Can Noise on Qubits Be Learned in Quantum Neural Network? A Case Study on QuantumFlow

Zhiding Liang\textsuperscript{\dagger}, Zhepeng Wang\textsuperscript{\ddagger}, Junhuan Yang\textsuperscript{\S}, Lei Yang\textsuperscript{\R}, Jinjun Xiong\textsuperscript{\S}, Yiyu Shi\textsuperscript{\D}, Weiwen Jiang\textsuperscript{\ddagger}
\textsuperscript{\dagger}University of Notre Dame, IN, USA. \textsuperscript{\ddagger}George Mason University, VA, USA. \textsuperscript{\S}University of New Mexico, NM, USA. \textsuperscript{\R}University at Buffalo, NY, USA.
zliang5@nd.edu; wjiang8@gmu.edu

Abstract—In the noisy intermediate-scale quantum (NISQ) era, one of the key questions is how to deal with the high noise level existing in physical quantum bits (qubits). Quantum error correction is promising but requires an extensive number (e.g., over 1,000) of physical qubits to create one “perfect” qubit, exceeding the capacity of the existing quantum computers. This paper aims to tackle the noise issue from another angle: instead of creating perfect qubits for general quantum algorithms, we investigate the potential to mitigate the noise issue for dedicate algorithms. Specifically, this paper targets quantum neural network (QNN), and proposes to learn the errors in the training phase, so that the identified QNN model can be resilient to noise. As a result, the implementation of QNN needs no or a small number of additional physical qubits, which is more realistic for the near-term quantum computers. To achieve this goal, an application-specific compiler is essential: on the one hand, the error cannot be learned if the mapping from logical qubits to physical qubits exists randomness; on the other hand, the compiler needs to be efficient so that the lengthy training procedure can be completed in a reasonable time. In this paper, we utilize the recent QNN framework, QuantumFlow, as a case study. Experimental results show that the proposed approach can optimize QNN models for different errors in qubits, achieving up to 28% accuracy improvement compared with the model obtained by the error-agnostic training.

Index Terms—Quantum machine learning, qubit mapping, noisy intermediate-scale quantum, model training

I. INTRODUCTION

Quantum computing has the quantum potential to provide exponential speedup over the classical computers\textsuperscript{8}\textsuperscript{27}\textsuperscript{1}\textsuperscript{22}\textsuperscript{2}, which is considered to be one of the best candidates for solving some of the complicated computational problems that cannot be solved by classical computing in a reasonable time\textsuperscript{3}\textsuperscript{20}\textsuperscript{9}. We are now witnessing the rapid development in both quantum hardware and quantum algorithms: on the hardware side, the newly emerging quantum computers (such as IBM Q\textsuperscript{11}) are developed with superconductor and can perform superposition and quantum entanglement on the physical quantum bits (qubits); on the other side, different quantum algorithms have been devised, such as Shor’s algorithm for primary factorization\textsuperscript{23} and the quantum approximate optimization algorithm (QAOA) for optimization problems\textsuperscript{6}. However, the current quantum computers have a high noise level and intermediate scale, known as noisy intermediate-scale quantum (NISQ). How to deal with the noise with limited number of qubits becomes an immanent problem.

Quantum Error Correction (QEC)\textsuperscript{4} is a promising solution to solve the noise issue in quantum computing by creating “perfect qubits” (e.g., error rate to be less than $10^{-15}$). It applies a set of noisy physical qubits to represent one perfect qubit, which can correct the errors made by the noise and significantly reduce the error rate. Once the QEC is achieved, it can be beneficial for general quantum algorithms. But, there is still a long way to achieve QEC, because it would require a large number (e.g., larger than 1,000\textsuperscript{20}) of physical qubits for a perfect qubit. On the other hand, quantum computer in the NISQ era has a very limited number of qubits; for example, IBM Q has at most 65 qubits on the IBM-Manhattan backend. Therefore, it is critical to find a more efficient way to solve the noise issue in the NISQ era.

