Quantum Neural Network Compression

Zhirui Hu[1,2], Peiyan Dong[3], Zhepeng Wang[1,2], Youzuo Lin[4], Yanzhi Wang[3], Weiwen Jiang[1,2]

[1]Electrical and Computer Engineering Department, George Mason University, Fairfax, Virginia 22030, United States
[2]Quantum Science and Engineering Center, George Mason University, Fairfax, Virginia 22030, United States
[3]Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115, United States
[4]Earth and Environmental Sciences Division, Los Alamos National Laboratory, NM, 87545, United States
(zhu2@gmu.edu; wjiang8@gmu.edu)

ABSTRACT

Model compression, such as pruning and quantization, has been widely applied to optimize neural networks on resource-limited classical devices. Recently, there are growing interest in variational quantum circuits (VQC), that is, a type of neural network on quantum computers (a.k.a., quantum neural networks). It is well known that the near-term quantum devices have high noise and limited resources (i.e., quantum bits, qubits); yet, how to compress quantum neural networks has not been thoroughly studied. One might think it is straightforward to apply the classical compression techniques to quantum scenarios. However, this paper reveals that there exist differences between the compression of quantum and classical neural networks. Based on our observations, we claim that the compilation/traspilation has to be involved in the compression process. On top of this, we propose the very first systematical framework, namely CompVQC, to compress quantum neural networks (QNNs). In CompVQC, the key component is a novel compression algorithm, which is based on the alternating direction method of multipliers (ADMM) approach. Experiments demonstrate the advantage of the CompVQC, reducing the circuit depth (almost over 2.5×) with a negligible accuracy drop (<1%), which outperforms other competitors. Another promising truth is our CompVQC can indeed promote the robustness of the QNN on the near-term noisy quantum devices.

ACM Reference Format:
Zhirui Hu, Peiyan Dong, Zhepeng Wang, Youzuo Lin, Yanzhi Wang, Weiwen Jiang. 2022. Quantum Neural Network Compression. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22), October 30-November 3, 2022, San Diego, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3508352.3549382

1 INTRODUCTION

Machine learning, in particular neural network, has shown its significant superiority on many practical tasks, like image classification, video segmentation, pattern recognition, etc. These tasks are not only the basis of processing classical data, like digital image or video in computer vision, texts or audio in natural language processing, but they are also able to be applied to study properties of physical systems, such as quantum phase recognition (QPR) [11, 21]. Moreover, with the consistent development of quantum sensing [12] and quantum memory [29], data will be acquired and stored in quantum format. This calls for the in-suit process of data in quantum computing. All these progresses significantly push forward the study of neural networks in quantum computing. As one of the typical quantum neural network (QNN), variational quantum circuit (VQC) is widely used in different applications [5, 7, 14, 25, 30, 36].

Existing works have theoretically shown the effectiveness and efficiency of VQC in the perfect scenario [4]; however, it is well known that the near-term quantum devices have high noise. Coherent and incoherent errors in quantum devices will in turn limit the quantum circuit depth/length (i.e., longer circuit has larger accumulated errors)[5, 10]. As such, when it comes to deploying QNN onto the near-term quantum devices, it is critical to control the circuit depth. In classical neural network (as shown in Figure 1(a)), the compression techniques (including pruning as shown in Figure 1(b) and quantization as shown in Figure 1(c)) are widely used to reduce the execution length. Inspired by this, in this work, we explore how to compress QNN to reduce its circuit length.

It seems straightforward to apply the classical compression techniques [8, 22, 23] to QNN; however, the fundamental difference on neural network designs on quantum computing bring new challenges, where QNN is composed of parameterized quantum gates, as shown in Figure 1(e). First of all, it is unclear which quantum gates can be pruned without the change of the function; as shown in Figure 1(h), in addition to the gates with the value of 0, there exist gates with other parameters, say 2π and 4π, which can also be pruned, called "pruning levels" in this paper. Second, unlike classical neural network having the same cost, the circuit depths of different parameter values in QNN are not fixed, called "quantization levels"; as shown in Figure 1(h), quantization levels of $\pi/2$ and $3\pi/2$ can reduce the circuit depth from 5 to 1 and 3, respectively. Last but not least, the "compression levels" (i.e., pruning level and quantization level) will not only be affected by the value of the parameter, it will also be affected by the type of gates, the quantum devices, and the compiler. In response to all the above challenges, this paper presents a holistic framework, namely CompVQC, to automatically compress a given QNN. More specifically, CompVQC provides the fundamental understandings of pruning and quantization in quantum settings; on top of this, a compression-level look-up-table (LUT) for quantum gates is used in the given QNN for a specific quantum backend and compiler. With the help of compression-level LUT, CompVQC applies alternating direction method of multipliers (ADMM) optimization approach to perform

ICCAD'2022, 30 October - 3 November 2022, San Diego, California, USA

ACM ISBN 978-1-4503-9217-4/22/10…$15.00
https://doi.org/10.1145/3508352.3549382

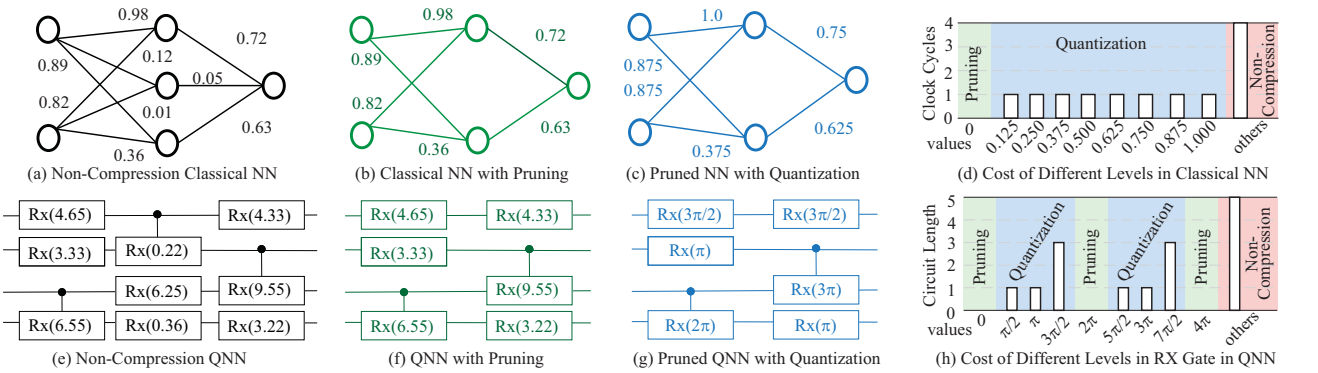

Figure 1: Compression on classical neural network and quantum neural network are fundamentally different: (a)-(c) the pruning and quantization on classical neural networks; (d) value of parameters for pruning and quantization and corresponding clock cycles/latency; (e)-(g) the pruning and quantization on variational quantum circuit, a type of quantum neural networks; (h) quantum gates can be pruned by different parameter values and different quantization values lead to different costs.

compression. Finally, the compressed QNN will be mapped to the actual quantum devices or quantum simulators for execution.

The main contributions of this paper are as follows.

- **Fundamentals.** To the best of our knowledge, this is the first work to rigorously define the compression of quantum neural network, revealing the difference on the compression between quantum neural networks and classical neural networks, and the needs of compilation-aware compression.
- **Framework.** The paper proposes an ADMM-based holistic framework, namely CompVQC, to automatically compress a given quantum neural network (QNN) and deploy it to the target quantum device with a given quantum compiler.
- **Evaluation.** The proposed framework is evaluated on IBM Quantum processors, providing the evidence that the compression can reduce the circuit depth and in turn improve the robustness of QNNs for noisy quantum devices.

