Q-Forge: An Efficient Two-Stage Framework for Noise-Aware Quantum Model Training

Ngoc-Truong Nguyen[1,2], Huy-Tan Thai*[1,2]

[1]University of Information Technology, Ho Chi Minh City, Vietnam
[2]Vietnam National University, Ho Chi Minh City, Vietnam
Email: truongnn@uit.edu.vn, tanth@uit.edu.vn
Corresponding author: tanth@uit.edu.vn

Abstract—**Training quantum models on noisy devices is challenging due to the accumulation of errors from quantum noise and the high computational costs. Conventional training methods often require a large number of epochs to converge, thereby increasing the time required. In this work, we propose *Q-Forge* — a two-stage training framework that combines *pre-training* on an ideal environment (noise-free) with *fine-tuning* on noisy devices to address the above drawbacks. In particular, we develop the *Q-ACS* algorithm to automatically determine the optimal transition point once the model has converged on the ideal environment. This point marks the transition from the training process to a noise-aware phase, where the model is adapted to the realistic error characteristics of a specific quantum device. Experimental results on three typical quantum models ($SQNN$, VQC_1, and VQC_2) show that *Q-Forge* reduces training time by up to 70% compared to the direct training method, while maintaining or improving accuracy (6-8% increase for VQC_2). These results confirm the feasibility and "plug-and-play" capability of *Q-Forge*, opening up an efficient approach for training quantum models on real quantum devices.**

Index Terms—**Quantum noise; quantum training;**

I. Introduction

Quantum computing (QC) has the potential to replicate exponential speeds with classical computers, opening up a new computational paradigm in many fields such as chemistry, cryptography, and database searching [1]. In particular, Quantum Machine Learning (QML) has emerged as a transformative approach that integrates classical machine learning (ML) with quantum computation [2], [3]. While classical ML algorithms are increasingly challenged by data explosion and computational costs, QC introduces a novel processing paradigm grounded in quantum physical phenomena, promising the ability to address these challenges [4], [5]. One of the prominent applications of QML is quantum neural networks (QNNs), which exploit quantum properties such as superposition and entanglement. This approach provides several key advantages that classical models cannot easily achieve, including access to a vastly larger computational space, a greater ability to represent complex data, and the potential for a significant computational speedup [6].

Currently, deploying QML on quantum devices faces significant challenges due to the hardware limitations of Noisy Intermediate-Scale Quantum (NISQ) systems [5]. NISQ devices are constrained by a limited number of qubits, shallow allowable circuit depths, and a pronounced susceptibility to quantum noise, all of which severely degrade computational reliability [7]. Common sources of this quantum noise include unintended interactions with the environment, imperfections in quantum gates, and measurement errors, collectively leading to deviations in computational outcomes [8], [9]. These noise lead to the accuracy degradation of QML models on NISQ devices at up to 60-64% RoQNN [10], QuantumNAT [11], compared to their performance on ideal simulators (noise-free). Consequently, bridging the sim-to-real gap between noise-free simulators and practical NISQ devices is essential to ensure the utility and reliability of QML in real-world applications.

The research community has primarily pursued two directions to address quantum noise in the NISQ era. The first approach focuses on designing noise-resilient quantum circuit architectures, such as QuantumNAS [12], QuantumSEA [13], and QWAS [14], which aim to reduce the number of quantum gates and optimize qubit mapping to enhance robustness against noise. The primary limitation of this approach is that it sacrifices the model's expressivity and portability for a noise-resilient architecture that is computationally expensive to find and only optimal for a single hardware platform [15], [16]. The second approach retains the original circuit architecture but modifies training strategies or procedures to enable quantum models to better adapt to noise, as exemplified by Qust [17], RoQNN [10], and QuantumNAT [11]. RoQNN and QuantumNAT have demonstrated that QNNs trained on a realistic quantum emulator (RQE) achieve less than 5% accuracy deviation when deployed on NISQ machines , whereas models trained on noise-free emulators (ideal environments) can suffer up to 60–64% degradation, highlighting the effectiveness of RQE-based training in bridging the sim-to-real gap. Therefore, any quantum model can be trained on RQE prior to its deployment on physical quantum hardware. This preliminary training stage reduces the overall time and cost, while also improving the model's adaptation to the noisy conditions of real devices.