Instead of creating perfect qubits for general quantum algorithms with QEC, in this paper, we aim to address the noise issue in an application-specific way, such that no more qubits or a very small number of additional qubits are needed, while the effects of error can be mitigated. More specifically, we would like to explore the design of a dedicated quantum algorithm to make it resilient to noise. Quantum neural network (QNN) is a good candidate to achieve such a goal since the neural network in computing-in-memory (CiM) platforms has already demonstrated its capability to be resilient to noise\textsuperscript{16}. Compared with the design of CiM-based neural network, quantum computing brings more challenges to identify the robust QNNs: (1) in terms of different properties (e.g., topology, error rate, number of qubits) on quantum computers, the implementations (or compilation) of the same QNN on hardware can be different, making the learned model useless; (2) the mapping of QNN to quantum computer needs to be integrated into the training process, leading to lengthy training time; and (3) the update of one weight in the noisy quantum computer may affect other weights, resulting in the traditional training approach to be not applicable. The network structure can affects robust\textsuperscript{17}\textsuperscript{31}\textsuperscript{32}\textsuperscript{15}\textsuperscript{14}\textsuperscript{24}\textsuperscript{29}\textsuperscript{3}\textsuperscript{28}, which we will explore in the future.

In this paper, we base on a recent QNN design, i.e., QuantumFlow\textsuperscript{12}\textsuperscript{13}, to demonstrate how to overcome the above challenges. We first devise an application-specific mapping algorithm. The proposed approach can fix the mapping from logical circuits to physical qubits for different quantum computers; meanwhile, it can minimize the number of gates to be used in physical qubits. In addition, the application-specific mapping follows the same procedure, which can significantly reduce the optimization time to better support training the neural network. Then, based on the proposed mapping algorithm,
we build up an error-aware training framework to identify the robust QNN model. Kindly note that the training procedure is completed at classical computing in our implementation, which can be integrated into quantum computing when the scale of the quantum computer is ready for QNNs.

The contribution of this paper is three-fold as follows:

• To our best knowledge, this is the very first work to learn the quantum error in quantum neural networks, and we demonstrate that a robust quantum neural network, namely QF-RobustNN, can be obtained.

• To support learning qubits’ noise in quantum neural networks, we first devise an application-specific mapping method for one state-of-the-art quantum neural network, QuantumFlow.

• Then, a general training framework is proposed to conduct the error-aware learning, which can be easily extended to support training on the quantum computer in the future when the quantum computer is ready for quantum neural networks, in terms of the number of qubits.

The paper is organized as follows: In Section II, we present the preliminaries and motivations of this work. Section III presents our proposed mapping algorithm and training framework. Evaluation results are reported in Section IV. Section V concludes this paper.

II. PRELIMINARIES

Qubits: In classical computing, the basic unit to represent data is a binary bit (i.e., 0 or 1); similarly quantum computing utilizes quantum bit (qubit) for data representation. Unlike a classical bit that can only represent 0 or 1, the qubit is to represent 0 and 1 simultaneously. More specifically, it describes a state $|\psi\rangle$ with the combination of the 0-state (i.e., $|0\rangle$) and 1-state (i.e., $|1\rangle$); i.e., $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha$ and $\beta$ are the amplitudes of 0-state and 1-state, and $\alpha^2$ and $\beta^2$ represent their probabilities. When $\alpha \neq 0$ and $\beta \neq 0$, the qubit can represent two-state at the same time, known as the “superposition” state.

Quantum Gates: In analogy to the classical logic gates that process classical data, quantum gates manipulate quantum information, i.e., the quantum states qubits. A quantum system is typically initialized to zero state (i.e., $|\psi\rangle = |0\rangle$). Given a Hamiltonian, $H$ has demonstrated that the Hamiltonian can be solved by Schrödinger’s equation, which can be decomposed into various combinations of single-qubit (e.g., H, S, and T gates) and multi-qubit quantum gates (e.g., CNOT gate). It has been proven that the combination of Clifford gates (e.g., CNOT, H, S gates) and T gate is a universal gate set to represent arbitrarily complex gates for quantum computing.

Quantum Circuit: A series of logical gates can construct the classical circuit; likewise, a series of quantum gates form the quantum circuit. According to the function/algorithm needing to be implemented, a quantum circuit can be formed by using the quantum gates in the universal gate set. The length/depth of the circuit reflects the time complexity of the quantum version of the implemented algorithm, and the width of the circuit indicates the number of qubits to be used.

