their negative impact on QNN performance using previously proposed correction methods [15].

C. RL-based QNN Deployment

Representation of QNNs (States). In JustQ, each state corresponds to a partial QNN circuit. The state tensor is constructed to accurately predict the characteristics of the final completed circuit. To achieve this, a natural choice is to utilize quantum unitary matrices for the partial circuit as they offer
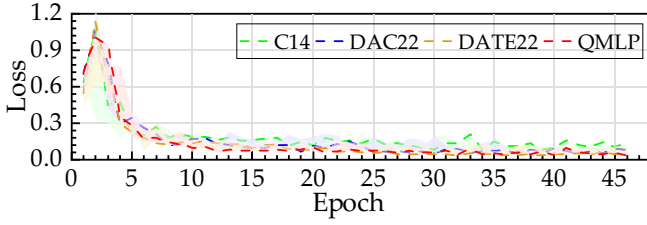
Fig. 6. Loss changes during the training process.

Fig. 7. Q-value changes during the training process.

a complete representation of a quantum state. However, the quantum unitary involves complex values, making it unsuitable for direct processing by the policy/target network. Therefore, we propose a state generator denoted as ❶ in Figure 5. The key idea is to split the complex-valued QNN unitary matrix corresponding to the state into two parts: the real-valued tensor and the imaginary-valued tensor. These two tensors are then concatenated along the third dimension to form the input to the policy/target networks. The output tensor of the action decision network is sliced based on the current state's action space, and the index of the maximum value in this slice represents the action to be taken in the current state.

Representation of Block Selection (Actions). To meet the requirement of a constant-sized action specification throughout the design process with an invariable DNN structure, we sum up the candidate blocks from all partitions and use the combined total as the output size for both the policy and target networks. This approach guarantees that the size of the output tensor remains constant, irrespective of the number of blocks available for selection. In this setting, the action space for each state is closely related to the block index corresponding to the current state. For instance, when the current state is U_1^*, the action space would be $\{U_2^1, U_2^2, \cdots, U_2^{n_2}\}$.

Flexible Optimization Goal (Reward). To ensure the integration of both fairness and accuracy in QNN deployment, a crucial aspect is designing a comprehensive reward function that guides the search effectively. To accommodate diverse design goals for various applications, we formulate a reward function that combines both fairness and accuracy while maintaining flexibility through the use of scaling factors, as shown in Equation 6. To strike a balance between fairness and accuracy, QNN designers can adjust the values of (α, β). For example, by setting (α=0.5, β=0.5), a balanced QNN deployment is achieved, with equal emphasis on both fairness and accuracy. Adjusting the values of α and β, such as setting (α=0.6, β=0.4) or (α=0.4, β=0.6), we can prioritize either fairness or accuracy over the other metric.

$$Reward = \alpha \cdot Fairness + \beta \cdot Accuracy \qquad (6)$$

To maintain a predefined reward, we assess the accuracy and fairness of the partial QNN circuit after each action. As denoted as ❷ in Figure 5, after each action, we measure the circuit's accuracy on real devices using the error model and workflow from the previous section. This estimation provides us with the error rate p, representing the fairness of the current QNN circuit. The final reward is then determined by combining both accuracy and fairness scores.
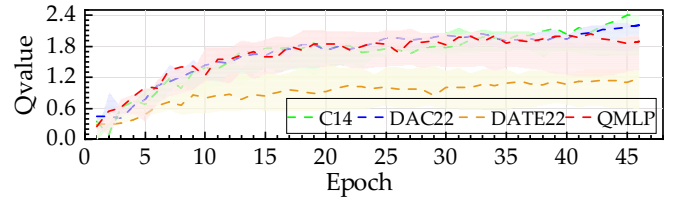
V. EXPERIMENTAL METHODOLOGY

Datasets & Benchmarks. We use two multi-attribute financial datasets: *Adult Income* [21] and *German Credit* [26], similar to [5]. *German Credit* contains 20 attributes, but we selected 8 based on their influence on credit prediction. We obtained a final dataset of 1000 loan applicants, split into 700 for training and 300 for testing due to quantum resource limitations. For *Adult Income*, we used a modified version with 8 attributes, randomly sampling 800 data points for training and 300 for testing. We evaluate JustQ using four recent QNNs: C14 [1], QMLP [2], DATE22 [3], and DAC22 [4]. C14 is a VQC designed for classification. QMLP utilizes a multilayer perceptron architecture. Both DATE22 and DAC22 employ adaptive circuits to enhance their performance.