However, training directly on RQE is not an optimal strategy. Training in noisy environments makes the loss landscape rough and contains many bad local minima, slowing down the convergence process [18]–[20]. In addition, this emulator requires substantial computational resources and significantly extends training time, even though the model at early stages is primarily learning basic representations. To address these limitations, we propose *Q-Forge*, a two-stage training strategy inspired by curriculum learning [21] . In this approach, the model is first trained in an ideal environment to quickly learn basic features, and subsequently fine-tuned in a noisy environment to achieve robust performance on RQEs. In particular, we introduce an adaptive curriculum switching algorithm, Q-ACS, which leverages the stability of the loss function to automatically identify the optimal transition point τ between the two stages. In this work, we have made the following contributions:

- Propose a two-stage curriculum learning strategy for training QML models, in which the quantum model is pre-trained on an ideal simulator to learn feature representations, and subsequently fine-tuned on RQEs to adapt to realistic noisy conditions.
- Develop an adaptive algorithm to automatically determine the optimal transition point τ, thereby enabling the model to establish a robust knowledge foundation while also allocating sufficient time to learn effective strategies for handling noise. Consequently, the proposed approach not only enhances the final performance in terms of both accuracy and stability but also contributes to the automation of the overall training process.
- Design and implement the Q-Forge framework as a holistic solution that integrates the two-stage curriculum learning strategy with the transition-point detection algorithm, enabling QML models to be trained with lower computational cost and improved stability on RQEs, prior to deployment on real quantum devices.

The remainder of this paper is structured as follows. Section II reviews the related work on quantum machine learning and noise-resilient training methods. Section III introduces the problem formulation and the proposed Q-FORGE framework. Section IV presents the experimental setup, while Section V reports and discusses the results. Finally, Section VI concludes the paper and outlines future research directions.

II. Related works

A. Noise-Aware and Sim-to-Real Training Frameworks

The noise challenge in NISQ devices has motivated a series of studies to enhance the robustness of QMLs. For instance, Hu et al. [22] proposed QuCAD, a framework designed to enable noise-aware model management by constructing a model repository and facilitating efficient noise adaptation on real quantum hardware. In parallel, Wang et al. [1], [12] introduced an approach that combines the parameter-shift rule with stochastic gradient pruning to eliminate unreliable gradients during training. Similarly, Wang et al. published QuantumNAT [11] and RoQNN [10], which proposed a three-stage training procedure to improve the noise immunity through post-measurement normalization, noise injection, and quantization. This method has shown to be effective in using RQE to train quantum models, reducing the gap between simulation and reality to 5%.

B. Noise Simulation Techniques

IBM Qiskit's Fake Provider API [1] is a technical solution for a faithful RQE noise simulator that bridges the gap between sim-to-real quantum devices. In the [22] study, the authors also used the IBM Aer simulator to simulate noise over 146 days of continuous experiments, demonstrating stable and reliable noise reproduction. Recently, Rabelo et al. [23] exploited RQE as *Fake-Lagos*, *Fake-Kolkata*, and *Fake-Washington* backends to model noise during the execution of the QAOA algorithm on the Max-Cut problem. The results show that the error map snapshots from real quantum hardware provide an effective means to study the behavior of algorithms under realistic noise conditions.

C. Theoretical Foundations

Our approach is developed based on two theories, including curriculum learning [21] and domain adaptation [24]. In the context of QNNs, we define the training in a noise-free environment as the source domain (the easy program) and the training on a realistic quantum emulator as the target domain (the hard program). In our experiments, we found that attempting to train a QNN directly on a realistic quantum simulator from scratch is computationally expensive and time-consuming. Therefore, we propose pre-training the quantum model on the ideal simulator, then adjusting the learned parameters on RQE to perform well in noisy environments.

III. Methodology

A. Overview

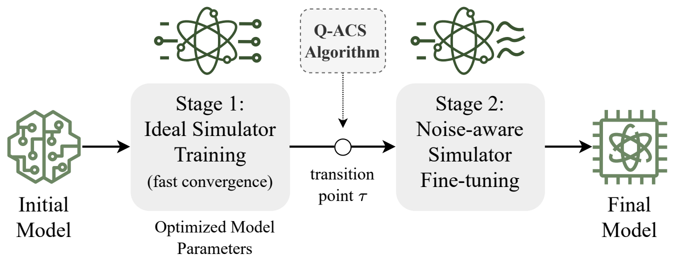
Q-ACS Algorithm
Stage 1: Ideal Simulator Training (fast convergence)
Stage 2: Noise-aware Simulator Fine-tuning
Initial Model
Optimized Model Parameters
transition point τ
Final Model

Fig. 1: The architecture of Q-Forge framework.