Quantum Processor: The quantum processor (a.k.a., quantum computer) is built upon a set of physical qubits, which are implemented by superconductor [18], Ion-Trap [19], etc. Taking the commonly superconductor-based quantum processor as an example, it has two limitations: (1) qubits are not fully connected; and (2) noise exists in operating qubits or a pair of qubits.

SWAP and BRIDGE for Limitation (1). Figure 1 shows an example with 3 qubits organized in a linear form, where logical qubit $|q_1\rangle$ is mapped to physical qubit $p_i$, i.e., $M(|q_1\rangle) = p_i$. And a CNOT gate operates between $M(|q_0\rangle)$ and $M(|q_2\rangle)$. Obviously, we cannot directly conduct CNOT the physical qubits has no channel for communication. To solve this, a BRIDGE operation and a SWAP operation in Figure 1(c)-(d) are designed. The idea of BRIDGE in Figure 1(c) is to create a unitary operator that performs the function of CNOT gate between two physically separate qubits, while the state of the middle qubit (i.e., $M(|q_1\rangle)$) is not changed. On the other hand, the SWAP in Figure 1(d) is to exchange the quantum states between $p_0$ and $p_1$, as such, after the SWAP, we will have $M(|q_0\rangle) = p_1$ and $M(|q_1\rangle) = p_0$, which can perform CNOT on $|q_0\rangle$ and $|q_1\rangle$. It is worth noticing that both BRIDGE and SWAP can work for the communication between two qubits needing to cross more than 1 physical qubit, but needing more CNOT gates for implementation, introducing higher cost.

Noisy Qubits leading to Limitation (2): Due to the physical material, there exists a high noise level in quantum qubits. In classical computing, the “bit flip” is a typical error; when it comes to the qubit, there are more types of error, which can be concluded as follows:

(1) Qubit-flip Error: This is manifested as a state flip of a qubit from $|0\rangle$ to $|1\rangle$ or versus.

(2) Phase Error: This is manifested as a phase shift of a qubit, which can be regarded as the flip on the phase (e.g., perform

![Fig. 1: BRIDGE and SWAP: (a) A CNOT gate with distance 2. (b) Assumed physical qubits connection. (c) Use the BRIDGE gate to achieve gate (a). (d) Use SWAP gate to achieve gate (a).](image)
Noise can significantly affect the performance of QNN. In Table I, we report the results on the MINST database: QuantumFlow on perfect qubits can obtain a 98.04% of accuracy. When we add bit-flip errors with error rates of 0.1 and 0.01, the accuracy degraded to 53.33% and 88.24%, respectively; if a weight in QNN cannot be mapped to a fixed circuit with a high demand on the compiler efficiency. For example, FFNN and QuantumFlow will build different quantum circuits for different weights.

III. MOTIVATION

Noise can significantly affect the performance of QNN. In Table I[25], we report the results on the MINST database: QuantumFlow on perfect qubits can obtain a 98.04% of accuracy. When we add bit-flip errors with error rates of 0.1 and 0.01, the accuracy degraded to 53.33% and 88.24%, respectively; if a weight in QNN cannot be mapped to a fixed circuit with a high demand on the compiler efficiency. For example, FFNN and QuantumFlow will build different quantum circuits for different weights.

III. MOTIVATION

Noise can significantly affect the performance of QNN. In Table I, we report the results on the MINST database: QuantumFlow on perfect qubits can obtain a 98.04% of accuracy. When we add bit-flip errors with error rates of 0.1 and 0.01, the accuracy degraded to 53.33% and 88.24%, respectively; if a weight in QNN cannot be mapped to a fixed circuit with a high demand on the compiler efficiency. For example, FFNN and QuantumFlow will build different quantum circuits for different weights.

