MIT Open Access Articles

QOC: Quantum On-Chip Training with Parameter Shift and Gradient Pruning

Massachusetts Institute of Technology

QOC: Quantum On-Chip Training with Parameter Shift and Gradient Pruning

[1]Hanrui Wang*, [2]Zirui Li*, [3]Jiaqi Gu, [4]Yongshan Ding, [3]David Z. Pan, [1]Song Han

[1]Massachusetts Institute of Technology, [2]Shanghai Jiao Tong University, [3]University of Taxes at Austin, [4]Yale University

https://qmlsys.mit.edu

ABSTRACT

Parameterized Quantum Circuits (PQC) are drawing increasing research interest thanks to its potential to achieve quantum advantages on near-term Noisy Intermediate Scale Quantum (NISQ) hardware. In order to achieve *scalable* PQC learning, the training process needs to be offloaded to real quantum machines instead of using exponential-cost classical simulators. One common approach to obtain PQC gradients is *parameter shift* whose cost scales *linearly* with the number of qubits. We present QOC, the first experimental demonstration of practical on-chip PQC training with parameter shift. Nevertheless, we find that due to the significant quantum errors (noises) on real machines, gradients obtained from naïve parameter shift have low fidelity and thus degrading the training accuracy. To this end, we further propose *probabilistic gradient pruning* to firstly identify gradients with potentially large errors and then remove them. Specifically, small gradients have larger relative errors than large ones, thus having a higher probability to be pruned. We perform extensive experiments with the Quantum Neural Network (QNN) benchmarks on 5 classification tasks using 5 real quantum machines. The results demonstrate that our on-chip training achieves over **90%** and **60%** accuracy for 2-class and 4-class image classification tasks. The probabilistic gradient pruning brings up to **7%** PQC accuracy improvements over no pruning. Overall, we successfully obtain similar on-chip training accuracy compared with noise-free simulation but have much better training *scalability*. The QOC code is available in the TorchQuantum library.

1 INTRODUCTION

Quantum Computing (QC) has great potential to achieve exponential acceleration over classical computers, which represents a computational paradigm shift in various domains. Parameterized Quantum Circuits (PQC) are circuits containing trainable weights and are promising to achieve quantum advantages in current devices. Among them, Quantum Neural Network (QNN) is one of the popular algorithms for machine learning tasks.

In order to achieve PQC quantum advantage, the number of qubit needs to be large enough, which casts great difficulty in the parameter training process. In existing PQC work [4, 11], the primary focus has been building quantum models that can outperform classical model accuracy. Thus they typically perform training on classical computers through software simulations and then perform inference with simulators as well (Figure 1 top). Although classical simulation is useful in understanding the capabilities of small-size PQC, it is not
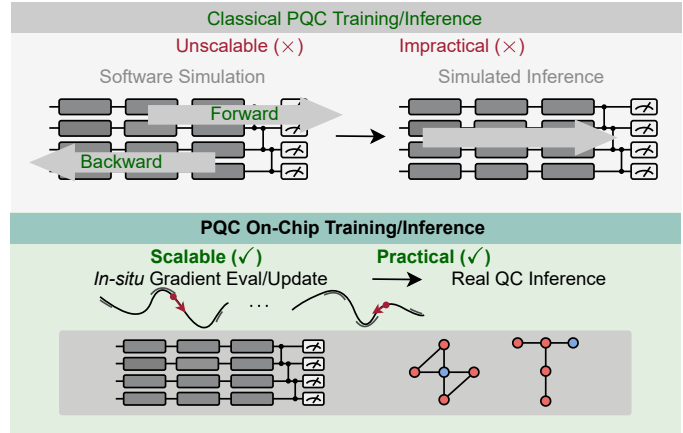
Figure 1: In QOC, PQC training and inference are both performed on *real quantum machines*, making the whole pipeline *scalable and practical*.

scalable due to the exponentially increased time and memory costs ($O(2^n)$, n is the qubit number). As shown in Figure 2(a), the space (#Regs) and time (#Ops) complexity of classical simulation grow exponentially as the number of qubits increases. To the authors' knowledge, this is the *first experimental demonstration* of efficient and scalable PQC on-chip training protocol. The optimization of parametrized quantum gates is offloaded to the quantum chips with *in-situ* gradient computation using *parameter shift* [14]. We also perform PQC evaluation on real quantum machines, making the results more *practical* as in Figure 1 bottom.

One of the major challenges to enable scalable and efficient PQC on-chip learning is the robustness against quantum noise. In the current Noisy Intermediate Scale Quantum (NISQ) [16] era, the gate error rates on real quantum devices are non-negligible (10^{-3} to 10^{-2}). In the context of PQC, such errors will lead to *noisy gradients* which can slow down convergence or even make training unstable. As shown in Figure 2(b), large gaps exist between the quantum on-chip training results and the classical noise-free simulation results.

