ArbiterQ: Improving QNN Convergency and Accuracy by Applying Personalized Model on Heterogeneous Quantum Devices

Tianyao Chu[†]
Zhejiang University
Hangzhou, China
tianyao_chu@zju.edu.cn

Siwei Tan[†]
Zhejiang University
Hangzhou, China
siweitan@zju.edu.cn

Liqiang Lu[*]
Zhejiang University
Hangzhou, China
liqianglu@zju.edu.cn

Jingwen Leng
Shanghai Jiao Tong University
Shanghai, China
leng-jw@sjtu.edu.cn

Fangxin Liu
Shanghai Jiao Tong University
Shanghai, China
liufangxin@sjtu.edu.cn

Congliang Lang
Zhejiang University
Hangzhou, China
langcongliang@zju.edu.cn

Yifan Guo
Zhejiang University
Hangzhou, China
guo2001yifan@zju.edu.cn

Jianwei Yin[*]
Zhejiang University
Hangzhou, China
zjuyjw@cs.zju.edu.cn

2025 62nd ACM/IEEE Design Automation Conference (DAC) | 979-8-3315-0304-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/DAC63849.2025.11132786

***Abstract*—In the current NISQ era, the performance of QNN models is strictly hindered by the limited qubit number and inevitable noise. A natural idea to improve the robustness of QNN is to involve multiple quantum devices. Nevertheless, due to the heterogeneity and instability of quantum devices (e.g., noise, frequent online/offline), training and inference on distributed quantum devices may even destroy the accuracy.**

In this paper, we propose ArbiterQ, a comprehensive QNN framework designed for efficient and high-accuracy training and inference on heterogeneous QPUs. The main innovation of ArbiterQ is it applies personalized models for each QPU via two uniform QNN representations: *model vector* and *behavioral vector*. The model vector specifies the logical-level parameters in the QNN model, while the behavioral vector captures the hardware-level features when implementing the QNN circuit. In this manner, by sharing the gradient among QPUs with similar behavioral vectors, we can effectively leverage parallelism while considering heterogeneity. We also propose shot-oriented inference scheduling, which is a much more fine-grained scheduling that can improve accuracy and balance the workload. The experiments show that ArbiterQ accelerates the training process by 4.03× with 7.87% loss reduction, compared with the previous distributed QNN framework EQC [1].

I. Introduction

Quantum neural network (QNN) has emerged as a popular quantum application that demonstrates practical utility in various tasks, including face recognition [2] and breast cancer prediction [3]. To be more specific, QNN is a new prototype of the neural network to handle machine learning tasks, where the classical feature is encoded into quantum states and subsequently evolved by quantum gates. A QNN model comprises multiple quantum circuit blocks that involve quantum superposition and entanglement to learn the features [4]. Since quantum computing exhibits the exponential expressiveness of quantum states and the huge potential parallelism [5], QNN is a promising model for the next generation of machine learning.

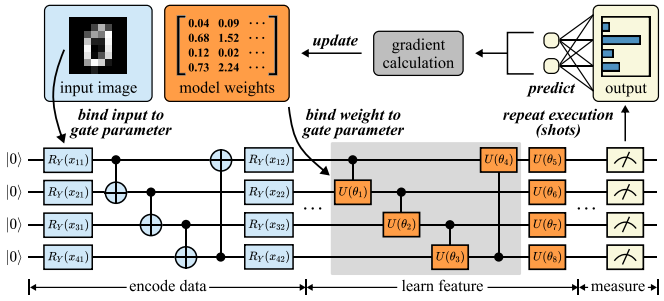
However, the scale of the current quantum processing unit (QPU) hinders the performance of QNN models, which arises from the limited number of qubits, rendering it impossible to deploy a large-scale QNN. For example, the inference of a 4-learning-layer QNN model takes around 0.26 seconds = ($50\mu s$ for circuit + $2\mu s$ for readout + $200\mu s$ for delay) × 1000 executions on the IBM *ibm_osaka* QPU. When training a single instance with parameter shift [6], it takes around 10 seconds = 0.26s × 40 parameters. A natural idea is to employ multiple quantum devices to facilitate the training and inference of the QNN model. However, the traditional methodology of parallel computing is hard to apply to quantum scenarios due to the huge theoretical gap, imperfect quantum devices, and vulnerable quantum state. Here, we summarize the main challenges to implementing QNN on heterogeneous devices:

- **Training challenge.** Even for the same QNN model, its circuit can be different after compilation due to the constraint of basic gates and topology [7]–[10]. On the other hand, quantum devices inevitably encounter various noises, leading to parameter shifting in diverse directions [11]. Therefore, the distinct behavior of the QNN circuit makes it difficult to construct a globally optimal QNN model on a heterogeneous quantum system.
- **Inference challenge.** Conventional distributed deep learning usually employs batch-level parallelism to speed up the training. However, this is unsuitable for quantum computing due to the its fundamentally different operational paradigm. Heterogeneous quantum hardware tends to have diverse preferences on tasks, leading to significant reductions in accuracy during inference compared to training.