Synthesis and NISQ Computers. We use BQSKit [19], [20] for approximate synthesis and Qiskit to deploy the synthesized QNNs on NISQ devices. We run experiments on six NISQ computers: 14-qubit `IBM_Melbourne`, 16-qubit `IBM_Guadalupe`, 20-qubit `IBM_Almaden`, 20-qubit `IBM_Boeblingen`, 27-qubit `IBM_Auckland`, and 27-qubit `IBM_Kolkata`.

Reinforcement Learning. We use DQL with ResNet19 as the policy/target network backbone. The search process comprises 1000 iterations to determine the optimal circuit. During RL training, we set the learning rate and decay factor γ respectively as 1e-3 and 0.99. We gradually decrease the exploration probability for ϵ-greedy from 0.05 to a final value of 1e-2.

Schemes. We use the default approximate synthesis [19], [20], denoted as *Quest*, as our baseline. We evaluate five different JustQ configurations against *Quest*: (1) *RL1*: α=0.1, β=0.9; (2) *RL2*: α=0.4, β=0.5; (3) *RL3*: α=0.5, β=0.5; (4) *RL4*: α=0.6, β = 0.4; (5) *RL5*: α=0.9, β=0.1.

VI. RESULTS AND ANALYSIS

Trainability of JustQ. Figures 6 and 7 visualize the training process, showing a consistent decrease in the loss and a steady increase in the Q-value. For instance, in the case of QMLP, the initial training loss value of 0.5356 decreased to 0.0865 by the 20th iteration, while the Q-value increased from 0.31 to 1.8382. Beyond this point, both the loss and Q-value stabilize without significant further changes. These results demonstrate the effective trainability of the *JustQ* model for nontrivial QNN deployment tasks.

Performance Results. We compared the performance of synthesized circuits generated by the baseline *Quest* and *JustQ* on all QNN benchmarks used in this study. Figure 8 and Figure 9 demonstrate that circuits generated by *JustQ* outperform those from the baseline *Quest* compiler in terms of accuracy and
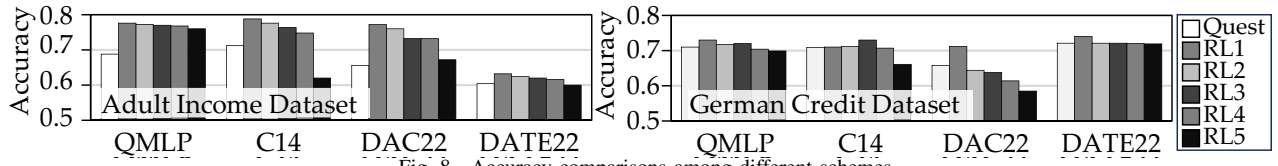
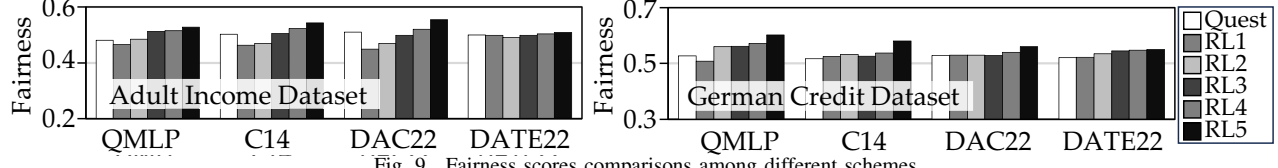Fig. 8. Accuracy comparisons among different schemes.

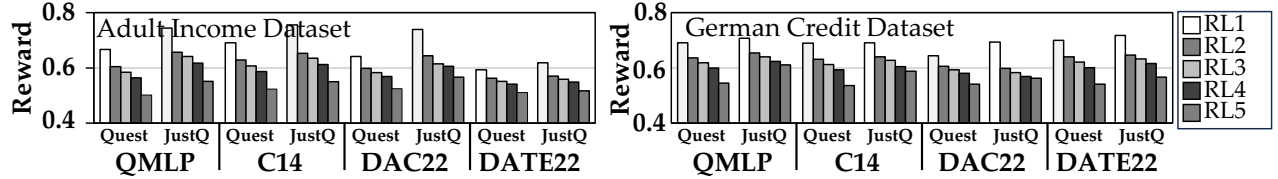Fig. 9. Fairness scores comparisons among different schemes.