III. MOTIVATION

Noise can significantly affect the performance of QNN. In Table I, we report the results on the MINST database: QuantumFlow on perfect qubits can obtain a 98.04% of accuracy. When we add bit-flip errors with error rates of 0.1 and 0.01, the accuracy degraded to 53.33% and 88.24%, respectively; if a weight in QNN cannot be mapped to a fixed circuit with a high demand on the compiler efficiency. For example, FFNN and QuantumFlow will build different quantum circuits for different weights.
Inference with Error-Aware on Quantum Computer. We will terminate the algorithm and output the identified QF-RobustNN. If the model is converged after the training step, the inference to obtain model accuracy and tune the parameters. Components:

i) shown in Figure 3. The framework is composed of four operations of the quantum-error-aware training framework, as defined as follows.

Quantum-error-aware neural network training: Given a neural network and its implementation on physical qubits, how to search for the weights of the neural network, such that the accuracy of the neural network model can be maximized with the consideration of error on qubits.

In the above problem, the implementation of a given quantum circuit needing to be implemented by a compiler, and therefore, we will further study how to design an application-specific compiler to support the training procedure, which is defined as follows.

Application-specific compiler: Given a dataset, the quantum algorithm, and quantum processor, how to map logical circuit to the quantum processor, such that each qubit is fixed to be mapped to a physical qubit and the circuit length can be minimized.

B. Quantum-error-aware training framework

In this section, we provide detailed procedures for the operation of the quantum-error-aware training framework, as shown in Figure 3. The framework is composed of four components: i) the start point with trained weights; ii) the logical quantum circuit created by QuantumFlow; iii) the mapping from logical circuit to physical qubits; and iv) complete the inference to obtain model accuracy and tune the parameters accordingly. If the model is converged after the training step, we will terminate the algorithm and output the identified QF-RobustNN.

To better demonstrate the framework, we formulate the procedures as follows:

\[ c_i = \text{circ}(W_i) \]
\[ m_i = \text{Map}(c_i, \text{PhyQ}) \]
\[ e_i = \text{Error}(m_i) \]
\[ a_i = \text{Inference}(m_i, e_i) \]

For one identified weight in the \( s \)th iteration, the circuit \( c_i \) is obtained, where the \( \text{circ} \) represents obtain the circuit from a QNN framework (e.g., QuantumFlow). Then, we map the circuit to the physical qubits (PhyQ) with a mapping function \( \text{Map} \), which will be introduced in the next subsection. After this, we will build up the error model \( e_i \) based on the mapping \( m_i \). Finally, we will conduct the inference to obtain the accuracy \( (a_i) \) of the QNN with weight \( W_i \), which is obtained by executing the physical circuit \( m_i \) with the error model \( e_i \). After this, we will be going through the training procedure to update the weights in the QNN.

1. **Train QNN to learn error info:** In the third step in Figure 3, the QNN training model is built up with the purpose of searching for the best training weights is created. The training weight will be learned with the error function \( \text{Error} \) to become aware of the quantum error and make the model resilient to the error \( \text{Error} \). Given an initial weight that performs best in perfect quantum qubits, the training procedure is to update the weights. Here, we can apply different optimization approach to update the weights. For example, we can apply deep reinforcement learning, where a recurrent neural network (RNN)-based controller will guide the updates. More specifically, the accuracy \( a_i \) will be the input of the controller to update the parameters of the RNN, and the RNN will then generate the weights to update the training weights of the first block in Figure 3. The best performing weight is denoted as 'Searched_Weight' by choosing the highest accuracy among all results. Comparing the results of 'Searched_Weight' with the results of the initial weights (called baseline weights), we can evaluate the effectiveness of the proposed training framework, which will be reported in Section V.

2. **Application-specific Mapping** \( C^N Z \) gates are the common gates utilized by QuantumFlow to implement the quantum neural networks, where \( N \) represents there are \( N \) control end with one target qubits using \( Z \) gate. To simplify the presentation, we choose \( C^N Z \) (i.e., CCCZ) gates as an example to describe our application-specific qubit mapping algorithm. The CCCZ gate and the decomposed circuit of the CCCZ gate are illustrated in Figure 3(a). Then, the decomposed circuit of the CCX gate which shows in Figure 3(b) that consists of basic CNOT gates to be executed during CCX gate operation that revealing logic qubits connection relationship. The topology of the physical qubits and logical-to-physical qubits initial mapping is provided in Figure 3(c). The assumed distribution and connectivity of physical qubits are shown in the coupling graph, where \( Q_n \) refers to the logic qubits \( q_n \) corresponding to their physical qubits in the quantum circuit.
Fig. 4: Describe the proposed qubits mapping algorithm steps and the corresponding gate connective behavior. (a) CCCZ gate and the decomposed version of CCCZ gate. (b) CCX gate and the decomposed version of CCX gate. (c) Topology of the physical qubits and logical-to-physical initial mapping.