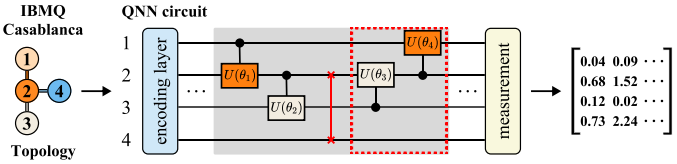
Prior works based on traditional parallelization are grounded in the preciseness of computation, which becomes ineffective when gathering the results from noisy quantum devices. For example, EQC [1] proposes ensemble learning for distributed training by abstracting the noise intensity as a voting weight to eliminate the high-degree noise. However, EQC applies unified model weights for distributed training, lacking the consideration of heterogeneity. Such unified weights can be suboptimal in each individual QPU, thereby resulting in the reduction of accuracy and massive epochs for convergence.

In this paper, we present ArbiterQ, a comprehensive QNN

[†]These authors contribute equally to this work. [*]Corresponding authors.



(a) Structure and operation of QNN.



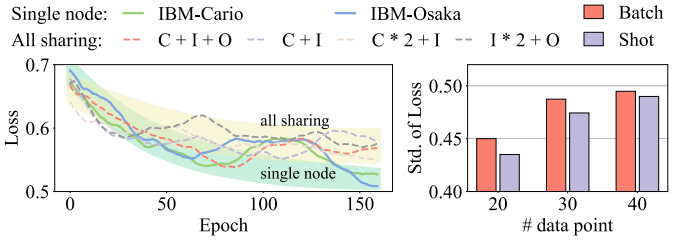
(b) QNN deployed on QPU.

Fig. 1: QNN model and its implementation on QPUs.

framework that improves the performance for training and inference on heterogeneous quantum devices. We first decompose the QNN circuit into *contextual vector* and *topological vector*. The contextual vector records the structural information of each layer in the model, where the element is defined as the cumulative error of a series of basic gates. While the topological vector is derived from the SWAP gates that originate from the topology constraint. These two vectors are further concatenated into a behavioral vector, which expressively describes the circuit behavior after implementing a specific QNN model.

To train on heterogeneous system, we employ a personalized QNN model in each QPU and design the similarity-aware gradient sharing scheme. Clearly, we only unify the network architecture and allow each device to house local model weights. By calculating the distance between the behavioral vectors of different devices, we propose to partially share the gradient among QPUs with similarity. This training approach enables efficient collaboration among variational quantum devices, substantially improving the accuracy and convergence speed. *To enable large-scale inference,* we propose to assign the tasks in the granularity of shot level since the inference results of QNN models rely on multiple shots to obtain the probability. Given the personalized model may exhibit different preferences for tasks, we divide the QPUs into multiple tori and adaptively allocate tasks to each torus. To be specific, we compensate for the noise in each torus, achieved by identifying the cycle period in the behavioral vectors using a discrete Fourier transform. In conclusion, this work makes the following contributions:

- We propose behavioral vectorization for modeling the QNN implementation on heterogeneous devices. By representing the contextual and topological features uniformly, ArbiterQ enables the distributed training with high accuracy and convergence speed.
- We design shot-oriented inference that distributes the tasks to a series of QPU tori. By compensating the noise in each torus, ArbiterQ shrinks the accuracy loss and balances the workload.
- We implement ArbiterQ with both quantum simulator and real quantum devices. Experiments show that ArbiterQ outperforms EQC [1] with 4.03× convergence speed and 7.87% accuracy loss. It also reduces the inference loss by 24.71% compared with batch-based scheduling.



(a) Poor convergence under all-sharing training. (b) Unfair batch-based inference.

Fig. 2: Motivational example of straightforward distributed QNN training (a 2-layer QNN model [12] trained with Wine dataset [13]).

II. Background

A. Quantum Neural Networks

Quantum neural network (QNN) is a new prototype of the neural network to handle machine learning tasks with classical data. QNN extracts the feature of data and outputs normalized results. Based on the rich data encoding structure in quantum states, QNN is a promising way to show the quantum advantage in practical tasks, such as face recognition [2] and breast cancer prediction [3], etc.

Figure 1 (a) depicts a typical QNN model that consists of three functional layers, namely the encoding layer, the learning layer, and the measurement layer. For the image dataset, the pixels are encoded into the parameters of a set of uniformly rotated gates, such as R_Y here. The learning layer comprises several repeated quantum circuit blocks that serve as the backbone of the QNN model, where the gate parameters can be regarded as trainable weights. The measurement layer outputs the probability distribution of quantum states by repeatedly executing and measuring the circuit multiple times. In this paper, we name every single measurement as one "shot".