Fig. 10. Reward score comparisons among different schemes.

fairness metrics when appropriate weights (>0.5) are applied. We also assess the trade-off between accuracy and fairness by comparing the weighted results of the circuits, as shown in Figure 10. The *JustQ* output circuit achieves a higher weighted sum of accuracy and fairness compared to the *Quest* output circuit. In summary, *Quest* underperforms due to its dual-annealing-based minimization approach [19], which prioritizes output distance over fairness and accuracy, leading to imbalanced circuits. In contrast, *JustQ* achieves a superior balance between accuracy and fairness in the synthesized circuits.

The scaling weights in Equation 6 are crucial for balancing accuracy and fairness. Results in Figure 10 show that higher accuracy weight leads to higher accuracy but lower fairness in the output circuit. For instance, DAC22 generated by *Quest* achieved an accuracy of 0.656 and a fairness score of 0.5101 on *Adult Income*. When balancing accuracy and fairness with (α=0.5, β=0.5), the resulting overall score is 0.583. However, the JustQ approach, under the same scaling weights, achieved an accuracy of 0.732 and a fairness score of 0.5546, resulting in a higher overall score of 0.6433. Notably, the JustQ output circuit outperformed the circuits generated by *Quest* algorithm on all three metrics (i.e., accuracy, fairness, and weighted result), highlighting the potential benefits of this approach.

VII. CONCLUSION

This work initiates research on fair and accurate QNNs by exploring the design space. We emphasize the deployment phase's critical role and its influence on QNN accuracy and fairness. We propose JustQ, a reinforcement learning-based framework for QNN deployment. Experimental results demonstrate JustQ's superiority, producing synthesized QNN models that excel in accuracy and fairness.

ACKNOWLEDGMENTS

REFERENCES

[1] S. Sim *et al.*, "Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms," in *Advanced Quantum Technologies*, 2019.

[2] C. Chu *et al.*, "QMLP: an error-tolerant nonlinear quantum MLP architecture using parameterized two-qubit gates," in *ISLPED*, 2022.

[3] T. Patel *et al.*, "OPTIC: A practical quantum binary classifier for near-term quantum computers," in *DATE*, 2022.

[4] H. Wang *et al.*, "Quantumnat: quantum noise-aware training with noise injection, quantization and normalization," in *DAC*, 2022.

[5] J. Guan *et al.*, "Verifying fairness in quantum machine learning," in *ICCAV*, 2022.

[6] S. Focardi *et al.*, "Quantum option pricing and quantum finance," *The Journal of Derivatives*, 2020.

[7] D. Parsons, "Possible medical and biomedical uses of quantum computing," *Neuroquantology*, 2011.

[8] Y. Cao *et al.*, "Potential of quantum computing for drug discovery," *IBM Journal of Research and Development*, 2018.

[9] C. Dwork *et al.*, "Fairness through awareness," in *ITCS*, 2012.

[10] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, 2018.

[11] D. Ristè *et al.*, "Demonstration of quantum advantage in machine learning," *npj Quantum Information*, 2017.

[12] S. Lloyd *et al.*, "Quantum Generative Adversarial Learning," *Phys. Rev. Lett.*, 2018.

[13] É. Genois *et al.*, "Quantum-tailored machine-learning characterization of a superconducting qubit," *PRX Quantum*, 2021.

[14] L. Hu *et al.*, "Quantum generative adversarial learning in a superconducting quantum circuit," *Science advances*, 2019.

[15] B. Nachman *et al.*, "Unfolding quantum computer readout noise," *npj Quantum Information*, 2020.

[16] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[17] M. Ahsan *et al.*, "Quantum circuit engineering for correcting coherent noise," *Physical Review A*, 2022.

[18] W. Huang *et al.*, "Fidelity benchmarks for two-qubit gates in silicon," *Nature*, 2019.

[19] T. Patel *et al.*, "Quest: systematically approximating quantum circuits for higher output fidelity," in *ASPLOS*, 2022.

[20] E. Younis *et al.*, "Berkeley quantum synthesis toolkit (bqskit) v1," tech. rep., 2021.

[21] "Census Income," 1996. DOI: 10.24432/C5S595.

[22] H. Hasselt, "Double q-learning," *NIPS*, 2010.

[23] J. Fan *et al.*, "A theoretical analysis of deep q-learning," in *Learning for Dynamics and Control*, 2020.

[24] M. Urbanek *et al.*, "Mitigating depolarizing noise on quantum computers with noise-estimation circuits," *Phys. Rev. Lett.*, 2021.

[25] J. J. Wallman and J. Emerson, "Noise tailoring for scalable quantum computation via randomized compiling," *Physical Review A*, 2016.

[26] H. Hofmann, "German Credit Data," 1994. DOI: 10.24432/C5NC77.