By carefully investigating the on-chip training process, we observe that small gradients tend to have large relative variations or even wrong directions under quantum noises, as shown in Figure 2(c). Also, not all gradient computations are necessary for the training process, especially for small-magnitude gradients. Those observations provide great opportunities for us to boost the robustness and efficiency of PQC on-chip learning. Inspired by that, we propose a *probabilistic gradient pruning* method to predict and only compute gradients of high reliability. Hence we can reduce noise impact and also save the required number of circuit runs on real quantum machines. In this paper, we are mainly using *QNNs as benchmarks* but the techniques can also be applied to *other PQCs* such as Variational Quantum Eigensolver (VQE). QOC has following contributions:
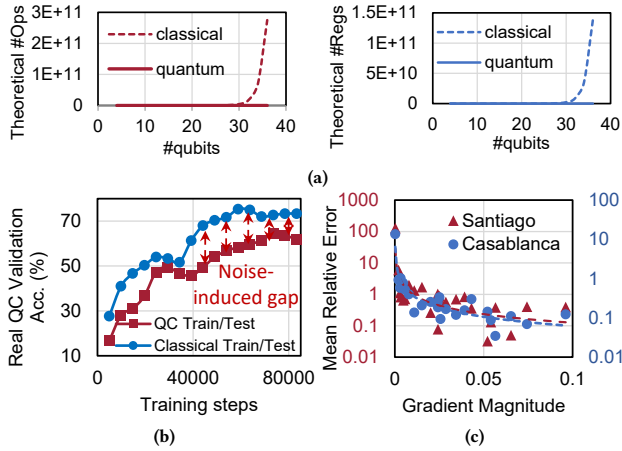
Figure 2: (a) Classical simulation has unscalable computational and memory costs. (b) Noises create significant accuracy gaps between PQC (QNN) classical simulation and on-chip training. (c) Small gradients suffer from larger relative errors, thus being less reliable.

- We are the *first work* to demonstrate the practicality of parameter shift on *NISQ machines*, achieving high PQC learning accuracy.
- A probabilistic gradient pruning method is proposed to improve the noise robustness by 5-7% and reduce the number of inference on real QC by 2× while maintaining the accuracy.
- Experimental deployment of QNN on 5 real quantum machines demonstrates that the proposed method can achieve over **90%** and **60%** accuracy for 2-class and 4-class image recognition tasks. Our framework enables scalable, robust, and efficient training of PQCs with large number of qubits and parameters.
- We open-source the parameter shift on-chip PQC training and gradient pruning code in the TorchQuantum library.

2 BACKGROUND

Quantum basics. Quantum circuits use quantum bit (called *qubit*) to store information, which is a linear combination of two basis states: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, for $\alpha, \beta \in \mathbb{C}$, satisfying $|\alpha|^2 + |\beta|^2 = 1$. An n-qubit system can represent a linear combination of 2^n basis states. A 2^n-length complex statevector of all combination coefficients is used to describe the quantum state. To perform computation on a quantum system, a sequence of parametrized quantum gates are applied to perform unitary transformation on the statevector, i.e., $|\psi(x, \theta)\rangle = \cdots U_2(x, \theta_2)U_1(x, \theta_1)|0\rangle$, where x is the input data, and $\theta = (\theta_1, \theta_2, \dots)$ are trainable parameters in quantum gates. In this way, input data and trainable parameters are embedded in the quantum state $|\psi(x, \theta)\rangle$. The computation results are obtained by qubit readout which measures the probability of a qubit state $|\psi\rangle$ collapsing to either $|0\rangle$ (i.e., output $y = +1$) or $|1\rangle$ (i.e., output $y = -1$) according to $|\alpha|^2$ and $|\beta|^2$. With sufficient samples, we can compute the expectation value: $\mathbb{E}[y] = (+1)|\alpha|^2 + (-1)|\beta|^2$. A non-linear network can be constructed to perform ML tasks by cascading multiple blocks of quantum gates and measurements.

Quantum noise. In real quantum computer systems, errors (noises) would occur due to unwanted interactions between qubits, imperfect control signals, or interference from the environment [6]. For example, quantum gates introduce *operation errors* (e.g., coherent errors and stochastic errors) into the system, and qubits also suffer from *decoherence error* (spontaneous loss of its stored information)
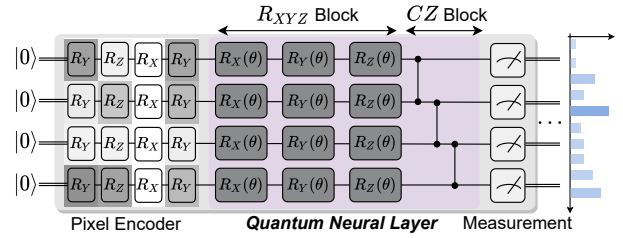

Figure 3: Quantum Neural Network (QNN) architecture.

over time. These noisy systems need to be characterized [13] and calibrated [7] frequently to mitigate the noise impact.

Quantum neural networks. Quantum Machine Learning (QML) [1, 9, 17, 18, 21] aims to leverage QC techniques to solve machine learning tasks and achieve much higher efficiency. The path to *quantum advantage* on QML is typically provided by the quantum circuit's ability to generate and estimate highly complex kernels [5], which would otherwise be intractable to compute with conventional computers. They have been shown to have potential speed-up over classical counterparts in various tasks, including metric learning [12], data analysis [10]. As shown in Figure 3, the quantum neural network is one type of QML model using variational quantum circuits with trainable parameters to accomplish feature encoding of input data and perform complex-valued linear transformations thereafter. Most of QNN trainings are exploratory and rely on classical simulation of small quantum systems. In our work, on the contrary, we explore the practical setting: the QNN training and inference are both performed on real quantum devices.