B. Quantum Processing Unit

Quantum device is the fundamental hardware that employs quantum gates for conducting unitary operators on qubits. In this paper, we focus on the implementation of superconducting quantum devices and name quantum processing units (QPU). The real-world QPUs usually involve different physical configurations, including the number of qubits, decoherence time, topology, and fidelity. These features lead to diverse performance when deploying the same QNN model on heterogeneous QPUs, as shown in Figure 1 (b).

First, quantum devices inevitably suffer from various noises, leading to the errors of quantum gates. These errors can be different at spatial and temporal level due to environmental interaction [14], hardware defect [15], and diverse crosstalk [16]. Second, the topological constraint requires inserting additional SWAP gates to enable double-qubit gates. Therefore, the

executed circuit can be different under heterogeneous QPUs. For example, Figure 1 (b) shows an inserted SWAP gate and relocated following gates in the executable quantum circuits for IBMQ Casablanca QPU. In a word, the circuit design of the QNN model exhibits diversity on heterogeneous quantum devices, resulting in different outputs and different optimal model weights.

C. Motivational Examples

As aforementioned, training a QNN model on multiple QPUs does not necessarily accelerate the training process and improve the convergence, which may even hurt the final accuracy due to the variation of heterogeneous devices. To illustrate this, Figure 2 provides a motivational example that implements a QNN model using three superconducting quantum chips: IBM Cario (C), IBM Osaka (O), and IBM Ithaca (I). In Figure 2 (a), we apply an all-sharing training approach that equally combines the gradients from distributed QPUs and keeps the weights the same across all devices.

We can find that, as the number of epochs increases, the loss curve of these two methods gradually diverges, with the loss of distributed training being much higher than the single-node training. Such discrepancy arises from the heterogeneity of QPUs, leading to sub-optimal gate parameters in QNN circuits. This motivates us to quantify the gate error on the QNN model and orchestrate a gradient-sharing scheme to counteract the noise effect.

Even though we have trained the QNN model with customized weights for each heterogeneous QPU, it is still challenging to enable efficient and stable inference on QPUs with spatial and temporal heterogeneity. Figure 2 (b) shows the standard deviation of loss value using batch-based inference on heterogeneous QPUs. Considering that QNN relies on multiple shots to get the results, we propose a much more fine-grained scheduling called shot-based inference. The shot-based inference has less differential accuracy, which comes from that batch-based scheduling infers each data point with a single QPU only, while shot-based scheduling adaptively divides shots and assigns tasks to smooth the QPU difference.

III. Similarity-aware Personalized Training

The key characteristic of the QNN model is that its optimal weight parameters are closely related to the device properties, such as noise and topology. Therefore, the inherent heterogeneity of quantum devices makes it difficult to leverage the parallelism of distributed computing. To tackle this, we apply a personalized QNN model for each individual QPU. As shown in Figure 3, the QPU keeps a unified model structure while having different weight parameters in the quantum gates. In this manner, the model can better fit heterogeneous hardware properties, leading to higher accuracy and robustness. We collect the weight parameters into a *model vector*. To quantify the heterogeneity of various QPUs and guide the co-optimization of every personalized model, we propose to vectorize the behaviors of implementing the QNN model on a specific QPU, namely *behavioral vector*, with the goal of capturing both

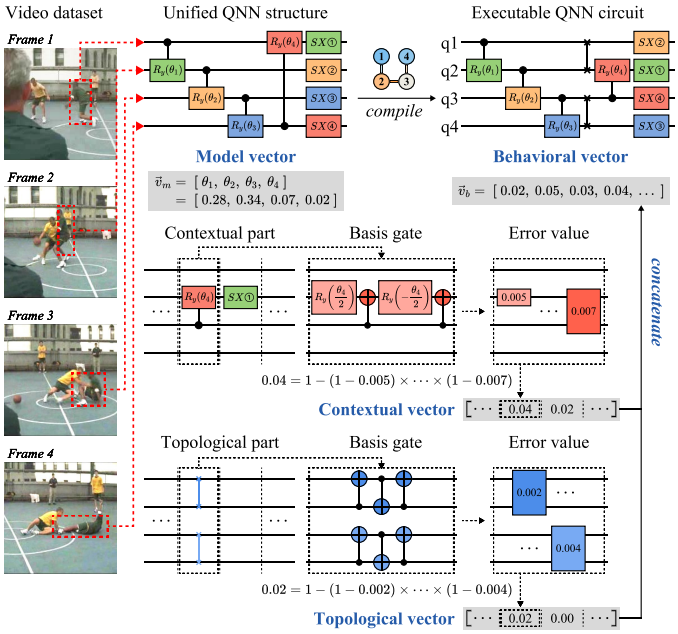


Fig. 3: Vectoring personalized QNN model and its compiled circuit.

contextual information and topological constraints. Then we introduce a similarity-aware gradient-sharing scheme, where the QPUs that show similarity in the behavioral vector are updated together.