and $AUX_n$ refers to the auxiliary qubits $aux_n$ corresponding to their physical qubits in a quantum circuit.

To make the qubit mapping for different $CNZ$ gates to be fixed, our design philosophy is to make the logical-to-physical mapping to be the same at the beginning and end of the mapping procedure, meanwhile, we do not incur additional SWAP gates to minimize the circuit length. The procedure of the mapping algorithm is shown in Figure 4(d). The dotted boxes with the same color in Figure 4(a) and Figure 4(d) have correspondence. In the first step, we need first to conduct the qubit level logical-to-physical mapping. This will be followed by the rule of interleaving the computing qubits and auxiliary qubits, and they are mapped to a chain of qubits. For example, Figure 4(c) gives an initial logical-to-physical mapping: $\{g_0, aux_0, q_1, aux_1, q_2, q_3\} \rightarrow \{P_0, P_1, P_2, P_3, P_4, P_5\}$. For convenient corresponding understanding, we refer $\{P_0, P_1, P_2, P_3, P_4, P_5\}$ as $Q_0, AUX_0, Q_1, AUX_1, Q_2, Q_3$ in Figure 4(d). Then, the computation of CCX gates will be arranged in two-phase: forward phase and backward phase.

In forward phase, at $t_1$, the CCX gate in black dotted box act on $g_0$, $q_1$ and $aux_0$ in Figure 4(a). The corresponding decomposed circuit in Figure 4(b) should be executed. It is obvious to see in the coupling graph in Figure 4(c) that $Q_0$ has already been connected to $AUX_0$, and $AUX_0$ is already connected to $Q_1$ by an edge of the coupling graph in black dotted box, so that $g_1$, $g_2$, $g_3$, and $g_4$ in Figure 4(b) can be directly executed. However, we still need to build connection between $Q_0$ and $Q_1$. Thus, we insert a SWAP operation $SWAP(Q_1, AUX_0)$ before $g_5$ to move logic qubit $q_1$ toward $aux_0$. After the SWAP operation, the mapping is update to $\{g_0, aux_0, q_1, aux_1, q_2, q_3\} \rightarrow \{Q_0, Q_1, AUX_0, AUX_1, Q_2, Q_3\}$, and $Q_0$ is connected to $Q_1$ that $g_5$ and $g_6$ can be executed. At $t_2$, a same procedure as $t_1$ is performed, and finally we can see that $|Q_1\rangle$ and $|AUX_1\rangle$ have connection in physical qubits, and we can perform the CZ gate in place at $t_3$.

In backward phase, operations are processed at $t_4$ and $t_5$ in Figure 4(c). At $t_4$, the required connection relationship is $Q_2$ connected to $AUX_1$, $AUX_0$ connected to $AUX_1$, and $Q_2$ connected to $AUX_0$. First step is put is a SWAP operation $SWAP(Q_2, AUX_1)$ in yellow dotted box before executing $g_1$ to move logic qubit $q_2$ toward $aux_1$. And the mapping is update to $\{g_0, aux_0, q_1, aux_1, q_2, q_3\} \rightarrow \{Q_0, Q_1, AUX_0, AUX_1, Q_2, Q_3\}$. This step is in order to build connection between $AUX_0$ and $AUX_1$. And after SWAP operation, $AUX_0$ is connected with $Q_2$ via the edge of bridge, $g_5$ and $g_6$ in yellow dotted box can be executed. At $t_5$, the same procedure is repeated. As a result, the qubit level logical-to-physical mapping is resumed, so that we can achieve our goal to make all weights to be mapped in a fixed and predictable way.