Figure 1 illustrates the overview of the *Q-Forge* framework, which consists of two consecutive training phases: (1) training in an ideal environment (noise-free) to initialize the initial parameters, and (2) fine-tuning on a noisy quantum device simulator (RQE) to adapt to real-world conditions. To automate the training process, we propose an algorithm termed *Q-ACS* (Q-Forge Adaptive Curriculum Switching), which determines the optimal switching point τ based on the stability of the loss function. This mechanism ensures that the model not only establishes a solid knowledge foundation but also has sufficient time to adapt to noise, thereby achieving both high and stable final performance.

[1]https://quantum.cloud.ibm.com/docs/en/api/qiskit-ibm-runtime/fake-provider

B. Two-Stage Training Curriculum

We first formulate the problem as follows: given a parameterized quantum neural circuit $V(\theta)$, we want to find θ^* such that the loss function $\mathcal{L}(V(\theta), \mathcal{D})$ is minimized when implemented on a real quantum device. Specifically, the loss function is defined as follows:

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \ell\big(y, \hat{y}(V(x;\theta))\big) \tag{1}$$

where $\mathcal{D}$ denotes the training dataset, x represents the input, y is the ground-truth label, $\hat{y}(V(x;\theta))$ is the output of the parameterized quantum neural circuit, and $\ell(\cdot,\cdot)$ refers to the local loss function (e.g., cross-entropy or mean squared error).

The training procedure in *Q-Forge* is divided into two complementary phases. In the first phase, the parameterized quantum model $V(\theta)$ is trained on an ideal simulator to obtain a good initialization for the subsequent training. The objective function is defined as the expectation value of an observation operator O:

$$f(\theta) = \langle 0|V^\dagger(\theta)\, O\, V(\theta)|0\rangle \tag{2}$$

where θ is the set of parameters to be trained. The derivative of this function is estimated using the *Parameter-Shift Rule (PSR)*:

$$\frac{\partial f(\theta)}{\partial \theta_i} = \frac{f\big(\theta + \frac{\pi}{2}\boldsymbol{e}_i\big) - f\big(\theta - \frac{\pi}{2}\boldsymbol{e}_i\big)}{2} \tag{3}$$

where $\boldsymbol{e}_i$ is the unit vector in the θ_i direction. This approach provides the correct gradient in the simulation environment, allowing to minimize the loss function $\mathcal{L}$ during training time τ, thereby obtaining an intermediate set of parameters:

$$\theta' = \arg\min_{\theta}\ \mathcal{L}_{\text{ideal}}\big(V(\theta)\big) \tag{4}$$

In the second stage, Q-Forge trains the V model on the noisy environment from the RQE. In this context, PSR is not feasible due to the current limitation when QNode is extended for broadcasted (vectorized) execution of circuits. Therefore, the PSR is replaced by the *finite-difference* method, where the derivative is approximated by:

$$\frac{\partial f(\theta)}{\partial \theta_i} \approx \frac{f(\theta + \delta\boldsymbol{e}_i) - f(\theta - \delta\boldsymbol{e}_i)}{2\delta} \tag{5}$$

with δ being a small, appropriately chosen noise. Although not as accurate as PSR in an ideal environment, this method still allows for fine-tuning of the intermediate parameter set θ' to adapt to the noise:

$$\theta^* = \arg\min_{\theta}\ \mathcal{L}_{\text{noise}}\big(V(\theta)\big) \tag{6}$$

C. Q-Forge Adaptive Curriculum Switching (Q-ACS) Algorithm

The detailed procedure of Algorithm 1 is described as follows. First, the model V is initialized with the initial parameter $\boldsymbol{\theta_0}$ and starts training in the Ideal Emulator. At each epoch e, the model V is updated according to the current parameter, and the loss function $\mathcal{L}(e)$ and the gradient norm