Evaluation is conducted on both IBM Aer simulator and IBM Quantum processors. Results on the MNIST and Fashion-MNIST datasets demonstrate that CompVQC can outperform vanilla VQC with significant circuit length reduction with less than 1% accuracy degradation. Compared with existing works that consider only pruning or quantization, CompVQC can reduce the circuit depth, reaching up to 2.5× with a competitive accuracy. On IBM Quantum processors with 2 synthetic datasets, CompVQC achieves upto 20% accuracy improvement, compared with Vanilla VQC.

The remainder of the paper is organized as follows. Section 2 reviews the related background; Section 3 presents the proposed CompVQC framework. Experimental results are provided in Section 4 and concluding remarks are given in Section 5.

2 BACKGROUND AND RELATED WORK

Variational Quantum Circuit (VQC). Variational quantum circuit [1, 15, 38] is a kind of quantum neural networks (QNNs), including parameterized quantum gates, where the parameter in each gate is trainable [35]. Figure 2(a) illustrates a general design of a VQC, which mainly consists of three parts[7]: (1) Input Encoder $\boldsymbol{U(x)}$: quantum gates with non-adaptable parameters for encoding classical data to quantum domain; (2) Trainable Quantum Layers $\boldsymbol{W(\theta)}$: quantum gates with adaptable parameters $\boldsymbol{\theta}$; (3) Measurement $\boldsymbol{M}$:

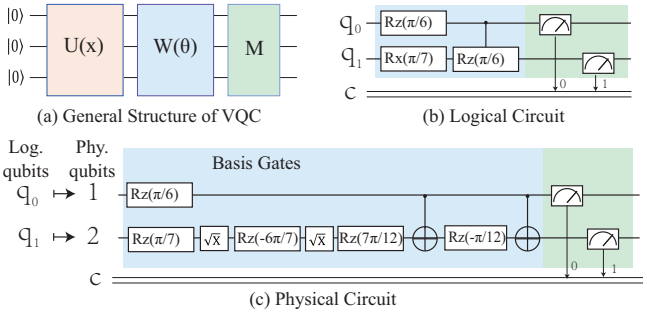

Figure 2: Background of VQC. (a) A general design of VQC; (b)(c) An example of compiling virtual circuit to physical circuit through IBM Qiskit.

the measurement operations to project the quantum output into the classical domain, whose output will be used for prediction.

The current VQC design [7] is usually constructed with logical gates, namely virtual circuit, which cannot be deployed on real quantum computers directly. The virtual circuit needs to be compiled to a physical circuit at first, in order to meet the hardware constraints of the real quantum computer and then be deployed onto it. The details of the compilation are shown in Figure 2. More specifically, there are three steps [33]: (1) Decompose the logical gates into physical basic gates supported by the real quantum computer; (2) Map the logical qubits to physical qubits, which produces a physical circuit; (3) Optimize the generated physical circuits for the shorter circuit depth. After the compilation , the physical gates on the physical circuit is executed following the order of directed acyclic graph (DAG) [28] of the circuit. We observed that the execution time of inference is highly coupled with the depth of the transpiled/physical circuit. Therefore, this paper utilizes transpiled circuit depth (TCD) to present the execution efficiency of VQCs.

Deep Neural Network (DNN) Compression. Pruning [2, 6, 13] and quantization [18, 19, 24] are effective in reducing the model size and speed up its execution, which have been thoroughly explored for classical DNNs. Wth the powerful ADMM optimization framework, [31, 39] achieves a very high compression ratio while maintaining high performance. However, there are few works focusing on the application of compression techniques to QNNs.

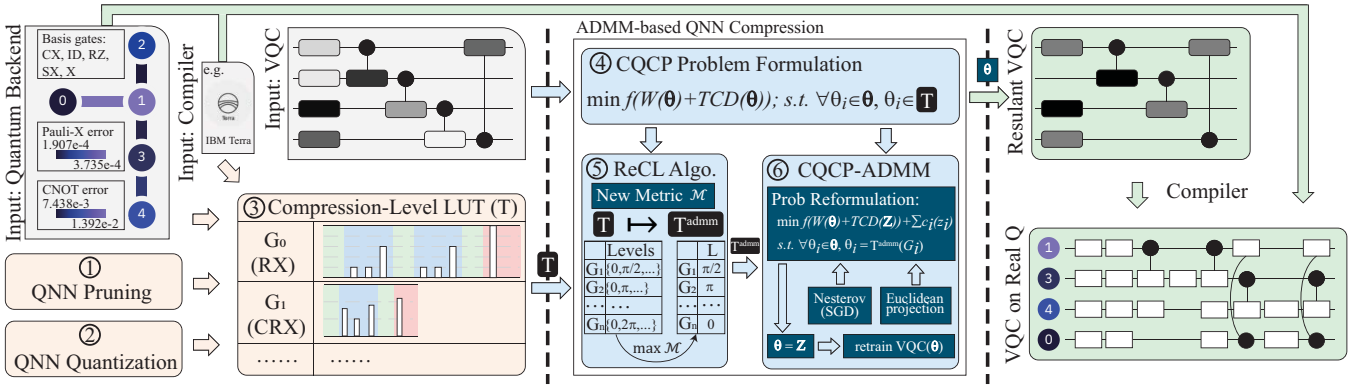

Figure 3: Overview of CompVQC to compress and deploy quantum neural networks via 3 stages.

ADMM Basis. ADMM [3] is extremely capable of solving constrained optimization problems, especially those with non-convex and combinatorial constraints. It decomposes an optimization problem to several sub-problems, solving separately. Given a generic convex constrained optimization problem:

$$\min_x f(x) \quad \text{s.t.} \quad x \subseteq X \tag{1}$$

ADMM transforms this problem to an unconstrained problem, with the indicator function as follows:

$$I_X(x) = \begin{cases} 0 & \text{if } x \in X \\ +\infty & \text{if } x \notin X \end{cases} \tag{2}$$

problem(1) is transformed with an auxiliary variable z and decomposes the objective function into two parts:

$$\min_x f(x) + I_X(x) \quad \text{s.t.} \quad x = z \tag{3}$$

Finally, the Lagrangian multipliers are added to remove the constraints and the target of the optimization becomes minimizing the augmented Lagrangian function L_ρ:

$$L_\rho(x, z, \lambda) = f(x) + I_X(z) + < \lambda, x - z > + \frac{\rho}{2} \|x - z\|_2^2 \tag{4}$$

where λ is the Lagrangian multiplier and ρ is a positive scalar, multiplied by a quadratic penalty term that stops x and z variables from being different. The problem is solved iteratively until the solution can not be further improved.

3 COMPVQC FRAMEWORK

Figure 3 shows the overview of the proposed CompVQC framework. Given a quantum device and a quantum circuit of QNN, CompVQC will go through three stages to compress a QNN and deploy the compressed QNN to the quantum device. We will introduce each stage in detail in the following texts.

Stage 1. *Preparation*

Stage 1 is the preparation stage. It provides the fundamentals to support the compression of QNN models. In this stage, there are three components: ① QNN pruning, which provides the basis of what pruning is in quantum scenario; ② QNN quantization, which formulates the costs of different parameters to identify the QNN quantization; ③ Compression-Level look-up-table (LUT), which is obtained based on the input quantum device and the understanding of QNN pruning and quantization. Note that the outcome of stage 1 is the compression-level LUT, which will be the base to support the compression in stage 2. In the following texts, we will explain each component in stage 1 in details.