V. RESULTS AND DISCUSSIONS

This section describes the experimental setup and results. We are given an initial weight that performs best on a synthetic dataset in the perfect model as our baseline for accuracy comparison. In addition, the existing compilers will be the baselines for efficiency comparison.

A. Accuracy Comparison
We evaluate the performance of QF-RobustNN on both IBM Qiskit based simulator and the IBM Quantum Processor (i.e., the ibmq_montreal backend with 27 qubits).

For the simulation, we set up different error rates with both bit-flip and phase-shift for the evaluation. Table III reports the results. When the error rate is 0.0001, the obtained QF-RobustNN is exactly the same as the baseline model, because the baseline model is the one with the highest accuracy for perfect qubits, and when the error rate is small, it still performs better than other models. However, when the error rate increased to larger than 0.0005, we observe that QF-RobustNN becomes different from the baseline model. Specifically, QF-RobustNN achieves 1%, 2%, and 5% accuracy improvements for the error rates set to 0.0005, 0.001, 0.01, respectively. Furthermore, when the error increased to 0.05 and 0.1, the baseline model has less than 50% accuracy, but QF-RobustNN identified by the proposed error-aware training framework can achieve 75% accuracy, with 24% and 28% accuracy gain. From the above results, it is clear that the proposed training framework can work for different error settings. In addition, with the increase of qubits’ error, the model that works best for the perfect setting can easily become useless. This emphasizes the importance of conducting error-aware learning.

B. Efficiency Comparison

We apply two existing compilers to be the baseline for efficiency comparison, including (1) Quantum Mapping Examples with Known Optimal (QUEKO) [26] and (2) heuristic-based Hardware-Aware mapping algorithm (HA) [21]. Table IV reports the comparison results on elapsed time.

C. Accuracy, Efficiency, and Circuit Depth

Finally, we execute benchmarks on ibmq_montreal to have an end-to-end comparison between the solution obtained by using HA and QF-RobustNN obtained by the proposed approaches. Table V reports the results. By using the baseline model and our compiling approach, the resultant system provides the accuracy of 74%. By applying QF-RobustNN, we can achieve improvement on accuracy to 75% using HA and 80% using our application-specific compiler. One reason that we can obtain higher accuracy is because our approach can better optimize the circuit with less number of extra SW AP gates, where HA uses 43 SWAP gates while the figure is 12 in our approach. In addition, we can see that our proposed compiler can achieve better efficiency over HA.

VI. CONCLUSION

In the NISQ era, noise is one critical issue needing to be resolved in quantum computing in order to make quantum computing practical. It is still a long way to go before we can achieve perfect qubits, either achieving a breakthrough in material or realize quantum error correction. In such a background, we proposed to design an application-specific optimization procedure to mitigate the effects from the qubits’ noise. Targeting Quantum Neural Network, we proposed the optimization framework to learn the error in the training procedure. We further proposed an application-specific compiler to match the needs of high efficient mapping to support the lengthy training procedure. Results on both simulation and actual quantum computers demonstrate that such an error-aware optimization can improve performance.

ACKNOWLEDGMENT

Access to the IBM Q Network was obtained through the IBM Q Hub at NC State. Special thanks to Zhirui Hu for her extensive discussion and help with this work.

REFERENCES

[1] Frank Arute et al. “Quantum supremacy using a programmable superconducting processor”. In: Nature 574.7779 (2019), pp. 505–510.
[2] Koen Bertels et al. “Quantum Accelerator Stack: A Research Roadmap”. In: arXiv preprint arXiv:2102.02035 (2021).
[3] Jinglei Cheng, Haoqing Deng, and Xuehai Qia. “Accqoc: Accelerating quantum optimal control based pulse generation”. In: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). IEEE. 2020, pp. 543–555.

[4] John Chiaverini et al. “Realization of quantum error correction”. In: Nature 432.7017 (2004), pp. 602–605.

[5] Alexander Cowtan et al. “On the qubit routing problem”. In: arXiv preprint arXiv:1902.08091 (2019).

[6] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. “A quantum approximate optimization algorithm”. In: arXiv preprint arXiv:1411.4028 (2014).