Pruning. Pruning techniques are widely used in the field of DNN [3, 19, 20, 23], performing an important role of the trade-off between accuracy and memory or time cost [15]. Recently, pruning techniques have been used in quantum tasks. Pruning the ansatz can bring time-efficient circuit and even higher performance on real QC [17]. In our work, we apply pruning techniques to prune unreliable gradients in order to mitigate the noise during training.

3 METHODOLOGY

To enable PQC on-chip learning, we first introduce an *in-situ* quantum gradient computation via parameter shift and its real QC implementation. A probabilistic gradient pruning method is proposed to save the gradient computation cost with enhanced noise-robustness and training efficiency. We study QNN as the benchmark PQC.

3.1 Parameter Shift Rule for Quantum Gradients

Parameter shift rule states that we can calculate the gradient of each parameter in some quantum circuits by simply shifting the parameter twice and calculating the difference between two outputs, without changing the structure of circuits or using any ancilla qubits. Prior works elaborate it based on quantum circuit function [2], however, in the next subsection we will show how parameter shift rules combined with backpropagation can be used in a real PQC task. Suppose an m-qubit quantum circuit is parametrized by n parameters $\theta = [\theta_1, \cdots, \theta_i, \cdots, \theta_n]$, the expectation value of measurements of this circuit can be represented by a **circuit function**,

$$f(\theta) = \langle\psi|U(\theta_i)^\dagger \widehat{Q} U(\theta_i)|\psi\rangle, \quad f(\theta) \in \mathbb{R}^m, \theta \in \mathbb{R}^n. \quad (1)$$

where θ_i is the scalar parameter whose gradient is to be calculated, and $U(\theta_i)$ is the gate where θ_i lies in. Here, for notation simplicity, we have already absorbed the unitaries before $U(\theta_i)$ into $\langle\psi|, |\psi\rangle$. Unitaries after $U(\theta_i)$ and observables are fused into $\widehat{Q}$. Usually, the

gates used in PQC can be written in the form $U(\theta_i) = e^{-\frac{i}{2}\theta_i H}$. Here H is the Hermitian generator of U with only 2 unique eigenvalues +1 and -1 (H's eigenvalues can be $\pm r$, but for simplicity we assume it's ± 1). In this way, the gradients of the circuit function f with respect to θ_i are,

$$\frac{\partial f(\theta)}{\partial \theta_i} = \frac{1}{2}\Big(f(\theta_+) - f(\theta_-)\Big),$$
$$\theta_+ = [\theta_1, \cdots, \theta_i + \frac{\pi}{2}, \cdots, \theta_n], \theta_- = [\theta_1, \cdots, \theta_i - \frac{\pi}{2}, \cdots, \theta_n], \quad (2)$$

where θ_+ and θ_- are the *positive shift* and *negative shift* of θ. Note that this parameter shift rule is *fundamentally different* from any numerical difference methods that only approximate the directional derivatives. Instead, Eq. 2 calculates the *exact* gradient w.r.t θ_i without any approximation errors or numerical issues.

We apply `softmax` on the expectation values of measurements $f(\theta)$ as the predicted probability for each class. Then we calculate the cross entropy between the predicted probability distribution p and the target distribution t as the classification loss $\mathcal{L}$,

$$\mathcal{L}(\theta) = -t^T \cdot \text{softmax}(f(\theta)) = -\sum_{j=1}^m t_j \log p_j, \quad p_j = \frac{e^{f_j(\theta)}}{\sum_{j=1}^m e^{f_j(\theta)}}. \quad (3)$$

Then the gradient of the loss function with respect to θ_i is $\frac{\partial \mathcal{L}(\theta)}{\partial \theta_i} = \left(\frac{\partial \mathcal{L}(\theta)}{\partial f(\theta)}\right)^T \frac{\partial f(\theta)}{\partial \theta_i}$.

Here $\frac{\partial f(\theta)}{\partial \theta_i}$ can be calculated on real quantum circuit by the parameter shift rule, and $\frac{\partial \mathcal{L}(\theta)}{\partial f(\theta)}$ can be efficiently calculated on classical devices using backpropagation supported by automatic differentiation frameworks, e.g., PyTorch and TensorFlow.

Now we derive the parameter shift rule used in our PQC models. Assume $U(\theta_i) = R_X(\theta_i), R_X(\alpha) = e^{-\frac{i}{2}\alpha X}$, where X is the Pauli-X matrix.

Firstly, the RX gate is,

$$R_X(\alpha) = e^{-\frac{i}{2}\alpha X} = \sum_{k=0}^{\infty} (-i\alpha/2)^k X^k/k!$$
$$= \sum_{k=0}^{\infty} (-i\alpha/2)^{2k} X^{2k}/(2k)! + \sum_{k=0}^{\infty} (-i\alpha/2)^{2k+1} X^{2k+1}/(2k+1)!$$
$$= \sum_{k=0}^{\infty} (-1)^k (\alpha/2)^{2k} I/(2k)! - i \sum_{k=0}^{\infty} (-1)^k (\alpha/2)^{2k+1} X/(2k+1)!$$
$$= \cos(\alpha/2)I - i\sin(\alpha/2)X. \quad (4)$$

Let $\alpha = \frac{\pi}{2}, R_X(\pm\frac{\pi}{2}) = \frac{1}{\sqrt{2}}(I \mp iX)$.

