Quantum Neural Networks Need Checkpointing

Christopher Kverne [1], Mayur Akewar [1], Yuqian Huo [2], Tirthak Patel [2], Janki Bhimani [1]
[1]Florida International University, Miami, Florida, USA
[2]Rice University, Houston, Texas, USA
{ckver001,makew001,jbhimani}@fiu.edu,{ch107,tp53}@rice.edu

ABSTRACT

Quantum Neural Networks (QNNs) harness quantum superposition and entanglement, offering promising advantages for machine learning tasks. However, noise in quantum computers frequently disrupts QNN training, wasting computational resources and extending queue times. This paper introduces the first QNN checkpointing framework to address this challenge. Through experiments on various quantum devices, we demonstrate that QNN behavior is fundamentally hardware-dependent, with the same model performing differently across platforms. This key finding shows that quantum checkpoints require additional metadata about hardware specifics and shot counts unique to quantum systems. Our framework requires minimal storage (only 186.6KB for a 100-qubit QNN) and negligible overhead, enabling frequent checkpointing to enhance training resilience and reproducibility in the NISQ era.

CCS CONCEPTS

• **Hardware** → **Quantum computation**; • **Computing methodologies** → **Machine learning**; • **Computer systems organization** → *Reliability*.

KEYWORDS

Quantum Computing, Quantum Neural Networks, Checkpointing, Circuit Optimization

ACM Reference Format:
Christopher Kverne [1], Mayur Akewar [1], Yuqian Huo [2], Tirthak Patel [2], Janki Bhimani [1]. 2025. Quantum Neural Networks Need Checkpointing. In *17th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '25), July 10–11, 2025, Boston, MA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3736548.3737834

1 INTRODUCTION AND MOTIVATION

The rapid advancements in computing have fueled the development of increasingly sophisticated machine learning (ML) models, driving demand for greater computational efficiency and scalability. While classical computing has enabled significant progress in deep learning, certain tasks remain computationally expensive, leading to a growing shift toward quantum computing and quantum neural networks (QNNs), which can potentially revolutionize ML, addressing problems that are too complex for classical systems [1, 3].

However, the instability of Near-Term Intermediate-Scale Quantum (NISQ) computers presents significant challenges for Quantum Machine Learning (QML) practitioners. Quantum training sessions frequently terminate prematurely due to gate errors, external noise, software bugs, and limited coherence times [15, 24, 33]. Currently, it is a common practice to restart training entirely after such failures, resulting in substantial computational waste and extended queue times for accessing limited quantum resources [26]. Additionally, the QML community faces a reproducibility crisis as research findings are predominantly shared through code repositories alone, without accompanying trained model parameters [1, 4, 7]. This limitation forces researchers to retrain models from scratch to reproduce prior work introducing further inefficiencies due to hardware variability.

These challenges collectively highlight the urgent need for robust checkpointing mechanisms specifically designed for quantum neural networks. While classical checkpointing is a well-established field, with extensive research into areas such as minimizing performance overhead (e.g., through asynchronous checkpointing) [18, 21, 22], optimizing storage use (e.g., via compression) [20, 30], and ensuring data consistency and rapid recovery in large-scale distributed environments [6, 32], its quantum counterpart is fundamentally different. For instance, the findings presented in this paper highlight that QNNs performance is intrinsically linked to the hardware used for execution meaning that, unlike their classical counterparts, quantum checkpoints must incorporate hardware-specific metadata. Furthermore, the hybrid quantum-classical nature of QNN training allows for quantum checkpoints to be saved concurrently with ongoing training operations.

Prior work in Quantum Machine Learning (QML) has explored concepts such as transfer learning [12, 13, 17, 23], which also aims to reuse prior computations, however it differs fundamentally from checkpointing, which enables training recovery, model sharing, and seamless continuation from saved states. *To the best of our knowledge, this is the first paper to explore quantum checkpointing, defining its critical components, unique challenges, and proposing a scalable framework to optimize QNN training resilience and efficiency in the NISQ era.* The key contributions of this work are:

- A comprehensive analysis of quantum checkpointing requirements, highlighting key differences from classical checkpoints.
- A novel framework for quantum checkpointing, outlining core components and potential enhancements.
- Empirical evaluation of the impact of quantum hardware characteristics on training performance and model migration.
- Effective optimization strategies (pruning, quantization, freezing) to reduce circuit complexity with negligible performance loss.
- Detailed analysis of quantum checkpoint storage, demonstrating minimal overhead and efficient scalability.