① QNN Pruning: A quantum gate can be pruned not only when its parameter/weight is 0.

$$RX(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \tag{5}$$

$$CRX(\gamma) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos\frac{\gamma}{2} & -i\sin\frac{\gamma}{2} \\ 0 & 0 & -i\sin\frac{\gamma}{2} & \cos\frac{\gamma}{2} \end{pmatrix} \tag{6}$$

As shown in Figure 1(h), in addition to the weight of 0, there exist multiple values that can make the quantum gate be pruned. And the operations corresponding to a quantum gate are the main reason of this conclusion. Here, we make an investigation using two typical quantum gates, $RX(\theta)$ for single-qubit gates and $CRX(\gamma)$ for mult-qubit gates. Their matrix representations are shown in Eq. 5 and Eq. 6, respectively. From these equations, we can see that parameters θ and γ will determine the gates' function.

$$RX(0 \text{ or } 4\pi) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix};\ CRX(0 \text{ or } 4\pi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7}$$

When we do the computation, the quantum gates will be operated on qubits. At each time, the system of qubits is in a specific quantum state, represented by a vector $|\psi\rangle$ (note that it can be a single-qubit system or multi-qubit system). The state of the system is changed by the quantum gate G, which is represented by a unitary matrix (e.g., G can be $RX(\theta)$ in Eq. 5). Then, the computation operated on the system is a matrix-vector multiplication, i.e., $|\psi_i'\rangle = G|\psi_i\rangle$. It is clear that if G is an identity matrix, then we have $|\psi_i'\rangle = |\psi_i\rangle$. It indicates that the quantum states are not changed by the quantum gates, and we can prune these gates without changing the function of the quantum circuit. In the previous examples, we observe that when the value of θ and γ are be 0 or 4π, $RX(\theta)$ and $CRX(\gamma)$ will become the identity matrix, as shown in Eq. 7. Kindly note that, we limit the range of parameters to $[0, 4\pi]$ since other values can be mapped to this range.

$$RX(2\pi) = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix};\ CRX(2\pi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{8}$$

We made another interesting observation that when θ is 2π, the corresponding RX gate can also be pruned. As shown in the left

equation in Eq. 8, $RX(2\pi)$ has the negative sign for each element in the identity matrix. Such a gate will flip the phase of each amplitude of a quantum state. Hence, it only affects the global phase without changing the relative phases. Since the change of the global phase is not observable for the measurement, we can still prune this gate. However, when it comes to $CRX(\gamma)$, if we set $\gamma = 2\pi$, we obtain the matrix in the right-hand part of Eq. 8. In this case, since not all phases is flipped, we cannot prune $CRX(2\pi)$. Based on these observations, we give the following lemma for QNN pruning.

Lemma 3.1. Given a quantum gate G and parameter $\boldsymbol{\theta}$, if the function of $G(\boldsymbol{\theta})$ is to multiply the identical matrix (i.e., I) or the negative of identical matrix (i.e., $-I$), then $G(\boldsymbol{\theta})$ can be pruned, and $\boldsymbol{\theta}$ is thus a pruning level for G.

② QNN Quantization: Specified parameters/weights in quantum gates can reduce the circuit depths.

In classical computing, computation using low-bit variables can reduce latency (i.e., clock cycles). For example, compared with 32-bit floating points, the binary number (quantized from floating points) can be processed using logical gates and thus the latency is significantly reduced. We expect to improve the performance (i.e., reduce the circuit depth) of QNNs using the similar way; unfortunately, parameters in quantum computing indicate the rotation of a quantum state, and the data type will not affect the circuit depth.

However, we observe that the value of the parameter will change the circuit depth. This is because the variational quantum circuit (VQC) will finally be compiled to physical quantum processors for execution; while, the given quantum processor $\mathbb{P}$ may not directly support all logical gates in VQC and needs to compile the unsupported gates to multiple available basis gates. For example, in Figure 3, the basis gates provided by the input quantum processor ($\mathbb{P}$) include $\mathbb{S} = \{CX, ID, RZ, SX, X\}$[1]. In this case, other gates (e.g., RX, RY, U, CU) have to be decomposed into the combination of these basis gates, which is contained in the compilation process.

Take $RX(\theta)$ gate as an example. The set $\mathbb{S}$ does not contain $RX(\theta)$ gate. To implement $RX(\theta)$ on the quantum processor $\mathbb{P}$, we compile it to realize its function using the basis gates in $\mathbb{S}$. On IBM Quantum systems, the compiler in Qiskit (denoted as $\mathbb{C}$) will convert $RX(\theta)$ by using a sequence of RZ and SX gates, as below.

$$RX(\theta) = RZ\left(\frac{5\pi}{2}\right) \cdot SX \cdot RZ(\theta + \pi) \cdot SX \cdot RZ(\frac{\pi}{2}) \tag{9}$$

Here, since we apply 5 sequential basis gates to express $RX(\theta)$, the circuit depth will be 5. For special values of θ, we can simplify this decomposition, and thus reduce the circuit depth. When $\theta = 3\pi/2$, we have $RX(3\pi/2) = RZ(-\pi) \cdot SX \cdot RZ(-\pi)$, where the depth is reduced to 3; and when $\theta = \pi/2$, we have $RX(\pi/2) = SX$, where the depth is further reduced to 1.

Define $len_{\mathbb{S},\mathbb{C}}(G(\theta))$ to be the function of the circuit depth of a gate G (say RX) with parameter θ under the basis gate set $\mathbb{S}$ using compiler $\mathbb{C}$. In the above $RX(\theta)$ example using IBM quantum processor with $\mathbb{S} = \{CX, ID, RZ, SX, X\}$ and IBM Qiskit's compiler $\mathbb{C}$, we have $len_{\mathbb{S},\mathbb{C}}(RX(\pi/2)) = 1$, and $len_{\mathbb{S},\mathbb{C}}(RX(3\pi/2)) = 3$. Row "Rx" in Table 1 gives the circuit depth of the compiled RX gate with different parameters. Based on these definitions, we put forward the following lemma to support the QNN quantization.

[1]For the details, please refer to https://en.wikipedia.org/wiki/Quantum_logic_gate

Table 1: circuit depth of compiled quantum gates on IBM quantum processors; parameters are in the range of $[0, 4\pi]$

Gate	0	π	2π	3π	4π	$\pi/2$	$3\pi/2$	$5\pi/2$	$7\pi/2$	others
RX	0	1	0	1	0	1	3	1	3	5
RY	0	2	0	2	0	3	3	3	3	4
CRX	0	8	5	9	0	11	11	11	11	11
CRY	0	8	6	8	0	10	10	10	10	10

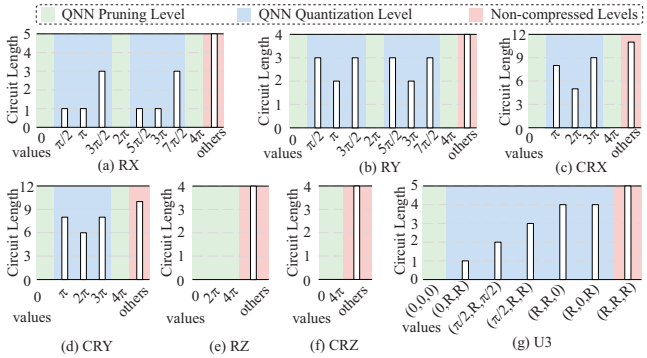

Figure 4: Compression-level LUT, including different quantum gates involved in the input VQC, based on the basis gate set $\mathbb{S} = \{CX, ID, RZ, SX, X\}$ and IBM Qiskit compiler.