As $f(\theta) = \langle\psi|R_X(\theta_i)^\dagger \widehat{Q} R_X(\theta_i)|\psi\rangle$, $R_X(\alpha)R_X(\beta) = R_X(\alpha + \beta)$, and $\frac{\partial}{\partial\alpha}R_X(\alpha) = -\frac{i}{2}XR_X(\alpha)$, we have

$$\frac{\partial f(\theta)}{\partial \theta_i} = \langle\psi|R_X(\theta_i)^\dagger(-\frac{i}{2}X)^\dagger\widehat{Q}R_X(\theta_i)|\psi\rangle + \langle\psi|R_X(\theta_i)^\dagger\widehat{Q}(-\frac{i}{2}X)R_X(\theta_i)|\psi\rangle$$
$$= \frac{1}{4}(\langle\psi|R_X(\theta_i)^\dagger(I - iX)^\dagger\widehat{Q}(I - iX)R_X(\theta_i)|\psi\rangle$$
$$- \langle\psi|R_X(\theta_i)^\dagger(I + iX)^\dagger\widehat{Q}(I + iX)R_X(\theta_i)|\psi\rangle)$$
$$= \frac{1}{2}(\langle\psi|R_X(\theta_i)^\dagger R_X(\frac{\pi}{2})^\dagger\widehat{Q}R_X(\frac{\pi}{2})R_X(\theta_i)|\psi\rangle$$
$$- \langle\psi|R_X(\theta_i)^\dagger R_X(-\frac{\pi}{2})^\dagger\widehat{Q}R_X(-\frac{\pi}{2})R_X(\theta_i)|\psi\rangle)$$
$$= \frac{1}{2}(f(\theta_+) - f(\theta_-)). \quad (5)$$

Without loss of generality, the derivation holds for all unitaries of the form $e^{-\frac{i}{2}\alpha H}$, e.g., RX, RY, RZ, XX, YY, ZZ, where H is a Hermitian matrix with only 2 unique eigenvalues +1 and -1.
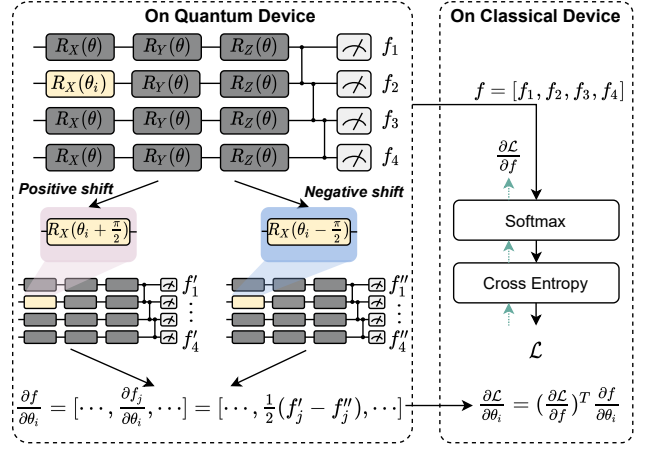

Figure 4: Quantum gradient calculation using the parameter shift rule on real quantum devices.

In our circuit functions, we assume each parameter lies in exactly one gate. However, there are cases that one parameter lies in multiple gates. In that case, we only need to calculate the gradient of the parameter in those gates separately and sum the gradients up to get the gradient of that parameter.

3.2 *In-situ* Gradient Computation on Real QC

To realize PQC on-chip learning, we implement a TrainingEngine, described in Alg. 1. This TrainingEngine contains three parts.

Jacobian calculation via parameter shift. In the first part, we sample a mini-batch of training data $\mathcal{I}$ in Line 6. For each example of the mini-batch, we set up the quantum encoder gates and then iteratively evaluate gradients for all parameters. In each iteration, we shift the parameter θ_i twice by $+\pi/2$ and $-\pi/2$ respectively. After each shift, we execute the shifted circuit on quantum hardware. The circuit will be *created, validated, queued, and finally run on real quantum machines*. As soon as we get the returned results of the two shifted circuits, i.e., $f(\theta_+)$ and $f(\theta_-)$, we apply Eq. 2 to obtain the upstream gradient $\frac{\partial f(\theta)}{\partial \theta_i}$, illustrated in the left part of Figure 4. Finally, we obtain the Jacobian matrix $\frac{\partial f(\theta)}{\partial \theta}$.

Down-stream gradient backpropagation. In the second part, we run the circuit without shift and get the measurement result $f(\theta)$. Then we apply `softmax` and cross-entropy function to the measured logits. In the end, we get the training loss $\mathcal{L}(\theta)$. Then we run backpropagation *only from the loss to the logits* to get the down-stream gradients $\frac{\partial \mathcal{L}(\theta)}{\partial f(\theta)}$, shown in the right part of Figure 4.

Gradient calculation. In the third part, we calculate the dot-product between down-stream gradients and the Jacobian and get the final gradients $\frac{\partial \mathcal{L}(\theta)}{\partial \theta} = \left(\frac{\partial f(\theta)}{\partial \theta}\right)^T \frac{\partial \mathcal{L}(\theta)}{\partial f(\theta)}$.