2 PRELIMINARIES

2.1 Quantum Computers and Qubits

Qubits, the fundamental units of quantum computers, are quantum particles exhibiting superposition (the ability to exist in multiple states simultaneously) and entanglement (correlations where the state of one qubit depends on another). Unlike classical bits, restricted to 0 or 1, a qubit's state $|\psi\rangle$ is expressed as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|\alpha|^2$ and $|\beta|^2$ are the probabilities of measuring $|0\rangle$ and $|1\rangle$, respectively.

Quantum computers process qubits through operations where superconducting quantum computers employ rotational gates such as R_X, R_Y, R_Z, and entanglement gates such as $CNOT$ to manipulate the state of qubits. These four gates form a universal set capable of any quantum computation and are commonly used in QNNs.

2.2 Quantum Neural Networks

QNNs are parameterized quantum circuits that exploit superposition and entanglement to perform machine learning tasks. A typical QNN consists of three primary components: **Encoding Layer:** Maps the classical input data into quantum states on the circuit using techniques such as basis encoding [31], amplitude encoding [28], angle encoding [19], or matrix-product-state encoding (e.g., $R_x(\pi \cdot x_i)|0\rangle$).
Variational Layers: Built of trainable quantum gates, including single-qubit rotations (e.g. $R_x(\theta_k)$, $R_y(\theta_k)$, $R_z(\theta_k)$) and entangling gates (e.g., CNOT), arranged in a structured pattern to optimize a target function where θ_k is a trainable parameter [31]. Figure 1 shows a 1-layer 3-qubit QNN.
Measurement Layer: Collapses the quantum state into classical outputs by measuring qubits, yielding an output probability distribution over 2^n states (for n measured qubits). The circuit has to be prepared and measured multiple times (i.e., multiple "shots" have to be run) to generate the distribution.

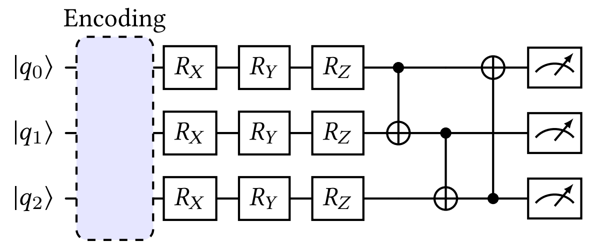


Figure 1: Schematic of a 3-qubit, 1-Layer Quantum Neural Network (QNN) with cyclic $CNOT$ gates employed in this study.

Training a QNN optimizes the variational parameters $\boldsymbol{\theta}$ to minimize a cost function C. However, quantum computers can't utilize backwards propagation to compute gradients in parallel. Instead they use the Parameter Shift Rule (PSR) to compute gradients one by one as described by Equation 1.

$$\frac{\partial C(\boldsymbol{\theta})}{\partial \theta_k} = \varsigma_k \left[C\left(\boldsymbol{\theta} + \frac{\pi}{4\varsigma_k}\mathbf{e}_k\right) - C\left(\boldsymbol{\theta} - \frac{\pi}{4\varsigma_k}\mathbf{e}_k\right)\right] \quad (1)$$

Where $\boldsymbol{\theta} \in \mathbb{R}^{l\times q\times g}$ is the tensor of all current parameter values; l represents the number of layers, q represents the number of qubits, and g represents the number of repeating gates in the variational layers (g = 3 for the ansatz we employed as seen in Fig 1). $\mathbf{e}_k$ is the k-th standard basis vector (a vector with 1 in the k-th position and 0s elsewhere), indicating that the shift is applied only to θ_k for each optimization step. For the commonly employed Pauli rotation gates (e.g., $R_X(\theta_k) = e^{-i\theta_k\sigma_X/2}, R_Y(\theta_k) = e^{-i\theta_k\sigma_Y/2}, R_Z(\theta_k) = e^{-i\theta_k\sigma_Z/2}$) the coefficient $\varsigma_k = \frac{1}{2}$, yielding a shift of $\pm\frac{\pi}{2}$. As seen in this formula, for a QNN with $|\boldsymbol{\theta}|$ parameters you need $2|\boldsymbol{\theta}| + 1$ total forward passes per input to compute $|\boldsymbol{\theta}|$ gradients, where the first forward pass is needed to compute the loss C.