Algorithm 1: Q-ACS Algorithm

```
Input : Training dataset $\mathcal{D}_{train}$, validation dataset
        $\mathcal{D}_{val}$; Epochs $n$, number of wires $m$;
        Thresholds: $\epsilon_{\min}$ (min_improvement), $P$
        (patience), $\delta$ (grad_norm_threshold)
Output: Final trained $V$.
1  Initialize model $V$;
2  Initialize $\mathcal{L}^* \leftarrow \infty$, $E_{\text{no\_imp}} \leftarrow 0$, $\tau \leftarrow n$;
3  for $e \leftarrow 1$ to $n$ do
4      if $e \le \tau$ then
5          Train model $V$ on Ideal Emulator using
             $\mathcal{D}_{train}$; Compute $\mathcal{L}(e)$ and $\|\nabla\mathcal{L}(e)\|$;
6          $\Delta\mathcal{L} \leftarrow \mathcal{L}^* - \mathcal{L}(e)$;
7          if $\Delta\mathcal{L} > \epsilon_{\min}$ then
8              $\mathcal{L}^* \leftarrow \mathcal{L}(e)$;
9              $E_{\text{no\_imp}} \leftarrow 0$;
10         else
11             $E_{\text{no\_imp}} \leftarrow E_{\text{no\_imp}} + 1$;
12         end
13         if $E_{no\_imp} \ge P$ and $\|\nabla\mathcal{L}(e)\| < \delta$ then
14             $\tau \leftarrow e$;
15         end
16     else
17         Train model $V$ on RQE using $\mathcal{D}_{train}$;
18     end
19     Evaluate $V$ on $\mathcal{D}_{val}$ and update training history;
20 end
```

$|\nabla\mathcal{L}(e)|$ are computed. From lines 7–12, the algorithm uses a counter $E_{\text{no\_imp}}$ to track the number of consecutive epochs in which the model does not achieve a significant improvement. The improvement is evaluated by $\Delta\mathcal{L} = \mathcal{L}^* - \mathcal{L}(e)$, where $\mathcal{L}^*$ is the best observed loss value, and the threshold $\epsilon_{\min}$ is used to determine the minimum required improvement. From lines 13–14, if the counter $E_{\text{no\_imp}}$ exceeds the threshold P epoch and the gradient norm is less than the threshold δ, the model V is considered to have converged in the ideal stage, and the training process switches to the noisy environment. Finally, from lines 15–16, if this condition is never satisfied, the model continues training on the ideal environment until the end of all epochs n, and then $\tau = n$ is chosen as the default handover point.

IV. Experimental Setup

The experiments are implemented on the IBM Quantum noise simulation environment using the *Qiskit*[2] framework. These providers (e.g., *FakeVigo*, *FakeManila*) reproduce the real hardware characteristics, including quantum gate errors, qubit loss and disconnection times, as well as physical coupling maps. Based on the Qiskit API, we build a Python library that allows automatic recommendation and selection of appropriate backends based on the number of qubits (num wires) of the circuit. The training process is implemented using

[2]https://www.ibm.com/quantum/qiskit

the *PennyLane*[3] framework, which is capable of integrating Qiskit and PyTorch. All experiments were conducted on the Athens backend, and each circuit was executed for 1000 shots. This setup was chosen to be consistent with the standard operating conditions for deployment on real quantum devices. The RQEs are emulated on a classical computer with the following configuration: Intel 13th Gen Core i9-13900K CPU, 64 GB RAM, Ubuntu 2022 (64-bit) operating system. We train and test two types of models, including the variational quantum circuit (VQC) model [25], [26] and the hybrid quantum–classical model SQNN [5], to evaluate the effectiveness of the proposed method. We also compare the training process of the proposed two-stage quantum model strategy (Q-Forge) with direct training strategies (e.g., QuantumNAT, RobustQNN) in noisy environments (baseline) to evaluate the noise robustness of our proposed method. The experiments were performed on the *MNIST 2-class* dataset, for 50 epochs with a batch size of 64. The experimental setup and parameters are summarized in I. The technique of shortening the training time by early stopping [27] is not within the scope of this study.

TABLE I: Summary of the experimental setup and parameters.