A. Vectorizing the QNN Behavior on QPUs

We start by vectorizing the executable circuit after compiling the QNN circuit under the constraints of a specific QPU. During vectorization, elements are arranged by the execution sequence in the QNN circuit, from left-top to right-bottom. Considering that several parallel gates may exchange after compilation, we unify the order following the sequence of the original QNN circuit. For example, we label the four parallel SX gates in the original QNN model from top to bottom. Due to the SWAP gate between qubit 1 and qubit 2, the gate SX① is routed to qubit 2. As a result, we put this gate in front of SX② to ensure the consistency of vectorization.

We decouple the executable circuit into a contextual part and a topological part, as shown in Figure 3. In the contextual vector, each element is derived from the original gate in the QNN model, which aims to record the structural information of each layer in the model. As mentioned before, the topological constraints necessitate SWAP gates to establish entanglement between two unconnected qubits. Thus, the topological part refers to these extra SWAP gates, also with each gate corresponding to one vector element.

The optimal weights mainly depend on the infidelity of quantum gates. Therefore, we formulate each element value as the cumulative error of a series of basic gates: $\vec{v}_c(i) = 1 - \prod_j (1 - e_{ij})$, where $\vec{v}_c(i)$ is the i-th element in the contextual vector, and e_{ij} represents the error of the j-th basis gate decomposed from the i-th gate. We apply a simple formula [17] to calculate the executional error of quantum gate $e = 1 - \mathrm{e}^{-\frac{t}{\tau}} f$, which involves the execution time (t),

depolarization time (τ, for single-qubit gate) or decoherence time (τ, for double-qubit gates), and fidelity (f). Figure 3 gives an example of calculating the element value for $R_y(\theta_4)$ gate, which is decomposed into four basic gates.

For the topological part, its vector length is aligned with the contextual vector. Specifically, the SWAP gate is generated from the two-qubit gate that comprises two nonadjacent qubits, which means the error from SWAP gates will lead to parameter deviation in its corresponding two-qubit gate. Taking Figure 3 as an example, two SWAP gates are inserted for implementing $R_y(\theta_4)$ gate, contributing to the fourth element in the topological vector. Similarly, we can calculate the element value for these two SWAP gates that imply the connectivity of QPU topology. Finally, we concatenate the contextual vector and topological vector, which fully extracts the behavior of deploying the QNN model on QPUs.

B. Similarity-aware Gradient Sharing

The training of personalized QNN models is conducted in the space formed by all behavioral vectors. Our key idea is to group the QPUs that exhibit similar behavioral vectors, which helps to leverage the parallelism of distributed devices and minimize the overhead of heterogeneity. To identify such similarity, we formulate the distance between QPU i and QPU j as follows,

$$dist(i,j) = \frac{\|\vec{v_b^i} - \vec{v_b^j}\|_2}{length(\vec{v_b})} \tag{1}$$

where $length(\vec{v_b})$ denotes the length of the behavioral vector. Then, we can obtain the grouping scheme by setting a distance threshold. Furthermore, we quantify the degree of similarity between every two QPU nodes to ensure that the devices with the highest similarity collaborate best. The similarity is defined as $sim(i,j) = e^{-\kappa \cdot dist(i,j)}$, where κ is a hyperparameter that determines the scope of similarity.

The QPUs that exhibit similarity tend to prefer the same optimal weights. Therefore, we only allow gradient sharing within the group. Concretely, in each iteration, the QPU node executes its local QNN model and calculates the loss by measurement. Given the loss function, we apply parameter shifting [18] to get the local gradient of weights, which will be sent to the QPU that shares the similarity. Accordingly, each QPU also buffers gradients from neighbor nodes and uses them to update the local QNN model. More importantly, the shared gradient is multiplied by the similarity degree to fully utilize the parallelism of near-homogeneous devices. This gradient sharing speeds up the training of each node.

IV. Shot-oriented Inference on QPU Torus

Paralleling large-scale inference tasks is necessitated by the demand for low latency and high throughput. As analyzed in Section II-C, shot-based parallelism, i.e. inferring each task with multiple devices, provides a much more fine-grained opportunity to schedule the inference tasks. With this insight, we propose the shot-oriented distributed inference that flexibly allocates the tasks at the shot level, achieving high accuracy and throughput of the overall system.

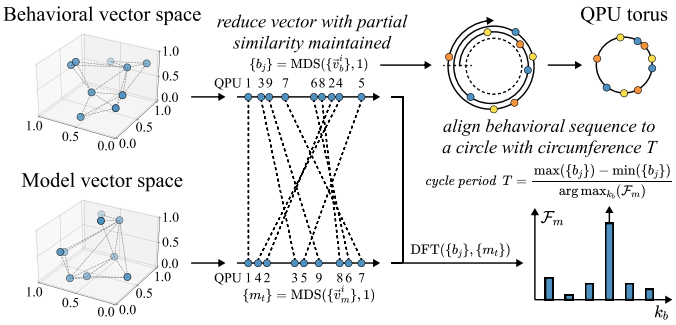


Fig. 4: Mapping QPUs onto torus using DFT.