3.3 Probabilistic Quantum Gradient Pruning

On quantum chips, there exist various noises and errors that could potentially diminish the fidelity of the computation results. When the gradient magnitude is small, noises could easily overwhelm the signals, such that the *gradients calculated on real quantum circuit become unreliable when they have small magnitude*. Those unreliable gradients have harmful effects on training convergence. Skipping the evaluation on those unreliable gradients can benefit both training convergence and efficiency. Besides, we observe that for most parameters, if the gradient magnitudes are far from zero for several
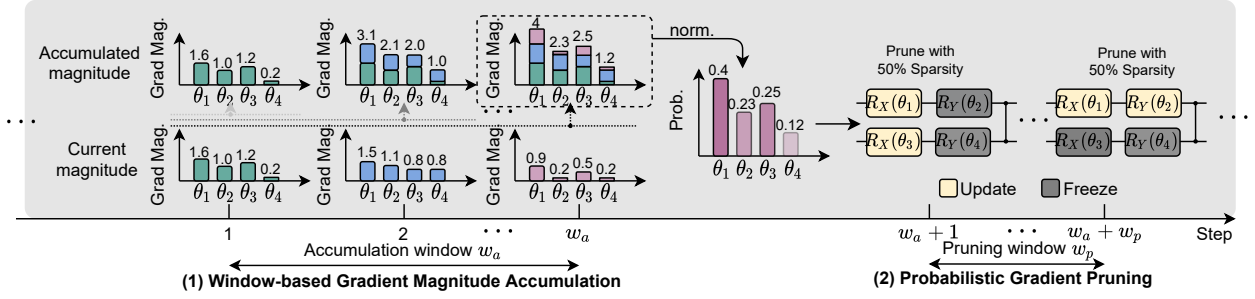
Figure 5: Efficient on-chip quantum gradient calculation with probabilistic gradient pruning. Gradient magnitudes are accumulated within the accumulation window and used as the sampling distribution. Based on the distribution, gradients are probabilistically pruned with a ratio r in the pruning window to mitigate noises and stabilize training.

steps, it will likely keep far from zero in the next several steps. Similarly, if the gradient magnitude remains small for some steps, it will likely keep small in the next several steps. This means the *gradient reliability is predictable* to some extent. Therefore, we propose the gradient pruning method to sample the parameters whose gradients are more reliable. This method helps training converge faster while also saving time by skipping the evaluation of unreliable gradients.

Alg. 1 describes the PQC on-chip training flow with probabilistic gradient pruning. We divide all the training steps into S stages and perform the pruning method periodically on each stage. For every stage, we split it into two phases, shown in Figure 5. The first phase is called *magnitude accumulation* with an accumulation window width w_a, and the second is called *probabilistic gradient pruning* (PGP) with a pruning window width w_p. We only apply pruning in the second phase, while the parameter subset is sampled from a probability distribution $\tilde{\theta} = \{\theta_i \sim P_M(\theta)|1 \le i \le (1-r)n\}$ based on the gradient information collected within the accumulation window.

In Lines 4-9, within the accumulation window, we record the magnitude of gradients of each parameter in each step and accumulate them until the window is over. At the end of the first phase, we can get an accumulator M that records the accumulated gradient magnitude for each parameter. Thus, when the pruning phase starts, we normalize the accumulated gradient magnitude and pass it to our sampler as the sampling distribution. In each pruning step, the sampler samples a subset of parameters $\tilde{\theta}$ with a pruning ratio of r, and we only evaluate gradients for them while the rest $\theta \backslash \tilde{\theta}$ is temporarily frozen.

There are three important hyper-parameters in our gradient pruning method: 1) accumulation window width w_a, 2) pruning ratio r, and 3) pruning window width w_p. The accumulation window width and pruning window width decide the reliability of the gradient trend evaluation and our confidence in it, respectively. The pruning ratio can be tuned to balance the gradient variances caused by noise perturbation and pruning. Thus, the percentage of the time saved by our probabilistic gradient pruning method is $r \frac{w_p}{w_a+w_p} \times 100\%$. In our experiments, we find that the setting (w_a=1, w_p=2~3, r=0.3~0.5) usually works well in all cases.

4 EXPERIMENTS

In this section, we deploy our PQC on-chip learning framework on real QC and evaluate it on 5 QNN tasks for image and vowel recognition. Compared with classical QNN training protocols, we can

---

Algorithm 1: PQC On-Chip Training with Probabilistic Gradient Pruning

Input : Accumulation window width w_a, gradient pruning ratio r, pruning window width w_p, training objective $\mathcal{L}$, initial parameters $\theta^0 \in \mathbb{R}^n$, training data $\mathcal{D}_{trn}$, initial step size η^0, and total stages S.