Category	Parameter	Specification
Computational environment	Hardware	CPU: Intel Core i9-13900K RAM: 64 GB
	Software	OS: Ubuntu 22.04 LTS (64-bit) Frameworks: PyTorch
Quantum Backend Config	Frameworks	*Qiskit*, *PennyLane*
	Backend	Athens
	Execution	1000 shots
Machine Learning Setup	Models	VQC and SQNN
	Baseline	Direct training strategies
	Dataset	*MNIST 2-class*
	Parameters	Epochs: 50, Batch size: 64

Quantum Model: The VQC model is built on a 4-qubit circuit, consisting of three main components: (i) an *Amplitude Embedding* layer to encode the input data, (ii) a parametrization layer iterated from the VQC blocks, and (iii) a Pauli-Z measurement layer. Each VQC block is designed in a sequential structure, consisting of an R_Z gate, an RY gate, an additional R_Z gate, and a two-qubit CRX gate arranged in a ring configuration, to ensure full connectivity between the qubits. We implement two versions, VQC_1 and VQC_2, which differ in the number of VQC blocks used. Meanwhile, SQNN consists of two separate quantum circuits, each built on four qubits: *quantum feature extractor* and *quantum classifier*. Each feature extractor encodes the input data through the R_X gate, then exploits quantum correlations between features using the $IsingXX$, $IsingYY$, and $IsingZZ$ gates, before measuring the expectation on the readout qubit to generate intermediate features. These features are then fed into the quantum classifier, re-encoded using the R_X gate, and processed through an $IsingYY$ layer to link the data qubits to the measurement qubits. In both models, the measurement is performed on the first qubit using the Pauli-Z expectation operator, which maps the quantum output to a binary probability distribution p_0, p_1 for classification. Each parameter layer in the circuit is optimized simultaneously through the backpropagation algorithm.

[3]https://pennylane.ai/

Datasets: Similar to previous studies [1], [28], [29], we randomly select 3,000 samples as training set and 500 samples as testing set from two digit classes *0* and *1* of MNIST dataset. Our focus is on the Q-Forge framework's performance, rather than optimizing the classical pre-processing pipeline. To reduce the computational complexity of the variational quantum circuit (VQC) model, we apply the *Principal Component Analysis* (PCA) technique to reduce the spatial dimension of the image from 28×28 pixels to feature vectors of size 1×8 [25], [26]. These feature vectors are then used as inputs to the VQC model in both VQC_1 and VQC_2 models.

V. Evaluation results

A. Evaluation of the training process of the proposed method

A two-stage training strategy *Q-Forge* enables models to quickly find the optimal parameter region on a noisy environment (RQE emulator). Figure 2 shows the learning curves including the loss and accuracy values of three quantization models when trained with Q-Forge and baseline. The *pre-training* stage on the ideal device not only leads to a steady decrease in loss and a gradual increase in accuracy. At the same time, it provides an initialization advantage, allowing the model to narrow the optimal parameter region when transferring to the noisy device, thereby achieving faster convergence than direct training. Specifically, for the SQNN model, at $\tau = 25$, the model loss after one training epoch on Q-Forge's *RQE emulator* is roughly equivalent to the result of baseline after 11 epochs (about 0.595), while the accuracy reaches 87%. For VQC_1, after 41 epochs of *pre-training* on the ideal environment ($\tau = 42$), Q-Forge immediately achieves a loss of 0.58 and an accuracy of 80% on *RQE emulator*, which is equivalent to baseline in the same number of epochs. In particular, VQC_2 demonstrates superior performance. After transfer at epochs 36–50, the model trained with *Q-Forge* consistently achieves a loss at least 0.02 lower and an accuracy more than 6% higher than the baseline on the *RQE emulator*.

B. Analysis of training results of the proposed method

The summary results in Table II show that *Q-Forge* significantly shortens the training time compared to baseline. Specifically, for the SQNN model, the time is reduced by 49.0% (45h41m vs. 89h37m) while the accuracy is almost equivalent (train: 87.4% vs. 88.0%; test: 86.3% vs. 87.7%). For VQC_1, *Q-Forge* not only achieves higher accuracy on the test set (80.3% vs. 79.0%) but also significantly shortens the training time from 35h55m to 7h31m (a reduction of 79.1%). In particular, with VQC_2, *Q-Forge* outperforms by reducing the training time by 70.1% (from 91h23m to 27h21m), while achieving an accuracy on the test set that is 8% higher than the