2.3 Hybrid QNN Training Process

The training of a QNN is a hybrid quantum-classical process that can be seen in Figure 2, summarized by these steps:

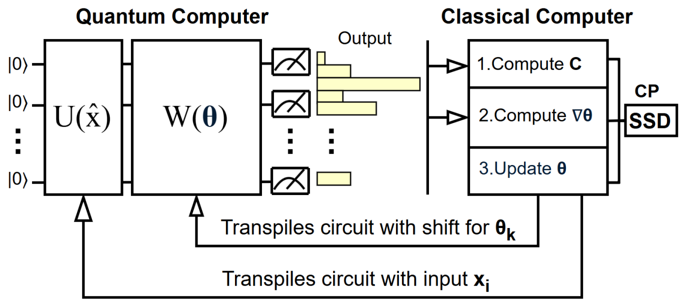


Figure 2: Quantum-Classical hybrid training approach enabling asynchronous checkpointing.

Loss Evaluation: Parameters ($\boldsymbol{\theta}$), stored classically, are transpiled into a quantum circuit. This circuit is run $\mathcal{M}$ times (shots) on a quantum computer returning an output distribution to the classical computer used to compute the loss.
Parameter-Shift Operations: To determine the gradient $\nabla\boldsymbol{\theta}$, for each parameter θ_k: The circuit is run with θ_k shifted by $\pm\frac{\pi}{2}$. This sequence of transpilation and execution for positive and negative shifts is performed for all parameters.
Gradient Computation & Parameter Update: The classical computer uses the outputs from all shifted evaluations to compute each component of the gradient $\frac{\partial C(\boldsymbol{\theta})}{\partial \theta_k}$ and updates the parameters based on these value. This process is then repeated for each input.
This demonstrates that quantum checkpointing is unique as the quantum computer only runs the forward pass and doesn't compute gradients or update and save the parameters, whereas a model run on a classical system (like GPU) is typically involved in both the training and checkpointing process.

3 QUANTUM CHECKPOINTING

To determine the optimal design of quantum checkpoints, we conducted a series of experiments exploring how QNNs perform across different hardware environments and under various optimization techniques. These experiments provide concrete evidence for our checkpointing framework design decisions and reveal unique quantum-specific considerations absent in classical checkpointing approaches. *The code used in these experiments is publicly available on GitHub (https://github.com/Damrl-lab/Quantum-Checkpointing) to promote reproducibility and encourage extensions of this work.*

3.1 Experimental Setup

Dataset and Task: To test our checkpointing approach, we employ the MNIST dataset [8], a widely used benchmark in machine learning, consisting of 28×28 grayscale images of handwritten digits. For the evaluation, we consider both binary classification (digits 0 and 1) and multi-class classification (digits 0, 1, 2, and 3), each containing 1,000 training and testing images per digit.

Quantum Simulation: Our experiments leverage the PennyLane framework [5] and the Qiskit Aer [2, 25] backend for running quantum circuits, which enables us to model and test different quantum devices. Specifically, we use four quantum hardware profiles: Rigetti Aspen (entry-level) [29], IBM Manila (intermediate) [2, 11], Google Sycamore (intermediate) [3], and IBM Eagle (advanced) [2, 11], which vary in their noise profiles and error characteristics. This diversity allows us to assess the robustness of our checkpointing strategy under a range of quantum hardware conditions.

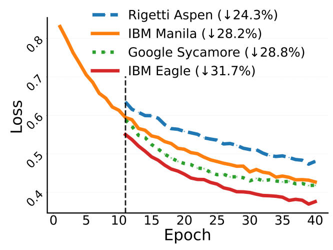
Error Sources: In line with the challenges mentioned in Section 1, we consider the following five major error sources: (1) *Single-qubit gate errors:* Errors in individual qubits during gate operations. (2) *Two-qubit gate errors (e.g., CNOT gate errors):* Errors during interactions between pairs of qubits. (3) T_1 *relaxation times (energy decay):* The loss of energy in qubits over time. 4) T_2 *dephasing times (loss of coherence):* The loss of information and coherence due to environmental factors. 5) *Readout (measurement) errors:* Errors that occur during the final measurement stage of the quantum computation. Each experiment was run with 1,000 shots per circuit execution. For the optimization process, we used either the Adam or SGD optimizer with a learning rate of 0.01, training for 40 epochs using a cross-entropy loss function.