A. Noise Compensation via QPU Torus

To enable large-scale inference with high accuracy, our goal is to minimize the heterogeneity (inference bias) among personalized QNN models. Clearly, the QPUs with adjacent behavioral vectors may introduce inference bias due to similar noise patterns, resulting in accuracy loss. Therefore, a natural idea to alleviate this is the compensation of various noises, which requires scattering the QPUs with minimum bias. However, the behavioral vector is high-dimensional, giving rise to huge time complexity to arrange QPUs with low behavioral similarity.

Therefore, we first use MDS algorithm [19] to reduce the dimension of the behavioral vector space and the model vector space, as shown in Figure 4. Using MDS algorithm, the nodes in these two spaces are arranged in two one-dimensional sequences meanwhile keeping the same relative distance as before. In other words, two adjacent nodes in the behavioral vector space are also close to each other after MDS. However, the MDS algorithm removes several connection information, which may cause two distant elements in the sequence to share similarities. To handle this, we apply the model sequence to classify such "distant similarity", which involves a non-uniform discrete Fourier transform (DFT) to obtain the initial torus to string the QPU nodes together. Mathematically, considering the model sequence as a signal sequence varying with the behavioral nodes, the sequence of model nodes is transformed as follows,

$$\mathcal{F}_m[k_b] = \sum_j m_t \cdot e^{-i\frac{2\pi}{\max(\{b_j\}) - \min(\{b_j\})} k_b b_j} \tag{2}$$

where $\{b_j\}$ and $\{m_t\}$ is the behavioral sequence and model sequence, respectively. The element m_t in the model sequence points to the same QPU of b_j. For example, in Figure 4, m_2 and b_8 point to the same QPU.

Next, we select the maximum frequency to calculate the cycle period in the behavioral sequence,

$$T = \frac{\max(\{b_j\}) - \min(\{b_j\})}{\arg\max_{k_b}(\mathcal{F}_m)} \tag{3}$$

where k_b is the frequency index of each model weight component after DFT $\mathcal{F}_m$. The period T implies the maximum distance among the nodes in the behavioral vector space. Thus, we construct the initial torus by aligning the behavioral sequence to a circle with a circumference of T, as shown

TABLE I: Training results on different tasks and QNN models.

Benchmark		Convergence epoch					Convergence loss				
		Single-node	All-sharing	EQC [1]	ArbiterQ	Speedup	Single-node	All-sharing	EQC [1]	ArbiterQ	Reduction
Iris [20]	**Model-CRz**	24	38	28	3	9.33×	0.0881	0.0905	0.0737	0.0566	23.23%
	Model-CRx	29	33	20	3	6.67×	0.1136	0.1057	0.0741	0.0608	17.90%
Wine [13]	**Model-CRz**	47	43	37	30	1.23×	0.4717	0.4686	0.4691	0.4550	3.02%
	Model-CRx	54	40	37	9	4.11×	0.4711	0.4739	0.4710	0.4550	3.40%
MNIST [21]	**Model-CRz**	185	198	144	87	1.66×	0.3213	0.3534	0.2858	0.2824	1.19%
	Model-CRx	159	67	98	28	3.50×	0.2910	0.2734	0.2736	0.2626	4.03%
HMDB51 [22]	**Model-CRz**	168	158	178	82	2.17×	0.3116	0.3218	0.3177	0.3102	2.35%
Average		-	-	-	-	4.03×	-	-	-	-	7.87%

The convergence loss is calculated by averaging the loss of the test dataset across all QPUs, without any inference scheduling strategy.

TABLE II: Benchmarks

Dataset	# sample	# feature	# qubits	# QNN weights
Iris [20]	100	4	2	8
Wine [13]	114	13	4	16
MNIST [21]	100	64	6	24
HMDB51 [22]	100	108	10	200

TABLE III: Configuration of simulators

QPU	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
1q-gate infidelity ($\times 10^{-4}$)	2.36	3.06	1.45	5.07	3.41	2.29	4.27	1.72	3.66	2.42
2q-gate infidelity ($\times 10^{-3}$)	7.58	8.67	4.81	4.33	3.69	2.93	4.62	3.66	2.90	9.75
T_1(us)	193	137	349	134	114	103	171	232	260	166
T_2(us)	21.4	67.1	84.7	89.2	96.5	25.7	83.2	47.9	58.4	38.6

in Figure 4. Finally, we divide it into multiple smaller sub-tori by equidistant partition and sort these tori according to their average accuracy. Since MDS algorithm maintains the similarity in the sequence of QPU nodes, the QPUs involved in each sub-torus exhibit low similarity, being able to compensate for the noise with each.