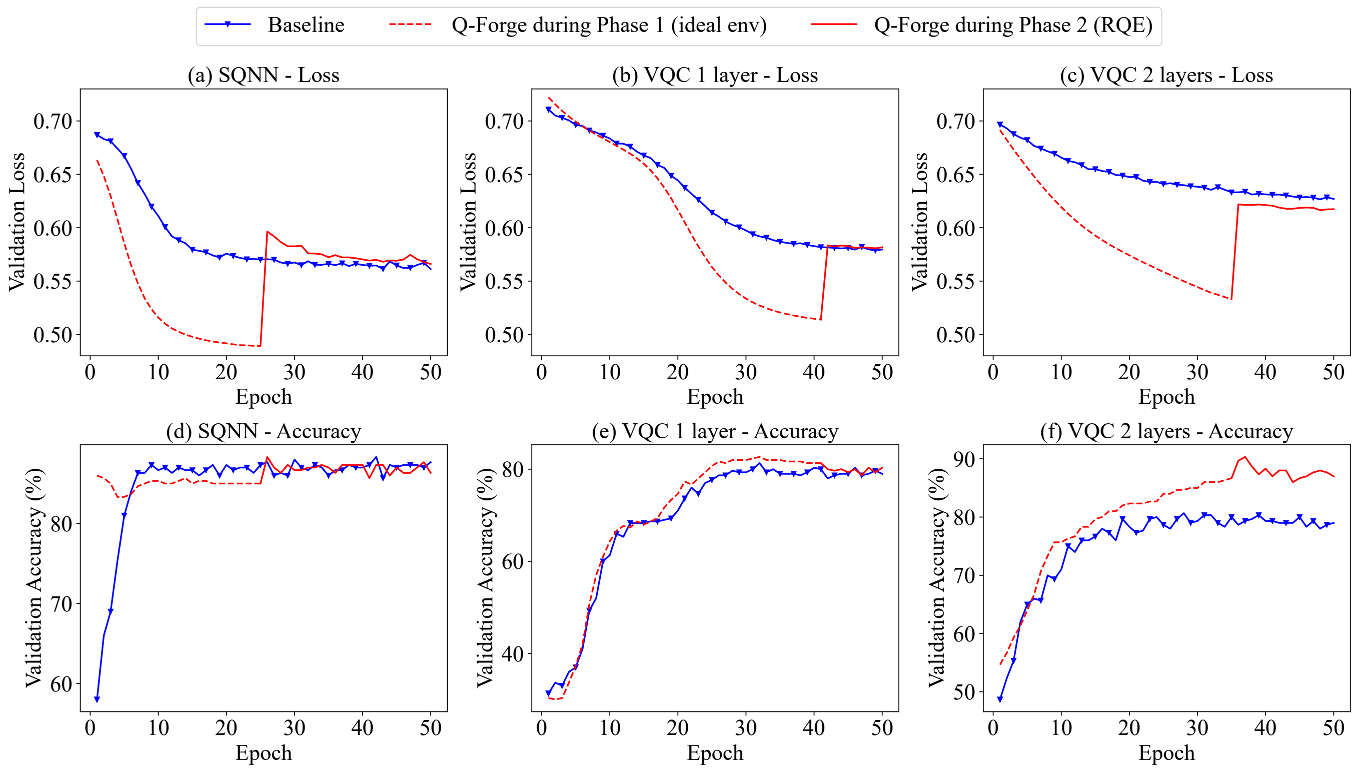

Fig. 2: The figure shows the training curves of three models in terms of loss (top row) and accuracy (bottom row). The solid blue line represents training/validation on the RQE only. The dashed red line denotes Q-Forge during Phase 1 (training on the ideal environment). The solid red line denotes Q-Forge during Phase 2 (fine-tuning on the RQE).

TABLE II: Performance evaluation of models after 50 epochs trained using *Q-Forge* and baseline.

Model	Method	Training		Testing		Time
		Acc	Loss	Acc	Loss	
SQNN	Baseline	88.0%	0.571	87.7%	0.561	89h37m
	Q-forge	87.4%	0.573	86.3%	0.566	45h41m
VQC_1	Baseline	81.0%	0.580	79.0%	0.580	35h55m
	Q-forge	81.1%	0.581	80.3%	0.582	07h31m
VQC_2	Baseline	82.7%	0.617	79.0%	0.627	91h23m
	Q-forge	88.0%	0.613	87.0%	0.617	27h21m

baseline. Although the final accuracy between the two methods on the SQNN and VQC_1 models has a small difference (less than 1%, possibly caused by quantization noise), *Q-Forge* still shows a superior advantage thanks to the savings in training costs. For VQC_2, both accuracy and training time are significantly improved, confirming that the main benefit of *Q-Forge* is to shorten the time to find the optimal parameter region of the quantum model, while also helping to reduce the *barren plateaus* [18] phenomenon.