[7] David J Griffiths and Darrell F Schroeter. Introduction to quantum mechanics. Cambridge University Press, 2018.

[8] Lov K Grover. “A fast quantum mechanical algorithm for database search”. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. 1996, pp. 212–219.

[9] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. “Quantum algorithm for linear systems of equations”. In: Physical review letters 103.15 (2009), p. 150502.

[10] Vojtěch Havlíček et al. “Supervised learning with quantum-enhanced feature spaces”. In: Nature 567.7747 (2019), pp. 209–212.

[11] IBM. “IBM Q Experience Device”. In: https://www.research.ibm.com/ibm-q/technology/experience/ (2019).

[12] Weiwen Jiang, Jinjun Xiong, and Yiyu Shi. “A codeign framework of neural networks and quantum circuits towards quantum advantage”. In: Nature communications 12.1 (2021), pp. 1–13.

[13] Weiwen Jiang, Jinjun Xiong, and Yiyu Shi. “When Machine Learning Meets Quantum Computers: A Case Study”. In: 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE. 2021, pp. 593–598.

[14] Weiwen Jiang et al. “Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search”. In: Proceedings of the 56th Annual Design Automation Conference 2019, 2019, pp. 1–6.

[15] Weiwen Jiang et al. “Achieving super-linear speedup across multi-fpga for real-time dnn inference”. In: ACM Transactions on Embedded Computing Systems (TECS) 18.5s (2019), pp. 1–23.

[16] Weiwen Jiang et al. “Device-circuit-architecture co-exploration for computing-in-memory neural accelerators”. In: IEEE Transactions on Computers 70.4 (2020), pp. 595–605.

[17] Weiwen Jiang et al. “Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start”. In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39.11 (2020), pp. 4154–4165.

[18] C Monroe et al. “Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects”. In: Physical Review A 89.2 (2014), p. 022317.

[19] Christopher Monroe and Jungsang Kim. “Scaling the ion trap quantum processor”. In: Science 339.6124 (2013), pp. 1164–1169.

[20] Prakash Murali et al. “Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers”. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. 2019, pp. 1015–1029.

[21] Siyuan Niu et al. “A Hardware-Aware Heuristic for the Qubit Mapping Problem in the NISQ Era”. In: IEEE Transactions on Quantum Engineering 1 (2020), pp. 1–14.

[22] Sutapa Saha et al. “Distributed Computing Model: Classical vs. Quantum vs. Post-Quantum”. In: arXiv preprint arXiv:2012.05781 (2020).

[23] Peter W Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: Proceedings 35th annual symposium on foundations of computer science. IEEE. 1994, pp. 124–134.

[24] Yuhong Song et al. “Dancing along Battery: Enabling Transformer with Run-time Reconfigurability on Mobile Devices”. In: arXiv preprint arXiv:2102.06336 (2021).

[25] Francesco Tacchino et al. “Quantum implementation of an artificial feed-forward neural network”. In: Quantum Science and Technology 5.4 (2020), p. 044010.

[26] Bochen Tan and Jason Cong. “Optimal layout synthesis for quantum computing”. In: 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD). IEEE. 2020, pp. 1–9.

[27] Swamit S Tannu and Moinuddin K Qureshi. “A case for variability-aware policies for nisq-era quantum computers”. In: arXiv preprint arXiv:1805.10224 (2018).

[28] Hanrui Wang et al. “Quantumnas: Noise-adaptive search for robust quantum circuits”. In: arXiv preprint arXiv:2107.10845 (2021).

[29] Zhepeng Wang et al. “Exploration of Quantum Neural Architecture by Mixing Quantum Neuron Designs”. In: IEEE. 2019, pp. 1015–1029.

[30] Xiaoji Xu et al. “Variational algorithms for linear algebra”. In: Science Bulletin (2021).

[31] Lei Yang et al. “Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks”. In: 2020 57th ACM/IEEE Design Automation Conference (DAC). IEEE. 2020, pp. 1–6.

[32] Lei Yang et al. “Co-exploring neural architecture and network-on-chip design for real-time artificial intelligence”. In: 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE. 2020, pp. 85–90.