3.2 Cross-Device Training

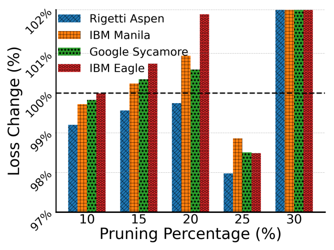
To investigate the feasibility of transferring QNN training across different quantum hardware platforms, we trained a 4-qubit, 2-layer QNN for 10 epochs on IBM Manila and transferred the trained model's parameters and optimizer states to Rigetti Aspen, IBM Eagle, and Google Sycamore for further training. Our aim was to understand if it's possible to continue the training on different devices and how each device impacts the training performance.

We found some surprising and interesting insights. The performance of the model was clearly hardware-dependent. As shown in Figure 3a, on the higher-performance systems (less noise, longer coherence times, higher gate fidelities, lower readout errors, etc.) like IBM Eagle, we saw a significant reduction in loss, around 31.7%, while on the lower-performance systems, like Rigetti Aspen, we noticed only about a 24.3% reduction in loss. This helped us derive the insight that *unlike classical ML training on CPU-GPU nodes, superior quantum hardware produces more accurate gradient estimations rather than merely faster execution.*

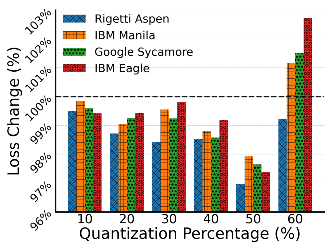
We also observed that models trained on lower-performance systems actually improved quite a bit when transferred to more powerful hardware. This suggests a



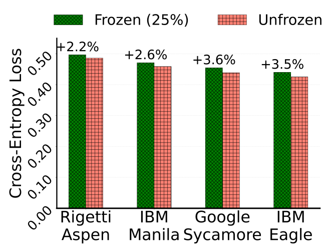
(a) Loss reduction after transferring a QNN trained on IBM Manila to various quantum hardware platforms for continued training.



(b) Effect of parameter pruning on QNN performance across quantum hardware, with dashed line indicating baseline (no pruning).



(c) Effect of quantization on QNN performance across quantum hardware, with dashed line indicating baseline (no quantization).



(d) Impact of freezing 25% of parameters during QNN training across quantum hardware, compared to no freezing.

Figure 3: Experimental results of quantum checkpointing across hardware and optimization techniques.

tiered model training approach could be beneficial, starting training on more accessible hardware (lower-performance systems are less in demand [26]) and then transitioning to higher-performing devices for fine-tuning and inference, similar principles have been employed by [34]. *Thus, we observe that, unlike classical checkpoints, quantum checkpoint performance is intrinsically linked to hardware characteristics, necessitating metadata about error profiles, coherence times, and gate fidelities to guide adaptation when resuming training on different hardware devices.*

3.3 Parameter Pruning and Quantization

Next, we explored how to optimize the quantum circuits themselves to improve model performance by applying pruning and quantization, two techniques commonly used in classical machine learning. These techniques can be combined with checkpoints to highlight parameters that can be minimized or removed when loading them. In pruning, we identify parameters close to 0, 2π, 4π, etc. and remove those gates from the circuit as they have minimal impact on the model's output. Quantization, on the other hand, involves converting continuous parameters into discrete "basis angles" that require fewer native gates during execution (e.g. an Rx gate with an angle of 87 degrees needs 5 native gates to be transpiled while an angle of 90 degrees only needs one).

Table 1: Circuit depth of rotational angles on compiled IBM quantum processors in the range $[0, 4\pi]$, constructed from [10]. Native gates $\mathbb{S} = \{CX, ID, RZ, SX, X\}$.

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
Rz	0	1	0	1	0	1	1	1	1	1
CX	1	1	1	1	1	1	1	1	1	1

Prunable and "basis angles" for IBM quantum processors can be seen in Table 1. Both of these techniques aim to reduce the gate count of the circuit, which reduces the total error in the system, potentially improving the model.