C. Evaluation of Q-ACS algorithm

We conducted an experiment to analyze the impact of different manual transition points on the training process. For this analysis, the VQC_1 model was trained within the F-Forge framework for 50 epochs with a batch size of 64, using a set of predefined transition points where $\tau \in \{10, 20, 30, 40\}$. The experimental results show that, when increasing the value of τ, the loss on the validation set gradually decreases while

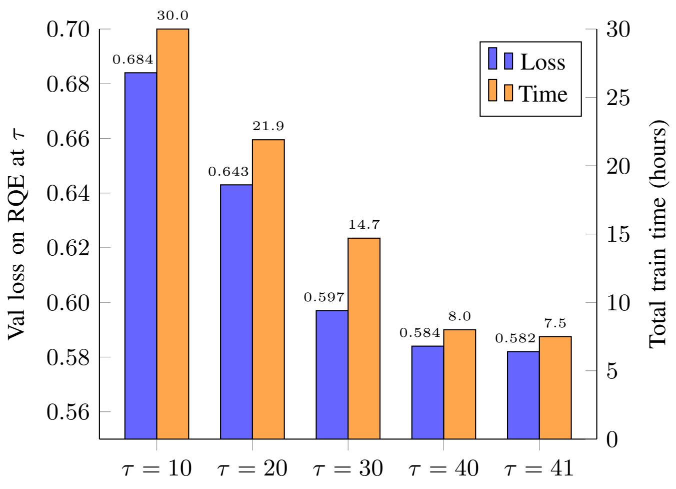

Fig. 3: The figure illustrates the impact of the transition points τ on the training process of the VQC_1 model within the *Q-Forge* framework.

the total training time is also significantly shortened. Figure 3 illustrates the training process of the VQC_1 model with different τ values in the *Q-Forge* method. Specifically, at $\tau = 10$ the model has validation loss = 0.684 with a training time of 30.0 hours; at $\tau = 20$ the loss decreases to 0.643 and the time is 21.9 hours; at $\tau = 30$ the loss continues to decrease to 0.597 with a time of 14.7 hours. Notably, a manual transition point at $\tau = 40$ results in a loss of 0.584 with a training time of 8.0 hours. In comparison, the transition point of $\tau = 41$, identified by the Q-ASC algorithm, achieves the minimum loss of 0.582 while also reducing the total training time to 7.5 hours. This confirms that the handoff point $\tau = 41$ chosen by the *Q-ASC* algorithm is optimal, providing the best balance between training efficiency and computational cost.

VI. Conclusion

In the NISQ era, the performance gap between ideal simulations and noisy quantum hardware remains a major barrier to practical applications of Quantum Neural Networks (QNNs). To address this challenge, this paper introduces Q-Forge, a two-stage training framework that enables quantum models to gain an initial advantage through ideal simulations before undergoing noise-aware fine-tuning on a realistic quantum emulator. Our initial experimental results suggest that Q-Forge not only significantly reduces the total training time but also enables the model to find a better parameter region than training directly in a noisy environment from the outset. While further validation is needed to confirm generalizability across larger systems and datasets, Q-Forge can become a useful tool to simplify and save costs for QML research, from building models that solve complex problems in noisy environments to developing reliability assurance mechanisms before practical deployment [28]. In the long term, Q-Forge represents a practical step toward the goal of deploying more robust and efficient QML applications on next-generation quantum devices.

Acknowledgement

This research is funded by University of Information Technology-Vietnam National University HoChiMinh City under grant number D1-2025-56.

References

[1] H. Wang, Z. Li, J. Gu, Y. Ding, D. Z. Pan, and S. Han, "On-chip qnn: Towards efficient on-chip training of quantum neural networks," *arXiv preprint arXiv:2202.13239*, 2022.

[2] M. Schuld, I. Sinayskiy, and F. Petruccione, "An introduction to quantum machine learning," *Contemporary Physics*, vol. 56, no. 2, pp. 172–185, 2015.

[3] E. H. Houssein, Z. Abohashima, M. Elhoseny, and W. M. Mohamed, "Machine learning in the quantum realm: The state-of-the-art, challenges, and future vision," *Expert Systems with Applications*, vol. 194, p. 116512, 2022.

[4] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[5] J. Wu, Z. Tao, and Q. Li, "wpscalable quantum neural networks for classification," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 38–48, IEEE, 2022.

[6] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli, and S. Woerner, "The power of quantum neural networks," *Nature Computational Science*, vol. 1, no. 6, pp. 403–409, 2021.