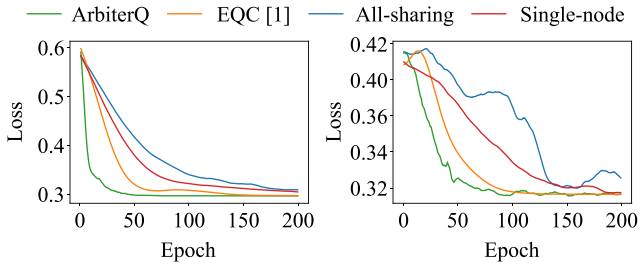
B. Shot-oriented Scheduling

With the available QPUs divided into multiple tori, we start assigning tasks via a warm-up stage to preliminarily sketch the difficulty of target tasks, i.e., the loss of each task. Then, we adopt a greedy-based scheduling that allocates "hard" tasks to the QPU torus with higher accuracy. In each torus, we assign the shots of each task to all QPUs proportionally according to the throughput. Besides, the inference tasks are also proportionally allocated according to the throughput of each QPU torus, resulting in fully balanced workload scheduling.

V. Evaluation

A. Experimental Setup

Datasets. Table II shows the benchmarks used in this paper. We choose two typical pattern recognition datasets, including Iris [20] and Wine [13]. We also conduct the evaluation on an image classification dataset MNIST [21] and a video classification dataset HMDB51 [22] that consists of 6,766 realistic video clips from 51 action categories, such as jump, kiss, and laugh. Similar to the prior methodology, we focus on the binary classification task by selecting two types of objects



(a) MNIST [21]. (b) HMDB51 [22].

Fig. 5: Convergence across different benchmarks.

from these datasets. The proportion of the training dataset and inference dataset is 80% and 20%, respectively.

QNN models. We adopt the angle encoding method [23] to transform classical input to the input quantum state. We apply two types of backbones used in [12], resulting in two QNN models, namely *Model-CRz* and *Model-CRx* (circuit 5 and 14 in Figure 2 of [12]). The hyperparameter κ defined in the gradient sharing is set to 20000.

Platforms. We evaluate ArbiterQ on the real-world quantum platform of ORIGIN QUANTUM [24] accessible via `PyQPanda` (3.8.3.4) [25]. It provides a 72-qubit superconducting quantum chip with a 6×12 grid qubit topology, supporting U3 and CZ gates. The average 1-qubit gate fidelity and 2-qubit gate fidelity are 99.72% and 95.86% respectively. To control the hardware specifications for a fair comparison, we also evaluate ArbiterQ on 10 simulators of quantum computers with 2 to 10 qubits, which are configured as Table III.

Comparison. ArbiterQ is compared with three QNN implementations, including 1) *EQC* [1]: which applies ensembled quantum computing that sets different voting weights according to the noise configuration; 2) *all-sharing*: which updates the gradient by calculating the average value across all QPUs; 3) *single-node*: which trains the QNN model on a single quantum device.

B. Training Performance

Table I compares the training performance associated with the convergence epoch and loss on different benchmarks.

Model convergence. Compared to EQC, all-sharing and single-node, the ArbiterQ accelerates the convergence with an average of 4.03×, 5.19×, 5.04× speedup, respectively.

TABLE IV: Configuration and results of shot-oriented inference

QPU			① ② ③ ④ ⑤ ⑥	① ② ③ ④ ⑤ ⑥ ⑦ ⑧	① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩
Iris [20]	ArbiterQ	Cycle T	0.0119	0.0177	0.0060
		Torus	{1, 3, 5} {2, 4, 6}	{4, 3, 8, 5} {2, 1, 6, 7}	{8, 6, 2, 1} {9, 7, 4} {10, 5, 3}
		Loss	0.0385	0.0354	0.0386
	EQC [1]	Loss	0.0745	0.0650	0.0741
Wine [13]	ArbiterQ	Cycle T	0.0247	0.0267	0.0177
		Torus	{3, 5, 6} {2, 4, 1}	{5, 1, 7, 3} {8, 2, 4, 6}	{2, 7, 4, 3} {1, 6, 10} {9, 8, 5}
		Loss	0.4985	0.4819	0.4626
	EQC [1]	Loss	0.5068	0.4971	0.4710

By sharing the gradients with similar behavioral vectors, parallelism is utilized to learn the image features with fewer iterations. Furthermore, the convergence curve depicted in Figure 5 demonstrates that our approach is more stable compared to other methods. Because our similarity-aware training prevents communication among the QPUs with distinct structural features, eliminating the negative weight sharing. Interestingly, the all-sharing distributed solution shows the worst convergence, indicating that the heterogeneity of quantum devices may even overwhelm the advantages of parallel computing.

Accuracy improvement. ArbiterQ shows the best accuracy loss compared to the other methods. This stems from that we employ the personalized QNN model and locally update weights that fit the device properties. ArbiterQ exhibits the best improvement on the Iris dataset with 17.9% - 23.2% loss reduction. This is because the number of weights for the Iris dataset is quite small, making it easier for ArbiterQ to search the global optimal weights. We observe that the accuracy of EQC is slightly lower than single-node, which comes from that the central model in EQC is limited by the sub-optimal model weights.