$\theta \leftarrow \theta^0, \eta \leftarrow \eta^0 \ t \leftarrow 0;$

for $s = 1, 2, \cdots, S$ **do**
  Initialize gradient magnitude accumulator $M \leftarrow \mathbf{0}^n$;
  for $\tau_a = 1, 2, \cdots, w_a$ **do**
    $t \leftarrow t + 1;$
    Sample a mini-batch $\mathcal{I} \sim \mathcal{D}_{trn}$;
    In-situ gradient evaluation via parameter shift
    $\nabla_\theta \mathcal{L}_\mathcal{I}(\theta) = \frac{1}{2}(\frac{\partial f(\theta)}{\partial \theta})^T \frac{\partial \mathcal{L}(\theta)}{f(\theta)};$
    Parameter update: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_\mathcal{I}(\theta);$
    Update magnitude accumulator $M \leftarrow$
    $M + |\nabla_\theta \mathcal{L}_\mathcal{I}(\theta)|;$
  for $\tau_p \leftarrow 1, 2, \cdots, w_p$ **do**
    $t \leftarrow t + 1;$
    Sample a mini-batch $\mathcal{I} \sim \mathcal{D}_{trn}$;
    Sample a subset with a ratio r based on accumulated gradient magnitude:
    $\tilde{\theta} = \{\theta_i \sim P_M(\theta)|1 \le i \le (1-r)n\};$
    $\tilde{\theta} \leftarrow \tilde{\theta} - \eta \nabla_{\tilde{\theta}} \mathcal{L}_\mathcal{I}(\theta);$

Output: Converged parameters θ

---

achieve 2-4% real QC test accuracy improvement with 2× convergence speedup. We also conduct extensive ablation studies to validate our scalability and the effectiveness of the proposed probabilistic gradient pruning method.

4.1 Experiment Setups

Benchmarks. We conduct our experiments on 5 QML tasks. QML are all classification tasks including MNIST [8] 4-class (0, 1, 2, 3), 2-class (3 and 6); Fashion [22] 4-class (t-shirt/top, trouser, pullover, dress), 2-class (dress and shirt); Vowel 4-class(hid, hId, had, hOd). MNIST and Fashion 2-class use the front 500 images as the training set and randomly sampled 300 images as the validation set. MNIST, Fashion 4-class uses the front 100 images as the training set and also randomly sampled 300 images as the validation set. The input images are all 28×28. We firstly center-crop them to 24×24 and

Table 1: Accuracy comparison among different settings. "Simu." represents "simulation".

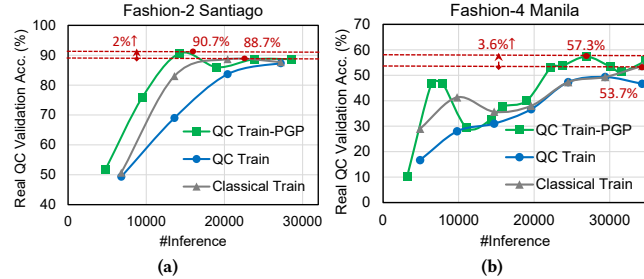| Method	Acc.	MNIST-4 Jarkata	MNIST-2 Jarkata	Fashion-4 Manila	Fashion-2 Santiago	Vowel-4 Lima
Classical-Train	Simu.	0.61	0.88	0.73	0.89	0.37
Classical-Train		0.59	0.79	0.54	0.89	0.31
QC-Train	QC	0.59	0.83	0.49	0.84	0.34
QC-Train-PGP		**0.64**	**0.86**	**0.57**	**0.91**	**0.36**

Figure 6: Real QC validation accuracy curves on different datasets and different quantum devices.

then down-sample them to 4×4 for MNIST and Fashion 2 and 4-class tasks. Vowel 4-class uses the front 100 samples as the training set and randomly sampled 300 samples as the validation set. For each sample, we perform principal component analysis (PCA) for the vowel features and take the 10 most significant dimensions.

All the tasks use four logical qubits. To embed classical image and vowel features to the quantum states, we first flatten them and then encode them with rotation gates. For down-sampled 4×4 images, we use 4RY, 4RZ, 4RX, and 4RY gates as the encoder. We put the 16 classical input values to the phases of 16 rotation gates, respectively. Therefore we can encode the classical values to quantum states. For 10 vowel features, we use 4RY, 4RZ, and 2RX gates for encoding.

The encoding gates are our hand-designed circuits. Our circuits are composed of several layers. There are 7 kinds of layers used to construct our circuits. (*i*) RX layer: Add RX gates to all wires; (*ii*) RY layer: same structure as in RX layer; (*iii*) RZ layer: same structure as in RX layer; (*iv*) RZZ layer: add RZZ gates to all logical adjacent wires and the logical farthest wires to form a ring connection, for example, an RZZ layer in a 4-qubit circuit contains 4 RZZ gates which lie on wires 1 and 2, 2 and 3, 3 and 4, 4 and 1; (*v*) RXX layer: same structure as in RZZ layer; (*vi*) RZX layer: same structure as in RZZ layer; (*vii*) CZ layer: add CZ gates to all logical adjacent wires.

For MNIST and Fashion 2-class tasks, the circuit contains 1 RZZ layer followed by 1 RY layer. For MNIST 4-class task, the circuit contains 3 RX+RY+RZ+CZ layers (1 RX layer, 1 RY layer, 1 RZ layer, and 1 CZ layer in series). For Fashion 4-class task, the circuit contains 3 RZZ+RY layers (1 RZZ layer followed by 1 RY layer). For Vowel 4-class task, the circuit contains 2 RZZ+RXX layers (1 RZZ layer followed by 1 RXX layer).

For the output of our quantum circuits, we measure the expectation values on Pauli-Z basis and obtain a value [-1, 1] from each qubit. For 2-class, we sum the qubit 0 and 1, 2, and 3 respectively to get 2 output values. For 4-class, we just use the four expectation values as 4 output values. Then we process the output values by Softmax to get probabilities.