[7] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[8] E. A. Vashukevich, V. Lebedev, I. Ilichev, P. Agruzov, A. Shamrai, V. Petrov, and T. Golubeva, "Broadband chip-based source of quantum noise with electrically controllable beam splitter," *Physical Review Applied*, 2021.

[9] K. Georgopoulos, C. Emary, and P. Zuliani, "Modeling and simulating the noisy behavior of near-term quantum computers," *Physical Review A*, 2021.

[10] H. Wang, J. Gu, Y. Ding, Z. Li, F. Chong, D. Z. Pan, and S. Han, "Roqnn: Noise-aware training for robust quantum neural networks," 2021.

[11] H. Wang, J. Gu, Y. Ding, Z. Li, F. T. Chong, D. Z. Pan, and S. Han, "Quantumnat: quantum noise-aware training with noise injection, quantization and normalization," in *Proceedings of the 59th ACM/IEEE design automation conference*, pp. 1–6, 2022.

[12] H. Wang, Y. Ding, J. Gu, Y. Lin, D. Z. Pan, F. T. Chong, and S. Han, "Quantumnas: Noise-adaptive search for robust quantum circuits," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 692–708, IEEE, 2022.

[13] T. Chen, Z. Zhang, H. Wang, J. Gu, Z. Li, D. Z. Pan, F. T. Chong, S. Han, and Z. Wang, "Quantumsea: In-time sparse exploration for noise adaptive quantum circuits," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1, pp. 51–62, IEEE, 2023.

[14] J. Chen, Z. Cai, K. Xu, D. Wu, and W. Cao, "Qubit-wise architecture search method for variational quantum circuits," *arXiv preprint arXiv:2403.04268*, 2024.

[15] L. Cincio, K. Rudinger, M. Sarovar, and P. J. Coles, "Machine learning of noise-resilient quantum circuits," *PRX Quantum*, 2020.

[16] Y. Du, T. Huang, S. You, M.-H. Hsieh, and D. Tao, "Quantum circuit architecture search for variational quantum algorithms," *npj Quantum Information*, vol. 8, 2020.

[17] T. Li, L. Lu, Z. Zhao, Z. Tan, S. Tan, and J. Yin, "Qust: Optimizing quantum neural network against spatial and temporal noise biases," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[18] S. Wang, E. Fontana, M. Cerezo, A. Sone, L. Cincio, and P. J. Coles, "Noise-induced barren plateaus in variational quantum algorithms," *Nature Communications*, vol. 12, 2020.

[19] J. McClean, S. Boixo, V. Smelyanskiy, R. Babbush, and H. Neven, "Barren plateaus in quantum neural network training landscapes," *Nature Communications*, vol. 9, 2018.

[20] D. S. França and R. García-Patrón, "Limitations of optimization algorithms on noisy quantum devices," *Nature Physics*, vol. 17, pp. 1221 – 1227, 2020.

[21] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

[22] Z. Hu, Y. Lin, Q. Guan, and W. Jiang, "Battle against fluctuating quantum noise: Compression-aided framework to enable robust quantum neural network," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2023.

[23] W. R. Rabelo, S. D. Prado, and L. G. Brunnet, "A qaoa approach with fake devices: A case study for the maximum cut in ring graphs," *arXiv preprint arXiv:2404.03501*, 2024.

[24] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, "Analysis of representations for domain adaptation," *Advances in neural information processing systems*, vol. 19, 2006.

[25] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, P.-Y. Chen, X. Ma, and H.-S. Goan, "Variational quantum circuits for deep reinforcement learning," *IEEE access*, vol. 8, pp. 141007–141024, 2020.

[26] C. Chu, N.-H. Chia, L. Jiang, and F. Chen, "Qmlp: An error-tolerant nonlinear quantum mlp architecture using parameterized two-qubit gates," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 1–6, 2022.

[27] L. Prechelt, "Early stopping-but when?," in *Neural Networks: Tricks of the trade*, pp. 55–69, Springer, 2002.

[28] C. Lu, E. Telang, A. Aysu, and K. Basu, "Quantum leak: Timing side-channel attacks on cloud-based quantum services," in *Proceedings of the Great Lakes Symposium on VLSI 2025*, pp. 252–257, 2025.

[29] H. Wang, Z. Li, J. Gu, Y. Ding, D. Z. Pan, and S. Han, "Towards efficient on-chip training of quantum neural networks," 2021.