C. Inference Performance

Table IV summarizes the accuracy loss on the test dataset using Iris [20] and Wine [13] benchmark. In this table, we provide the detailed configuration of shot-oriented scheduling, including the cycle period after discrete Fourier Transformation and the torus after equidistant partition. Overall, ArbiterQ achieves a 24.71% reduction of loss on these two benchmarks, where we assume that EQC adopts batch-based inference. This suggests that our designed scheduling strategies can significantly enhance the inference accuracy by adaptively assigning the tasks. Moreover, ArbiterQ has lower accuracy loss with a larger number of QPUs as it involves more tori with diverse preferences, which helps to avoid local optimal scheduling.

D. Evaluation on Real-world QPU

We design a 2-qubit QNN model with U3 and CZ basic gates of *origin_wukong* QPU [24]. In particular, we select 4 groups of qubits on this chip and form them as a distributed

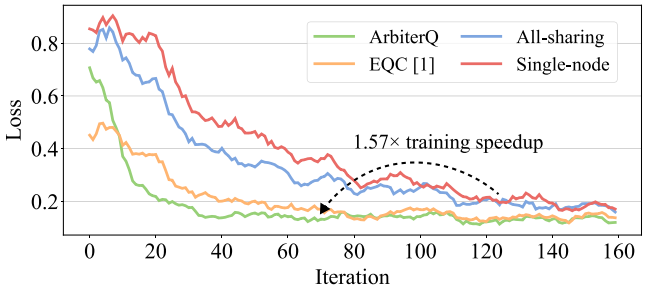


Fig. 6: Model convergence on the real-world QPU.

system. Due to the limited access to the quantum cloud, we only assign partial training data on real QPU. Overall, the results in Figure 6 on the real QPU match the trend in Table I, demonstrating the validness of our design. The final loss of ArbiterQ, EQC, all-sharing, single-node is 0.1045, 0.1092, 0.1397, and 0.1383, respectively. In terms of the convergence speed, ArbiterQ is 1.57×, 1.62×, and 1.64× faster than EQC, all-sharing, and single-node, respectively.

VI. Related Work

Quantum neural network. To enlarge the scale of the problem that the QNN can solve, [26] introduced a quantum-classical hybrid method to image classification. Additionally, [27] researches the effect of different quantum-classical workload splitting techniques during the distributed training QNN model, including circuit parallelization and data parallelization. There are also works that study the multi-device job scheduling for variational quantum algorithms including QNN [28].

Quantum noise mitigation. Researchers have proposed various methods to mitigate noise in QNN. [29] applies the classical error mitigation method to real-world QNN execution. The first effort to enable training on heterogeneous QPUs is EQC [1], which adopts a weighted parameter updating according to the estimated errors of the devices. There are also general optimization methods that can optimize circuit noises, such as QuCT [30] proposed to vectorize the circuit gate by gate for fidelity optimization and unitary decomposition.

VII. Conclusion

We present ArbiterQ, the first framework for full-process deployment of the QNN model on heterogeneous QPUs. Personalized QNN models are tailored to each QPU and the gradient-sharing strategy is facilitated to tackle the architectural heterogeneity of quantum devices. The shot-oriented inference scheduling enhances the accuracy and workload balance effectively. Overall, ArbiterQ represents a significant advancement toward the practical usage of QNNs on real-world quantum computers.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (No.2023YFF0905200), the Zhejiang Provincial Natural Science Foundation of China under Grant (No.LR25F020002), and the National Natural Science Foundation of China (No.623B1001).

REFERENCES

[1] S. Stein, N. Wiebe, Y. Ding, P. Bo, K. Kowalski, N. Baker, J. Ang, and A. Li, "Eqc: Ensembled quantum computing for variational quantum algorithms," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 59–71.

[2] Y. Xu, X. Zhang, and H. Gai, "Quantum neural networks for face recognition classifier," *Procedia Engineering*, vol. 15, pp. 1319–1323, 2011.

[3] P. Li and H. Xiao, "Model and algorithm of quantum-inspired neural network with sequence input based on controlled rotation gates," *Applied Intelligence*, vol. 40, p. 107–126, 2014.

[4] A. Ranjan, T. Patel, D. Silver, H. Gandhi, and D. Tiwari, "Proximl: Building machine learning classifiers for photonic quantum computing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 834–849.

[5] Y. Wu, J. Yao, P. Zhang, and H. Zhai, "Expressivity of quantum neural networks," *Phys. Rev. Res.*, vol. 3, p. L032049, Aug 2021.

[6] D. Wierichs, J. Izaac, C. Wang, and C. Y.-Y. Lin, "General parameter-shift rules for quantum gradients," *Quantum*, vol. 6, p. 677, 2022.

[7] A. Molavi, A. Xu, M. Diges, L. Pick, S. Tannu, and A. Albarghouthi, "Qubit mapping and routing via maxsat," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1078–1091.

[8] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1001–1014.

[9] 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.