Quantum devices and compiler configurations. We use IBM quantum computers via qiskit API [7] to submit our circuits to real superconducting quantum devices and achieve quantum on-chip training. We set all the circuits to run 1024 shots.
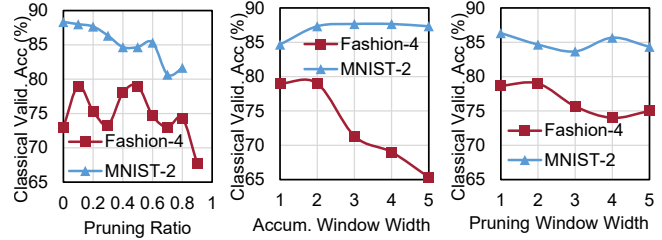

Figure 7: Ablation on pruning ratio, accumulation window width, and pruning window width.

Baseline. We have two baselines. (1) QC-Train: We train our model without gradient pruning, i.e., calculating gradients of every parameter in each step. The gradient calculation is deployed on real quantum circuits. (2) Classical-Train: We train our QNN model completely on classical computers. We use a vector to record the amplitudes of the quantum state, utilize complex matrix multiplication to simulate quantum gates, and sample based on the amplitude vector to simulate quantum measurement.

The QC-Train-PGP line shows training on real quantum circuits while applying our probabilistic gradient pruning. In all the cases, we adopt accumulation window size 1, pruning ratio 0.5, and pruning window size 2, except for Fashion-4, we adopt pruning ratio 0.7, and other settings remain the same.

4.2 Main Results

QNN results. Table 1 shows the accuracy of comparison on 5 tasks. In each task, we show 4 accuracy values, which are (1) accuracy of Classical-Train tested on classical devices, (2) accuracy of Classical-Train tested on real quantum circuits; (3) accuracy of QC-Train tested on real quantum circuits; (4) accuracy of QC-Train-PGP tested on real quantum circuits. In each task, the accuracy is collected after finishing a certain number of circuit runs. We train and evaluate MNIST-2 and MNIST-2 on ibmq_jakarta, Fashion-4 on ibmq_manila, Fashion-2 on ibmq_santiago, and Vowel-4 on ibmq_lima.

The noise-free accuracy is usually the highest among the other three, because it represents the accuracy without any noise perturbation. The QC-Train-PGP usually takes second place because compared to Classical-Train, it has the advantage of noise awareness, and compared to QC-Train, it suffers *less* from noise thanks to gradient pruning.

Training curves. Figure 6 shows the real QC validation accuracy curve during training. The X-axis is the number of inferences (how many circuits have been run). The Y-axis is the accuracy of the validation dataset tested on real quantum circuits. MNIST 4-class runs on the ibmq_jakarta machine. We observe that given a fixed inference budget, our QC-Train-PGP achieves the best accuracy of 63.7% while the Classical-Train only achieves 59.3%.

We further train Fashion 2-class on ibmq_santiago. QC-Train-PGP only takes 13.9k inferences to reach the peak accuracy 90.7%, while the best accuracy Classical-Train can achieve is merely 88.7% at the cost of over 30k inferences.

4.3 Ablation Studies

Ablation on gradient pruning. In Figure 7, we evaluate the training performance with different pruning ratios r, accumulation window size w_a, and pruning window size w_p on Fashion-4 and MNIST-2 tasks. We find that the $r = 0.5$ is generally a good setting for our tasks. Overly large pruning ratios will induce too many gradient variances that harm the training convergence. For the accumulation window

Table 2: The proposed probabilistic pruning is better than deterministic pruning.

Method	MNIST-4	MNIST-2	Fashion-4	Fashion-2
Deterministic	0.61	0.82	0.72	0.89
Probabilistic	**0.62**	**0.85**	**0.79**	**0.90**

Table 3: Adam optimizer can outperform SGD and Momentum optimizers.

Optimizer	MNIST-4	MNIST-2	Fashion-4	Fashion-2
SGD	0.5	0.8	0.45	76
Momentum	0.55	0.83	0.66	0.90
Adam	**0.61**	**0.88**	**0.75**	**0.91**

size, $w_a = 1$ or 2 are suitable choices. When w_a is too large, the accumulated gradient magnitudes are similar among all parameters, leading to a nearly uniform sampling distribution. This will bring undifferentiated pruning, and the accuracy will drop as the Fashion-4 curve shows. The pruning window w_p should also not be too large. As w_p grows, the accumulated gradient magnitudes used to instruct our pruning become less reliable.

Discussion on scalability. Figure 8 shows the superior scalability of quantum on-chip training. Classical simulation runtime exponentially increases as #qubits scales up, while the runtime on real quantum machines scales nearly linearly to #qubits. The classical curve in Figure 8 represents runtime and memory cost of running 50 circuits of different #qubits with 16 rotation gates and 32 RZZ gates. The curve before 22 qubits is measured on a single NVIDIA RTX 2080 Ti GPU; points after 24 qubits are extrapolated. The quantum curve before 27 qubits is tested on ibmq_toronto; the points after 30 qubits are extrapolated.

We can observe clear quantum advantages on circuits with more than 27 qubits. In terms of memory cost, classical simulation consumes thousands of Gigabits for storage which is intractable. In contrast, on quantum machines, the information is stored in the quantum state of the circuit itself with negligible memory cost.