Lemma 3.2. Given a quantum processor with a set of basis gate $\mathbb{S}$, a quantum compiler $\mathbb{C}$, a quantum gate G, and a parameter $\boldsymbol{\beta}$, if $len_{\mathbb{S},\mathbb{C}}(G(\boldsymbol{\beta})) < \max_{\forall \boldsymbol{\theta}}\{len_{\mathbb{S},\mathbb{C}}(G(\boldsymbol{\theta}))\}$, then, $G(\boldsymbol{\beta})$ is a quantized quantum gate and $\boldsymbol{\beta}$ is a quantization level for G under $\mathbb{S}$ and $\mathbb{C}$.

We draw another interesting observation from Table 1: the effects of parameters on different gates are different. For example, with the parameter of $3\pi/2$, we can reduce the circuit depth of RX; however, the circuit depth cannot be reduced for CRX and CRY. This motivates us to build a compression-level look-up-table (LUT) for each gate, as discussed below.

③ Compression-Level LUT: Different quantum gates have varied pruning and quantization levels.

As shown in Table 1 and Figure 4, not only quantization levels but also pruning levels of quantum gates are different. We uniformly call them "compression level". Here, one compression level may not be a single parameter, but a tuple $\langle \theta, \phi, \lambda \rangle$. This depends on the parameterized quantum gate. For example, $U3$ and $CU3$ gates have three Euler angles/parameters.

Given a set of basis gate $\mathbb{S}$ and a compiler $\mathbb{C}$, component ③ in *Stage 1* will generate a compression-level LUT for all gates involved in the input VQC. As an example, Figure 4 illustrates the LUT built by CompVQC. Kindly note that, since $\mathbb{S}$ contains RZ gate, Figures 4(e)-(f) only contain the pruning levels without quantization levels. For the ease of illustration, we only present the parameters within the range of $[0, \pi/2]$ of $U3$ gates in Figure 4(g).

Based on Lemma 3.1, Lemma 3.2, and the compression-level LUT built by CompVQC, we have the following Corollary in *Stage 1*.

Corollary 3.1. The quantum neural network (QNN) compression is to reduce the depth of the compiled quantum circuit, which is affected by the set of basis gates supported by the given quantum device and the quantum compiler. That is, QNN compression needs compilation awareness.

***Stage 2**. Compression*

Before introducing the details of how to compress QNNs, we first formulate the compiler-aware QNN compression problem below.

④ Compilation-aware QNN Compression Problem (CQCP).

As shown in Figure 3, there are two inputs of stage 2: (1) a parameterized VQC circuit, denoted as $W(\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the trainable parameters and $\theta_i \in \boldsymbol{\theta}$ is the parameter associated with the i^{th} quantum gate (G_i); (2) the compression-level LUT, denoted as T. For the gate G_i, $T(G_i)$ is a set, including all potential compression levels of G_i. For example, if G_0 is a *CRY* gate, then we have $T(G_0) = \{0, \pi, 2\pi, 3\pi, 4\pi\}$ (see Figure 4(d)); (3) a user-specified quantum compiler $\mathbb{C}$.

The CQCP problem is defined as follows: Given a VQC $W(\boldsymbol{\theta})$, a compression-level LUT T, and a quantum compiler $\mathbb{C}$, the problem is to determine the trainable parameters $\boldsymbol{\theta}$, such that

$$\begin{aligned} \min_{\{\boldsymbol{\theta}\}} \quad & f(W(\boldsymbol{\theta})) + TCD(\boldsymbol{\theta}) \\ s.t. \quad & \forall \theta_i \in \boldsymbol{\theta}, \theta_i \in T(G_i) \end{aligned} \tag{10}$$

where function f represents training loss on the given dataset, and function $TCD(\boldsymbol{\theta})$ is the circuit depth of VQC $W(\boldsymbol{\theta})$ compiled by the compiler $\mathbb{C}$.

In the regular QNN training, we have $T(G_i) \in \mathbb{R}$ (i.e., no constraint), and thus the optimal gates ($G_i(\theta_i)$) for each layer can be obtained by classical gradient-based methods, e.g. stochastic gradient descent (SGD). However, if we directly integrate the QNN compression in the training process, the constraint of $T(G_i)$ becomes combinatorial and non-convex, which prevents the Eq (10) from being solved through gradient-based methods. Besides, $TCD(\boldsymbol{\theta})$ can only be obtained from the quantum circuit compilation, which is not a derivable function.

To solve the aforementioned constrained non-convex optimization CQCP problem, there are two possible directions: (1) directly apply the existing mathematical programming techniques [9, 16], such as employing a stochastic path-integrated differential estimator; (2) transform the constrained non-convex problem to an unconstrained convex problem, such as employing alternating direction method of multipliers (ADMM) approach [3]. The large search space of CQCP brings challenges for both directions, that is, the number of possible combinations of all parameters (s.t., $\theta_i \in T(G_i)$) exponentially grows along with the number of gates in VQC, implying an exponential number of constraints (called "constraint set"). According to previous work [32], the constraint set is highly degenerate, which tends to cause standard nonlinear program solvers, getting bogged down in many steps without convergence of the cost function. In this paper, we explore solutions on direction (2).

⑤ Compression-Level LUT Reconstruction for ADMM

To enable ADMM approach, the first and foremost challenge is how to deal with the large search space, while not affecting the model performance. Our design principle is to reconstruct the constraints of CQCP problem in two ways: (1) we downsize the compression-level LUT, i.e., $T \mapsto T^{admm}$, aiming at reducing the size of search space; (2) we relax the constraints that all parameters have to be compressed, i.e., $\forall \theta_i \in \boldsymbol{\theta}, \theta_i \in T(G_i)$ by enabling a portion of parameters to be real numbers to improve model performance. In this way, we expect that we can reduce the search space without corrupting the model performance.

In component ⑤, we will present an algorithm, namely *ReCL*, to reconstruct the compression-level LUT: $T \mapsto T^{admm}$. More specifically, we will select the most appropriate compression level for each gate as the constraint but not a list of compression levels, i.e., $\forall G_i$, $T^{admm}(G_i) \in T(G_i)$. The downsizing process is conducted based on a new metric, $\mathcal{M}$, which can quantify the impact of the gate value on accuracy and TCD simultaneously. Given a VQC $W(\boldsymbol{\theta})$, the metric $\mathcal{M}(\boldsymbol{\theta}, G_i(\gamma_{i,k}))$ describes the impact of G_i by changing $\theta_i \in \boldsymbol{\theta}$ to $\gamma_{i,k}$ (i.e., the k^{th} compression level in $T(G_i)$). Let $\boldsymbol{\theta^{i,k}}$ be the new parameters, where $\forall \theta_j \in \boldsymbol{\theta}, j \neq i$, we have $\theta_j^{i,k} = \theta_j$, and we have $\theta_i^{i,k} = \gamma_{i,k}$. Then, we have the below formula:

$$\mathcal{M}(\boldsymbol{\theta}, G_i(\gamma_{i,k})) = acc(W(\boldsymbol{\theta^{i,k}})) \cdot \tau(\boldsymbol{\theta^{i,k}}, \boldsymbol{\theta}) \tag{11}$$

where $acc(W(\boldsymbol{\theta^{i,k}}))$ is the accuracy VQC under the new parameters, and $\tau(\boldsymbol{\theta^{i,k}}, \boldsymbol{\theta})$ indicates the compression ratio by changing parameters from $\boldsymbol{\theta}$ to $\boldsymbol{\theta^{i,k}}$, that is, $\tau(\boldsymbol{\theta^{i,k}}, \boldsymbol{\theta}) = \frac{TCD(\boldsymbol{\theta^{i,k}})}{TCD(\boldsymbol{\theta})}$. In this way, we can make sure accuracy and TCD value in the same numerical standard.