For our experiments, we used a 6-qubit, 3-layer QNN and tested various pruning and quantization levels across four different quantum devices. For parameter pruning and quantization, we implemented a distance-based selection strategy. We first calculated the Euclidean distance between each trainable parameter and its nearest prunable value (for pruning) or 'basis angle' (for quantization). Parameters with the smallest distances were prioritized for modification. Specifically, we selected the top n% of parameters with the shortest distances and set them exactly equal to their nearest prunable value or basis angle.

Our pruning experiments revealed that the quality of quantum hardware significantly impacts the benefits of pruning. As Figure 3b shows, while all systems reached optimal performance at 25% pruning, Rigetti Aspen consistently outperformed its baseline across all pruning levels (i.e., the loss is decreased compared to no pruning, as indicated by the dashed line in Figure 3b). The results show that pruning generally benefits noisier quantum hardware more where the threshold is crucial, as modest pruning can reduce loss by reducing error, but excessive pruning can eliminate important information. In the case of quantization, as seen in Figure 3c, we found that 50% quantization was the optimal threshold across all systems, resulting in 2-3% loss reduction. Similar to pruning, quantization is hardware-dependent and generally benefits noisier hardware more.

3.4 Parameter Freezing

Finally, to reduce the computational overhead of QNN training, we investigated parameter freezing, where parameters with small, effectively converged gradients are fixed. This technique reduces the required forward passes per data point proportionally to the percentage of parameters frozen, due to the PSR (Eq. 1). In our experiment, after training for five epochs, we froze the 25% of parameters exhibiting the smallest average gradients over recent data points. This simplified method, inspired by the adaptive threshold approach in [16], yielded comparable results with lower complexity. We then continued training the 'frozen' model for 35 epochs, comparing its performance against a model trained without freezing.

As shown in Figure 3d, the results were encouraging: the performance penalties were modest, with Rigetti Aspen showing a 2.2% increase in loss and IBM Eagle showing a 3.5% increase in loss. Interestingly, lower-performance systems experienced smaller performance penalties from parameter freezing. We believe these results come from two reasons: (1) Noisier systems calculate worse gradients (as shown in Sec 3.2), and smaller gradients are likely harder to calculate. (2) Small parameter updates are likely less impactful on noisier systems.

Our experiments with pruning, quantization, and parameter freezing reveal a consistent pattern: optimization strategies for QNNs are inherently hardware-dependent, with noisier systems generally benefiting more. These findings have direct implications for checkpointing strategies. Rather than storing specific pruned, quantized, or frozen states, quantum checkpoints should maintain the original parameters along with relevant metadata (pruning or quantization candidates, and gradient history). This approach enables each quantum system to adaptively determine appropriate optimization thresholds based on its specific noise characteristics and performance profile. For example, a checkpoint loaded onto a noisier system might apply more aggressive pruning or begin parameter freezing earlier in the training process. This flexibility is crucial because, as our experiments demonstrate, optimization thresholds that benefit one system (e.g., 60% quantization or 20% pruning) may not benefit others.

4 PROPOSED FRAMEWORK FOR QUANTUM CHECKPOINT

Based on our experimental results, we propose a comprehensive framework for quantum checkpointing that addresses the unique challenges of QNNs while optimizing for both performance and resource efficiency. Our framework defines four essential components and three optional components for effective quantum checkpoints.

4.1 Essential Components

1. Circuit Parameters: The primary component of any quantum checkpoint is the set of rotation angles $\boldsymbol{\theta}$.

2. Optimizer State: Quantum checkpoints must preserve the complete state of the classical optimizer, like momentum terms for Adam.

3. Hardware Profile: Unlike classical checkpoints, quantum checkpoints must include metadata about the training hardware's noise characteristics. As demonstrated in our experiments, model performance is intrinsically linked to the quantum hardware's error rates and coherence properties. Similarly, this information is crucial when applying pruning, quantization, and freezing. This information could be stored in a separate file from the parameters and doesn't need to be updated as frequently as they change less often. We recommend storing key hardware metrics including:

- Single and two-qubit gate error rates
- Qubit coherence (T_1 and T_2) times
- State preparation and measurement errors (SPAM)
- Gate execution times
- Qubit measurement times

4. Number of Shots: This information is essential because shots directly impacts the precision of gradient estimates during training. Fewer shots introduce statistical noise that can destabilize optimization, while more shots increase accuracy but at higher computational cost. When resuming training from a checkpoint, maintaining consistent shots preserves the statistical properties of the learning process ensuring stable convergence.