Probabilistic vs. deterministic gradient pruning. Our pruning is decided by a random sampler based on the accumulated gradient magnitude. We call this probabilistic pruning. If the sampler only samples the parameters with the biggest accumulated gradient magnitude, this is called deterministic pruning. We adopt probabilistic pruning instead of deterministic pruning because deterministic pruning limits the degree of freedom and increases the gradient sampling bias. Table 2 shows that deterministic pruning has 1%-7% accuracy loss compared with probabilistic pruning.

Different optimizers. Table 3 shows the accuracy tested on classical devices trained with different optimizers. The learning rate is controlled by a cosine scheduler from 0.3 in the beginning to 0.03 in the end. We test SGD, SGD with a momentum factor of 0.8, and Adam on MNIST-4, MNIST-2, Fashion-4, and Fashion-2, and found that Adam always performs the best. Hence, all the experiments are done using Adam optimizers by default.

5 CONCLUSION

In this work, for the first time, we present an efficient and robust on-chip training framework for PQC and demonstrate its effectiveness on real quantum devices. By leveraging parameter shift, we can calculate the exact quantum gradients directly on quantum machines, thus achieving high *scalability*. To alleviate the negative impact of
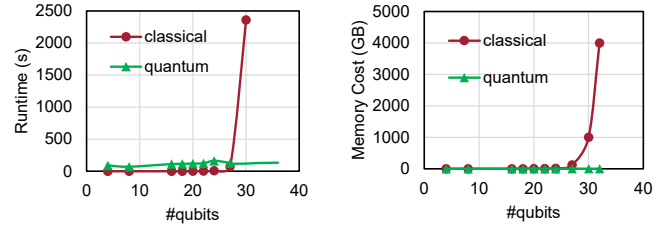

Figure 8: Runtime and memory cost comparison between classical simulation and quantum on-chip run.

quantum noises on gradients, we further propose the probabilistic gradient pruning technique to avoid updating parameters with unreliable gradients. Experimental results on 5 classification tasks and 5 machines demonstrate that QOC achieves comparable accuracy with noise-free simulations. We hope QOC can open an avenue towards practical training of large PQC models for quantum advantage.

REFERENCES

[1] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. 2017. Quantum machine learning. *Nature* 549, 7671 (2017).

[2] Gavin E Crooks. 2019. Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition. *arXiv:1905.13311* (2019).

[3] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[4] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. 2009. Quantum algorithm for linear systems of equations. *Physical review letters* 103, 15 (2009), 150502.

[5] Vojtěch Havlíček et al. 2019. Supervised learning with quantum-enhanced feature spaces. *Nature* 567, 7747 (2019), 209–212.

[6] Cheng-Yun Hsieh, Chen-Hung Wu, Chia-Hsien Huang, His-Sheng Goan, and James Chien Mo Li. 2020. Realistic fault models and fault simulation for quantum dot quantum circuits. In *2020 57th (DAC)*. IEEE, 1–6.

[7] Qiskit IBM. [n. d.].

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[9] 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 *ICCAD*. IEEE, 1–7.

[10] Seth Lloyd, Silvano Garnerone, and Paolo Zanardi. 2016. Quantum algorithms for topological and geometric analysis of data. *Nature communications* 7, 1 (2016).

[11] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. 2013. Quantum algorithms for supervised and unsupervised machine learning. *arXiv:1307.0411* (2013).

[12] Seth Lloyd, Maria Schuld, Aroosa Ijaz, Josh Izaac, and Nathan Killoran. 2020. Quantum embeddings for machine learning. *arXiv:2001.03622* (2020).

[13] Easwar Magesan, Jay M Gambetta, and Joseph Emerson. 2012. Characterizing quantum gates via randomized benchmarking. *Physical Review A* 85, 4 (2012).

[14] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. 2018. Quantum circuit learning. *Physical Review A* (2018).

[15] Le Thanh Nguyen-Meidine et al. 2020. Progressive Gradient Pruning for Classification, Detection and DomainAdaptation. arXiv:1906.08746 [cs.LG]

[16] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (2018), 79.

[17] Hanrui Wang, Yongshan Ding, Jiaqi Gu, Yujun Lin, David Z Pan, Frederic T Chong, and Song Han. 2022. QuantumNAS: Noise-adaptive search for robust quantum circuits. *HPCA* (2022).

[18] Hanrui Wang, Jiaqi Gu, Yongshan Ding, Zirui Li, Frederic T Chong, David Z Pan, and Song Han. 2022. QuantumNAT: Quantum Noise-Aware Training with Noise Injection, Quantization and Normalization. *DAC* (2022).

[19] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. SpAtten: Efficient sparse attention architecture with cascade token and head pruning. In *HPCA*. IEEE.

[20] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. 2020. Apq: Joint search for network architecture, pruning and quantization policy. In *CVPR*.

[21] Zhepeng Wang, Zhiding Liang, Shanglin Zhou, et al. 2021. Exploration of Quantum Neural Architecture by Mixing Quantum Neuron Designs. In *ICCAD*. IEEE.

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

[23] Zhekai Zhang, Hanrui Wang, Song Han, and William J Dally. 2020. SpArch: Efficient architecture for sparse matrix multiplication. In *HPCA*. IEEE.