On top of the new metric, *ReCL* algorithm is to reconstruct the compression-level LUT (T^{admm}) by setting $T^{admm}(G_i)$ to $\gamma_{i,k}$ yield to the maximum $\mathcal{M}(\boldsymbol{\theta}, G_i(\gamma_{i,k}))$. More specifically, the process is conducted by traversing all quantum gate in VQC, and iteratively change the parameter of one gate G_i in the following two steps.

- Traverse all compression level $\gamma_{i,k} \in T(G_i)$ and obtain metric $\mathcal{M}(\boldsymbol{\theta}, G_i(\gamma_{i,k}))$.
- Set T_{ij}^{admm} be $\gamma_{i,k}$ which leads to the highest metric, i.e. $max_{\forall \gamma_{i,k} \in T(G_i)}[\mathcal{M}(\boldsymbol{\theta}, G_i(\gamma_{i,k}))]$.

From these steps, we can reconstruct and downsize compression-level LUT, which reduces the constraint set of the ADMM and accelerates the training convergence.

⑥ ADMM-Based Compilation-aware QNN Compression

Based on the reconstructed compression-level LUT T^{admm}, we will decompose the original CQCP problem into subproblems that can be solved separately and iteratively until convergence using ADMM, denoted as CQCP-ADMM. As mentioned in component ⑤, in ADMM optimization, we relax the constraint to enable a portion of parameters to be not compressed in each iteration: $T^{admm} \mapsto T^{s,r}$, where T^s is the relaxed compression constraint set, and r is the index of iterations.

With these understandings, we reformulate the CQCP (Eq. 10),

$$\begin{aligned} \min_{\{\theta_i\}} \quad & f(W(\boldsymbol{\theta})) + TCD(\boldsymbol{Z}) + \sum_{\forall z_i \in \boldsymbol{Z}} c_i(Z_i), \\ s.t. \quad & \forall \theta_i \in \boldsymbol{\theta}, \quad \theta_i = T^{admm}(G_i). \end{aligned} \tag{12}$$

where $\boldsymbol{Z}$ is a set of auxiliary variables for subproblem decomposition and $z_i \in \boldsymbol{Z}$ is corresponding to $\theta_i \in \boldsymbol{\theta}$; $f(W(\boldsymbol{\theta})) + TCD(\boldsymbol{Z})$ corresponds to the objective function in the original CQCP problem (Eq. 10). Besides, we use an indicator function (Eq. 13) to move the constraint set toward the objective function in each iteration r.

$$c_i(Z_i) = \begin{cases} 0 & \text{if } \theta_i \in T^{s,r}(G_i), T^{s,r} = T^{admm} \odot mask^r, \\ +\infty & \text{if } otherwise. \end{cases} \tag{13}$$

where $mask^r$ is built to indicate the parameters will be compressed at iteration r. In detail, $mask^r$ is built by three steps:

(1) $\forall G_i$, it calculates the $distance_i^r$ between $\theta_i^{r+1}+\lambda^r$ and $T^{admm}(G_i)$ and the transpiled gate $depth$, $LUT[T^{admm}(G_i)]$;
(2) We sort the importance of all gates in terms of $\alpha \cdot distance_i^r + (1-\alpha)\cdot depth$, $(0 < \alpha < 1)$;
(3) According to the required compression rate τ, $mask^r(G_i) = 1$ if $distance_i^r$ is in the front of $\tau \times |\boldsymbol{G}|$ gates, otherwise $mask^r(G_i) = 0$. Notation $|\boldsymbol{G}|$ is the total number of gates.

With $mask^r$, we define operation $T^{admm} \odot mask^r$ to relax the condition of compression on gate G_i and formulate $T^{s,r}$; in particular, we have $T^{s,r}(G_i) = T^{admm}(G_i)$, if $mask^r = 1$.

By reformulating CQCP problem, it is decomposed into three subproblems in Eq. 12, which can be solved according to the augmented Lagrangian function [17].

$$\theta_i^{r+1} = \arg\min_{\theta_i} f(W(\theta_i)) + \sum_{i=1}^{N} \frac{\rho_i}{2} ||\theta_i - Z_i^r + \rho_i^{-1}\lambda_i^r||_2^2, \tag{14}$$

$$Z_i^{r+1} = \arg\min_{Z_i} \sum_{i=1}^{N} [\frac{\rho_i}{2} ||\theta_i^r - Z_i + \rho_i^{-1}\lambda_i^r||_2^2 + c_i(Z_i)] + TCD(Z), \tag{15a}$$

$$\text{s.t.} \quad Generate\ mask^r\ by\ step(1) \sim (2)\ of mask\ building\ process, \tag{15b}$$

$$Z_i^{r+1} = \begin{cases} Z_i^r, & \text{otherwise,} \\ LUT_s(\theta_i^{r+1} + \lambda_i^r), & \text{if } G_i \in mask^r. \end{cases} \tag{15c}$$

$$\lambda^{r+1} = \lambda^r + \rho(\theta^{r+1} - Z^{r+1}). \tag{16}$$

The first sub-problem (14) can be solved using a general gradient-based method or Nesterov's algorithm. Note that sub-problem (14) is convex. It minimizes the training loss similar to (10), while it tends to produce θ_i close to Z_i in order to minimize the second term in the objective function as well. ρ is the hyperparameters for the regularization factor of the second term.

Solving the second sub-problem (15) ensures that the gate parameters $\boldsymbol{\theta}$ corresponding to $\boldsymbol{Z}$ satisfies the compression-level constraints, $\theta_i \in T_i^s$, and minimizes the TCD. Essentially, optimizing the sub-problem (15) produces a $\boldsymbol{Z}$ with minimal changes to the $\theta^r + \rho^{-1}\lambda^r$ vector and projects this vector into the feasible region of gate parameters. More specifically, $TCD(\boldsymbol{Z})$ is a function of the transpiled depth of the different gates (Table 1,Fig 4). And the solution of Z_i^{r+1} is obtained by (15c) which is the Euclidean projection, i.e. LUT_s the look-up-table operation of $T_i^{s,r}$, and avoids the derivation of $c(\boldsymbol{Z})$ and $TCD(\boldsymbol{Z})$. Please note that a single iteration of CQCP-ADMM produces a solution $(\boldsymbol{Z})$ that meets the T^s constraints, while the result may be sub-optimal in terms of the training loss. Hence, multiple iterations of the CQCP-ADMM algorithm provide a high-quality solution on both the training loss and the TCD reduction.

The third sub-problem (16) updates the λ with the generated θ and Z to accelerate the convergence of the CQCP-ADMM. If θ_i^{r+1} and the corresponding Z_i^{r+1} are close to each other, then λ_i^{r+1} will be close to λ_i^r. Otherwise, by updating λ_i^{r+1}, the divergence of θ_i and Z_i values will be minimized in the next iteration of the CQCP-ADMM. The stopping metric for the CQCP-ADMM algorithm is defined as follows,

$$||\theta_i^r - \theta_i^{r+1}||_2^2 < \zeta, \quad ||Z_i^r - Z_i^{r+1}||_2^2 < \zeta \tag{17}$$

Finally, based on the gap between θ and Z solutions, one can terminate the algorithm once the gap is lower than a certain threshold. Due to the small size of the VQC at this stage, in our experience, 10∼15 iterations of the CQCP-ADMM algorithm are enough to generate superior solutions both on the accuracy performance and TCD reduction of the VQC.