[10] T. Tomesh, P. Gokhale, V. Omole, G. S. Ravi, K. N. Smith, J. Viszlai, X.-C. Wu, N. Hardavellas, M. R. Martonosi, and F. T. Chong, "Supermarq: A scalable quantum benchmark suite," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 587–603.

[11] T. Li, L. Lu, Z. Zhao, Z. Tan, S. Tan, and J. Yin, "Qust: Optimizing quantum neural network against spatial and temporal noise biases," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 4, pp. 1434–1447, 2025.

[12] S. Sim, P. D. Johnson, and A. Aspuru-Guzik, "Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms," *Advanced Quantum Technologies*, vol. 2, no. 12, p. 1900070, 2019.

[13] S. Aeberhard, D. Coomans, and O. Y. de Vel, "Comparative analysis of statistical pattern recognition methods in high dimensional settings," *Pattern Recognit.*, vol. 27, pp. 1065–1077, 1994.

[14] D. Suter and G. A. Álvarez, "Colloquium: Protecting quantum information against environmental noise," *Rev. Mod. Phys.*, vol. 88, p. 041001, Oct 2016.

[15] B. Gardas, J. Dziarmaga, W. H. Zurek, and M. Zwolak, "Defects in quantum computers," *Sci. Rep.*, vol. 8, no. 4539, 2018.

[16] P. Zhao, K. Linghu, Z. Li, P. Xu, R. Wang, G. Xue, Y. Jin, and H. Yu, "Quantum crosstalk analysis for simultaneous gate operations on superconducting qubits," *PRX Quantum*, vol. 3, p. 020301, Apr 2022.

[17] Y. R. Sanders, J. J. Wallman, and B. C. Sanders, "Bounding quantum gate error rate based on reported average fidelity," *New Journal of Physics*, vol. 18, no. 1, p. 012002, dec 2015.

[18] H. Wang, Z. Li, J. Gu, Y. Ding, D. Z. Pan, and S. Han, "Qoc: Quantum on-chip training with parameter shift and gradient pruning," in *ACM/IEEE Design Automation Conference (DAC)*, ser. DAC '22. San Francisco, CA, USA: Association for Computing Machinery, 2022, p. 655–660.

[19] N. Saeed, H. Nam, M. I. U. Haq, and D. B. Muhammad Saqib, "A survey on multidimensional scaling," *ACM Comput. Surv.*, vol. 51, no. 3, may 2018.

[20] A. Unwin and K. Kleinman, "The iris data set: In search of the source of virginica," *Significance*, vol. 18, 2021.

[21] Y. Li, R.-G. Zhou, R. Xu, J. Luo, and W. Hu, "A quantum deep convolutional neural network for image recognition," *Quantum Science and Technology*, vol. 5, no. 4, p. 044003, jul 2020.

[22] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "Hmdb: A large video database for human motion recognition," in *2011 International Conference on Computer Vision*. Barcelona, Spain: Institute of Electrical and Electronics Engineers, 2011, pp. 2556–2563.

[23] M. Weigold, J. Barzen, F. Leymann, and M. Salm, "Data encoding patterns for quantum computing," in *Proceedings of the 27th Conference on Pattern Languages of Programs*, ser. PLoP '20. USA: The Hillside Group, 2022.

[24] Q. Quantum, "Qrigin wukong cloud service," https://console.originqc.com.cn/zh/computerServices/services, 2024.

[25] ——, "Pyqpanda document," https://pyqpanda-tutorial-en.readthedocs.io/en/latest/, 2023.

[26] M. Alam, S. Kundu, R. O. Topaloglu, and S. Ghosh, "Quantum-classical hybrid machine learning for image classification (iccad special session paper)," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. Grenoble, France: Institute of Electrical and Electronics Engineers, 2021, pp. 1–7.

[27] T. Khare, R. Majumdar, R. Sangle, A. Ray, P. Seshadri, and Y. Simmhan, "Parallelizing quantum-classical workloads: Profiling the impact of splitting techniques," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1. Los Alamitos, CA, USA: IEEE Computer Society, sep 2023, pp. 990–1000.

[28] M. Wang, P. Das, and P. J. Nair, "Qoncord: A multi-device job scheduling framework for variational quantum algorithms," in *Proceedings of the 57th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '57, 2024.

[29] H. Wang, J. Gu, Y. Ding, Z. Li, F. T. Chong, D. Z. Pan, and S. Han, "Quantumnat: Quantum noise-aware training with noise injection, quantization and normalization," in *ACM/IEEE Design Automation Conference (DAC)*, ser. DAC '22. San Francisco, CA, USA: Association for Computing Machinery, 2022, p. 1–6.

[30] S. Tan, C. Lang, L. Xiang, S. Wang, X. Jia, Z. Tan, T. Li, J. Yin, Y. Shang, A. Python *et al.*, "Quct: A framework for analyzing quantum circuit by extracting contextual and topological features," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 494–508.