In the final step of the CQCP-ADMM algorithm, we choose Z as the final solution, and the compression requirements and the optimal TCD reduction is met, yet the solution may be sub-optimal in terms of the accuracy performance of the VQC. At this step, we generate the compression position (mask) from the Z and retrain the VQC with the mask to recover the accuracy performance, which for the maximum TDP compression with a slight accuracy degradation.

Stage 3. *Deployment.* After we obtain the compressed VQC, CompVQC will leverage the input compiler to do the transpilation that maps the logical quantum circuit to the given quantum device.

4 COMPVQC: EXPERIMENT

4.1 Experiment Setups

Datasets. We evaluate CompVQC on 2 common classification datasets (MNIST [27] and Fashion MNIST [37]) and synthetic datasets. For MNIST, we extract 2 classes (i.e., digits 3 and 6), denoted as "MNIST-2". For Fashion-MNIST, we extract 2-4 classes from dress, shirt, and T-shirt/top, denoted as "Fashion-MNIST-2", "Fashion-MNIST-3", "Fashion-MNIST-4", respectively. All data are downsampled to 4×4 dimensions. We generate two synthetic datasets, denoted as "Syn-Dataset-4" and "Syn-Dataset-16" to indicate the data with 4 features (i.e., a 4-dimension input vector) and 16 features, respectively. We generate 100 data with 2 classes ($C1$ and $C2$) using 2 normal distributions ($D1$ and $D2$). For class $C1$, the front half of features (i.e., front 2 features for Syn-Dataset-4) follows $D1$, while the tail half of features follows $D2$; data in class $C2$ are generated on the opposite way. For all datasets, we apply 90% samples for the train set and 10% for test set.

QNN models. According to the dimension N of inputs, the QNN involves $\log_2(N)$ qubits. The circuit contains 3 parts: encoding, computation, and measurement.

Encoding. We apply 'amplitude encoding' presented in [20] for Fashion-MNIST and angle encoding' presented in [26] for MNIST and synthetic datasets. For amplitude encoding, we convert the input values to the amplitudes by L2-normalization. For angle encoding, each input dimension is associated with 1 rotation gate and the value is encoded to the phase of the associated rotation gate. For 16-dim input, we use 4RY, 4RZ, 4RX, and 4RY gates; while for 4-dim input, we use 2RY and 2RZ gates.

Computation. All models consist of 'RX', 'RY', 'RZ', 'CRX', 'CRY', and 'CRZ' gates. Each gate has one trainable parameter. We apply the VQC designs in [7] to build QNN for different datasets. There are 50, 50, 30, 22, 22, and 14 gates for MNIST-2, Fashion-MNIST-4, Fashion-MNIST-3, Fashion-MNIST-2, Syn-Dataset-16, and Syn-Dataset-4.

Measurement. The classification results are obtained based on the measurement results, which is the expectation values of states on Pauli-Z basis. Note that for 3 classes on 4 qubits, we divide the first 15 states into 3 groups and sum up amplitudes in each group to generate the output values. Then we process the output values by Softmax to get probabilities for classification.

Table 2: Comparison among different methods on the accuracy performance and the TCD of the VQC

Compression Method	MNIST-2		Fashion-MNIST-2	
	Acc. (vs. Baseline)	TCD (Speedup)	Acc. (vs. Baseline)	TCD (Speedup)
Vanilla VQC	82.74%(0)	121(0)	87.58%(0)	92(0)
Zero-Only-Pruning	80.58%(-2.16%)	70(1.73×)	86.92%(-0.67%)	63(1.46×)
CompVQC-Pruning	81.83%(-0.91%)	74(1.64 ×)	87.41%(-0.17%)	47(1.96×)
CompVQC-Quant	80.99%(-1.75%)	108(1.10×)	86.25%(-1.33%)	74(1.24×)
CompVQC	**81.83%(-0.91%)**	**47(2.57×)**	**87.58%(-0.00%)**	**47(1.96×)**

Quantum devices and compiler configurations. We use both IBM Qiskit Aer simulator and IBM-Q quantum computers via Qiskit APIs. Specifically, we apply Qiskit Terra as the compiler for all QNN circuits. The real-quantum device executions are conducted on "ibm_largos", "ibm_perth", or "ibm_jakarta" backends.

Competitors. We compare the proposed CompVQC against three competitors: (1) Zero-Only Pruning [34], which prunes the VQC model without considering Compilation; (2) CompVQC-Pruning, which is based on CompVQC by only involving pruning levels in the compression-level LUT; (3) CompVQC-Quant, which is also based on CompVQC and only involve quantization levels in the compression-level LUT. For all experiments, we use the "Vanilla VQC" without compression as the baseline.

4.2 Results on Qiskit Aer

A. Comparison among compression approaches.

Table 2 reports the comparison results of different compression methods on MNIST-2 and Fashion-MNIST-2. We compare both accuracy and circuit depth (TCD), using Vanilla VQC as a baseline. Since TCD indicates the latency of the quantum circuit, the reduction in TCD can be expressed by the speedup.

From Table 2, on MNIST-2, our proposed CompVQC performs best among all competitors. Specifically, compared with Vanilla VQC, CompVQC has the lowest TCD, achieving 2.57× reduction with a negligible accuracy drop (−0.91%). On the other end, CompVQC-Quant performed poor on both metrics, with a large accuracy loss (-1.75%) and a slight TCD reduction (1.10×). It is not difficult to understand this result that pruning removes entire gates with a similar loss of accuracy, but quantization only reduces the number of physical quantum gates. On this basis, we conclude that quantum pruning is more effective than quantization.

Another interesting observation is made on two pruning methods: CompVQC-pruning and Zero-Only-Pruning. Both methods can achieve 1.6×-1.7× reduction on TCD, but Zero-Only-Pruning suffers a severe drop in accuracy performance (2.16% ↓); while CompVQC-pruning achieves the same accuracy as CompVQC. When comparing with pruning-only methods, we observe that CompVQC can achieve TCD reduction nearly without accuracy drop.

All the above results show that only prune parameters approaching to 0 cannot provide the best performance, and we need to involve compilation/traspilation in the compression process to generate accurate and efficient quantum circuits; furthermore, with both pruning and quantization, it will lead to the best performance.

Similarly, the results on Fashion-MNIST-2 in Table 2 draw the similar conclusion. The only exception is CompVQC and CompVQC-Pruning can consistently reduce TCD. Nevertheless, the accuracy

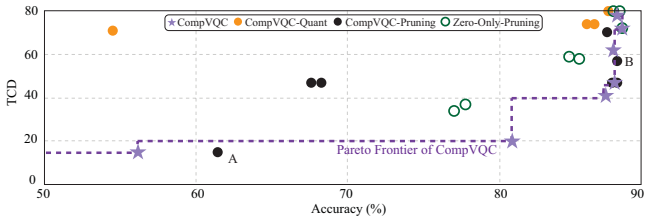

Figure 5: Main results: The Accuracy-Circuit Depth Tradeoff on Fashion-MNIST2

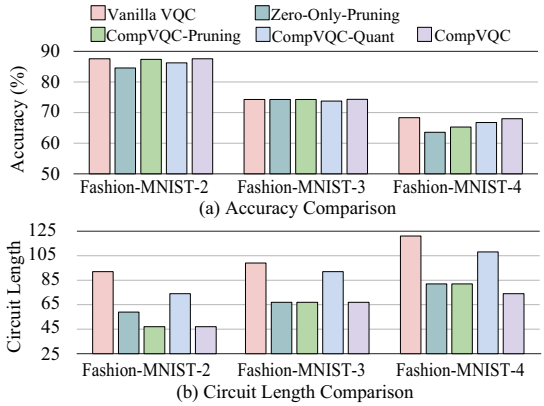

Figure 6: Main Results: CompVQC Scalability on Fashion-MNIST with 2-4 class

of CompVQC is still a little higher than CompVQC-Pruning. These results inspired us to explore the accuracy-TCD tradeoff. **B. Accuracy vs. TCD.** Figure 5 explores the accuracy-TCD tradeoff for different approaches on Fashion-MNIST2 dataset. The x-axis and y-axis represent accuracy and TCD respectively. With the objective in Eq. 10, the right-bottom corner will be the ideal solution, indicating the highest accuracy and the lowest TCD. In the figure, the dotted line represents the Pareto frontier obtained by our proposed CompVQC. It is clear that this Pareto frontier dominates all solutions of CompVQC-Quant and Zero-Only-Pruning. In addition, for CompVQC-Pruning, there are only two solutions escaping the Pareto frontier of CompVQC (point A and point B in the figure). These results emphasize that the proposed CompVQC can make the best tradeoff between accuracy and quantum circuit depth. **C. Scalability on different datasets.** Figure 6 reports the results of all compression approaches on Fashion-MNIST datasets with 2-4 classes. Figure 6(a) shows that CompVQC can achieve higher accuracy over other competitors, which is the closest one to the vanilla VQC. From Figure 6(b), we can see that pruning or quantization methods can significantly reduce the circuit depth (TCD), and CompVQC can consistently obtain the lowest TCD for all datasets. Since the dataset with more classes has more quantum gates, the above results show that the proposed approach can consistently work for different scales of input VQC circuits.

4.3 Results on IBM Quantum Processors

Table 4 reports the results on both Qiskit Aer simulator and the actual IBM Quantum processors using two synthetic datasets. As shown in this table, CompVQC can reduce circuit depth by 2.09× and 2.22× on Syn-Dataset-4 and Syn-Dataset-16, respectively. On accuracy, it is clear that CompVQC can maintain the accuracy to be

Table 3: Optimize different VQC architectures with different numebr of layers on Fashion-MNIST-2 dataset by CompVQC

Method		QLayer22				QLayer30				QLayer38				QLayer50			
		Acc.	vs. BL.	TCD	vs. BL.	Acc.	vs. BL.	TCD	vs. BL.	Acc.	vs. BL.	TCD	vs. BL.	Acc.	vs. BL.	TCD	vs. BL.
Vanilla VQC		87.58%	baseline	92	baseline	87.58%	baseline	99	baseline	90.17%	baseline	117	baseline	90.33%	baseline	121	baseline
CompVQC	(Accuracy)	88.08%	0.50%	72	1.28×	88.08%	0.50%	72	1.38×	90.08%	-0.08%	103	1.14×	89.25%	-1.08%	78	1.55×
CompVQC	(TCD)	87.00%	-0.58%	41	2.24×	87.00%	-0.58%	42	2.36×	87.33%	-2.83%	38	3.08×	87.75%	-2.58%	51	2.37×

Table 4: Case studies on IBM Quantum Processors

Datasets		Syn-Dataset-4		Syn-Dataset-16	
Compression Method		Acc. (vs. Baseline)	TCD (Speedup)	Acc. (vs. Baseline)	TCD (Speedup)
Qiskit Aer	Vanilla VQC	94%(0)	23(0)	96%(0)	51(0)
	Comp-VQC	99%(5%)	11(2.09×)	98%(2%)	23(2.22×)
IBM Q	Vanilla VQC	79%(-15%)	23(1.00×)	86%(-10%)	51(1.00×)
	Comp-VQC	99%(5%)	11(2.09×)	98%(2%)	23(2.22×)

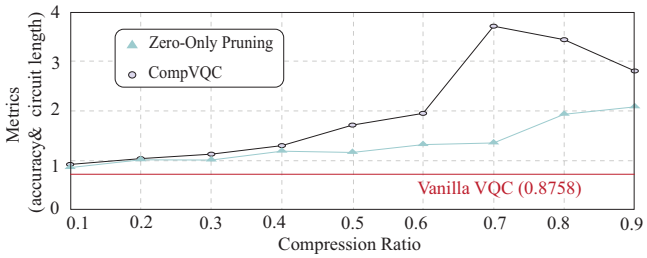

Figure 7: CompVQC on different compression ratio

the same on the noisy real quantum computers for both datasets; on the other hand, the Vanilla VQC has a 15% and 10% accuracy drop on two datasets respectively, after deploying the VQC to the real quantum processors. The above results demonstrate that the quantum compression can reduce circuit depth, and in turn make the QNN more robust to the noise on the real quantum processors. This also points out that quantum compression facilitates the practical deployment of quantum circuits on real machines. We further conduct a group of experiments with Syn-Dataset-4 on different backends of IBM Quantum processors. Results are reported in Table 5. It is clear that for 3 backends (i.e., "ibm_lagos", "ibm_perth", and "ibm_jakarta") with different noise levels, the propsoed CompVQC can always achieve over 98% accuracy, while the accuracy of Vanilla VQC is from 79% to 92%. Results show that CompVQC obtain robust VQC target different quantum devices.

4.4 Ablation Studies

All the above results have already demonstrated the superiors of CompVQC and the scalability of CompVQC on different datasets. In this subsection, we conduct ablation studies on different compression ratio and different sizes of VQC on the same dataset.

A. CompVQC consistently outperforms compilation-agnostic compression at different Compression Ratios

Figure 7 reports the comparison of CompVQC and Zero-Only-Pruning on different ratios. The y-axis indicates the metric in Eq. 11, which is the higher the better. As a reference, the metric of vanilla VQC is 0.875. From this figure, it is clear that CompVQC consistently achieves higher metric over the existing Zero-Only-Pruning; moreover, the improvement is enlarged along with the increase in compression ratio from 0.1 to 0.7. When the compression ratio reaches 0.8, we observe a decrease of metrics on CompVQC, this is mainly because the accuracy is decreased. Overall, we can see that CompVQC consistently performs better than the existing compilation-agnostic compression.

Table 5: CompVQC on different backends

Acc.(vs. Baseline)	ibm_lagos	ibm_perth	ibm_jakarta
Vanilla VQC(TCD=23)	79%(0)	86%(0)	92%(0)
CompVQC(TCD=11)	99%(20%)	98%(12%)	100%(8%)

B. On the same dataset, CompVQC can significantly reduce circuit depth with guaranteed accuracy.

Table 3 reports the results of CompVQC on Fashion-MNIST-2 dataset applying VQC with a different number of gates. Note VQC will repeat a block/layer by multiple times to integrate more parameters. As shown in this table, with more gates, say "QLayer50" with 50 gates, Vanilla VQC can obtain higher accuracy. In the table, we have two rows of CompVQC, the one with "(Accuracy)" indicates the results with the superior accuracy in the exploration results, while the other one with "(TCD)" is for the exploration with an accuracy threshold 87.00%. From this table, it is clear that the accuracy-oriented CompVQC can reduce circuit depth while maintaining high accuracy, for the VQC with 22 and 30 layers, CompVQC can achieve 1.28× and 1.38× reduction on circuit depth, while obtaining 0.5% accuracy gain. In addition, for the TCD-oriented CompVQC, it can achieve more than 2× reduction on circuit depth while maintaining the accuracy to be at least 87%. These results again show the scalability of the proposed framework and its ability to make the best accuracy and circuit depth tradeoff.

5 CONCLUSION

In this work, we first study the fundamental of the compression (including both pruning and quantization) in QNN, revealing the difference of compression between QNNs and classical DNNs. We proposed CompVQC, a framework that automatically compresses a given quantum neural network and deploys it to the target quantum device with a specific quantum compiler. Evaluations on MNIST & Fashion-MNIST show the effectiveness of CompVQC, which is the superior among the three competitors. We also deploy the vanilla model and compressed model by CompVQC to real IBM quantum devices, and find that CompVQC can make the QNN more robust to the noise by reducing the circuit depth. We envision the fundamental of QNN compression and the optimization studied in this paper will push the real machine learning applications to the near-term quantum devices.

ACKNOWLEDGEMENT

This work was supported in part by the National Seurity Education Center (NSEC) Quantum Sensing, the George Mason University Quantum Science & Engineering Center, and National Science Foundation CCF-2008514.

REFERENCES

[1] MV Altaisky. 2001. Quantum neural network. *arXiv preprint quant-ph/0107012* (2001).

[2] Ren Ao, Zhang Tao, Wang Yuhao, Lin Sheng, Dong Peiyan, Chen Yen-kuang, Xie Yuan, and Wang Yanzhi. 2020. Darb: A density-adaptive regular-block pruning for deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5495–5502.

[3] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning* 3, 1 (2011), 1–122.

[4] Carlos Bravo-Prieto, Ryan LaRose, Marco Cerezo, Yigit Subasi, Lukasz Cincio, and Patrick Coles. 2020. Variational quantum linear solver: A hybrid algorithm for linear systems. *Bulletin of the American Physical Society* 65 (2020).

[5] Lukas Burgholzer, Robert Wille, and Richard Kueng. 2022. Characteristics of reversible circuits for error detection. *Array* 14 (2022), 100165.

[6] Sung-En Chang, Yanyu Li, Mengshu Sun, Runbin Shi, Hayden K-H So, Xuehai Qian, Yanzhi Wang, and Xue Lin. 2021. Mix and match: A novel fpga-centric deep neural network quantization framework. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 208–220.

[7] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. 2020. Variational quantum circuits for deep reinforcement learning. *IEEE Access* 8 (2020), 141007–141024.

[8] Hsin-Pai Cheng, Yuanjun Huang, Xuyang Guo, Feng Yan, Wei Wen, Hai Li, Yiran Chen, et al. 2018. Differentiable Fine-grained Quantization for Deep Neural Network Compression. In *NIPS 2018 Workshop on Compact Deep Neural Networks with Industrial Applications (CDNNRIA)*.

[9] Yuejie Chi, Yue M Lu, and Yuxin Chen. 2019. Nonconvex optimization meets low-rank matrix factorization: An overview. *IEEE Transactions on Signal Processing* 67, 20 (2019), 5239–5269.

[10] James P Clemens, Shabnam Siddiqui, and Julio Gea-Banacloche. 2004. Quantum error correction against correlated noise. *Physical Review A* 69, 6 (2004), 062313.

[11] Iris Cong, Soonwon Choi, and Mikhail D Lukin. 2019. Quantum convolutional neural networks. *Nature Physics* 15, 12 (2019), 1273–1278.

[12] Christian L Degen, F Reinhard, and Paola Cappellaro. 2017. Quantum sensing. *Reviews of modern physics* 89, 3 (2017), 035002.

[13] Peiyan Dong, Siyue Wang, Wei Niu, Chengming Zhang, Sheng Lin, Zhengang Li, Yifan Gong, Bin Ren, Xue Lin, and Dingwen Tao. 2020. Rtmobile: Beyond real-time mobile acceleration of rnns for speech recognition. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[14] Motohiko Ezawa. 2021. Variational Quantum Support Vector Machine based on \Γ matrix expansion and Variational Universal-Quantum-State Generator. *arXiv preprint arXiv:2101.07966* (2021).

[15] Alexandr A Ezhov and Dan Ventura. 2000. Quantum neural networks. In *Future directions for intelligent systems and information sciences*. Springer, 213–235.

[16] Cong Fang, Chris Junchi Li, Zhouchen Lin, and Tong Zhang. 2018. Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. *Advances in Neural Information Processing Systems* 31 (2018).

[17] Michel Fortin and Roland Glowinski. 2000. *Augmented Lagrangian methods: applications to the numerical solution of boundary-value problems*. Elsevier.

[18] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. 2019. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4852–4861.

[19] Zhezhi He and Deliang Fan. 2019. Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11438–11446.

[20] Weiwen Jiang, Jinjun Xiong, and Yiyu Shi. 2021. A co-design framework of neural networks and quantum circuits towards quantum advantage. *Nature communications* 12, 1 (2021), 1–13.

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

[22] Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi. 2020. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 4154–4165.

[23] Weiwen Jiang, Xinyi Zhang, Edwin H-M Sha, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. 2019. Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[24] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. 2019. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4350–4359.

[25] Sami Khairy, Ruslan Shaydulin, Lukasz Cincio, Yuri Alexeev, and Prasanna Balaprakash. 2020. Learning to optimize variational quantum circuits to solve combinatorial problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 2367–2375.

[26] Ryan LaRose and Brian Coyle. 2020. Robust data encodings for quantum classifiers. *Physical Review A* 102, 3 (2020), 032420.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[28] Zhiding Liang, Zhepeng Wang, Junhuan Yang, Lei Yang, Yiyu Shi, and Weiwen Jiang. 2021. Can Noise on Qubits Be Learned in Quantum Neural Network? A Case Study on QuantumFlow. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–7.

[29] Alexander I Lvovsky, Barry C Sanders, and Wolfgang Tittel. 2009. Optical quantum memory. *Nature photonics* 3, 12 (2009), 706–714.

[30] Prasanna Ravi, Suman Deb, Anubhab Baksi, Anupam Chattopadhyay, Shivam Bhasin, and Avi Mendelson. 2021. On Threat of Hardware Trojan to Post-Quantum Lattice-Based Schemes: A Key Recovery Attack on SABER and Beyond. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 81–103.

[31] Runbin Shi, Peiyan Dong, Tong Geng, Yuhao Ding, Xiaolong Ma, Hayden K-H So, Martin Herbordt, Ang Li, and Yanzhi Wang. 2020. Csb-rnn: A faster-than-realtime rnn acceleration framework with compressed structured blocks. In *Proceedings of the 34th ACM International Conference on Supercomputing*. 1–12.

[32] Arvind Srinivasan, Kamal Chaudhary, and Ernest S Kuh. 1992. RITUAL: A performance driven placement algorithm. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 39, 11 (1992), 825–840.

[33] Davide Venturelli, Minh Do, Bryan O'Gorman, Jeremy Frank, Eleanor Rieffel, Kyle EC Booth, Thanh Nguyen, Parvathi Narayan, and Sasha Nanda. 2019. Quantum circuit compilation: An emerging application for automated reasoning. (2019).

[34] Hanrui Wang, Yongshan Ding, Jiaqi Gu, Zirui Li, Yujun Lin, David Z Pan, Frederic T Chong, and Song Han. 2021. Quantumnas: Noise-adaptive search for robust quantum circuits. *arXiv preprint arXiv:2107.10845* (2021).

[35] Zhepeng Wang, Zhiding Liang, Shanglin Zhou, Caiwen Ding, Yiyu Shi, and Weiwen Jiang. 2021. Exploration of Quantum Neural Architecture by Mixing Quantum Neuron Designs. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–7.

[36] Robert Wille and Rolf Drechsler. 2021. Introduction to the Special Issue on Design Automation for Quantum Computing. , 2 pages.

[37] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[38] Shilu Yan, Hongsheng Qi, and Wei Cui. 2020. Nonlinear quantum neuron: A fundamental building block for quantum neural networks. *Physical Review A* 102, 5 (2020), 052421.

[39] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 184–199.