4.2 Optional Enhancement Components

1. Gradient History: Our parameter freezing experiments in Section 3.4 demonstrated that storing gradient history enables significant computational savings through intelligent freezing strategies. By freezing 25% of parameters, we reduced forward passes by 25% while incurring only 2.2-3.5% loss increases. The freezing threshold can be adapted to the hardware that the checkpoint is trained on or moved to.

2. Pruning and Quantization Masks: As optimal pruning and quantization thresholds are hardware-dependent, checkpoints can store pruning and quantization masks from the original training, providing starting points that can be fine-tuned for new hardware.

3. Qubit Mapping: For checkpoints intended for use on the same quantum hardware, preserving the specific qubit mapping ensures the same physical qubits are used when loading the model. (qubits with similar error profiles will give more consistent results).

4.3 Size and Overhead of Checkpoint

The size and computational overhead of our proposed quantum checkpoint are both minimal compared to classical checkpoints. For a QNN with l layers, q qubits, and g repeating gates in the ansatz layer (g=3 when Rx, Ry, Rz rotation gates are employed), we quantify the storage requirements as follows:

Circuit Parameters: $l \times q \times g$ double precision values (64-bit each) for the rotation angles, totaling $8 \times l \times q \times g$ bytes.

Optimizer State: For Adam optimizer [14], momentum and variance vectors require $2 \times l \times q \times g$ double-precision values, totaling $16 \times l \times q \times g$ bytes.

Hardware Profile: q qubit error rates, e two-qubit error rates (where e represents the number of edges in the qubit connectivity graph), $2 \times q$ coherence times (T_1 and T_2), and q readout errors, totaling approximately $(4q + e) \times 8$ bytes for a fully connected topology. Note: $e = \frac{1}{2}q(q-1)$ if all qubits are connected to each other (fully connected graph); however, the qubit connectivity is typically much lower due to hardware constraints (order $O(q)$) [3, 9, 27].

Total Size: To illustrate the storage requirements of our proposed checkpointing framework, consider a QNN with 100 qubits and 20 layers, a scale significantly larger than QNNs currently employed. The approximate checkpoint size is:

- Circuit Parameters: $8 \times 20 \times 100 \times 3 = 48$ KB
- Optimizer State: $16 \times 20 \times 100 \times 3 = 96$ KB
- Hardware Profile: ≈ 42.8 KB (assuming full connectivity, typically much lower)
- Total: ≈ 186.8 KB

Even for such a large-scale QNN, the total checkpoint size remains modest. Notably, quantum checkpoints are generally much smaller than their classical counterparts, primarily because QNNs often require fewer parameters due to the inherent expressivity of qubits and entanglement [1]. The computational overhead of saving these quantum checkpoints is also minimal for two primary reasons:

(1) All necessary checkpoint data (parameters, optimizer state, hardware information, etc.) already reside in the classical computer's memory during the hybrid QNN training process, eliminating the need for data transfers from specialized hardware like GPUs (See Sec. 2.3).
(2) Checkpointing can be performed asynchronously by the classical computer during the quantum computer's circuit evaluation period, as these evaluations typically take longer than writing the checkpoint data to disk.

This negligible overhead and size enables highly frequent checkpointing strategies with no impact on training performance. We recommend checkpointing after every epoch and potentially even more for particularly unstable quantum hardware or large epochs.

5 DISCUSSION

Our results emphasize the need for dynamic hardware-aware QNN checkpointing. We observed that noisier systems can tolerate more aggressive optimizations, such as higher pruning, quantization, or freezing thresholds. However, the precise mechanism for dynamically adjusting these thresholds based on hardware error profiles when transferring checkpoints across systems remains an open challenge.

Additionally, a key limitation of quantum checkpointing concerns the transferability of checkpoints between fundamentally different quantum computing architectures. Although our framework enables checkpoint portability within a given technology class (e.g., between different superconducting processors), direct transfer between different platforms, such as superconducting gate-based systems and neutral-atom Rydberg systems, is currently infeasible. These architectures differ significantly in their physical qubit implementations, control mechanisms (e.g., microwave gates versus laser pulses), and the nature of their trainable parameters (e.g., gate angles versus pulse characteristics like Rabi frequency). Bridging this gap might necessitate advanced techniques, perhaps similar to transfer learning, to map learned knowledge from one type of quantum system to another.

6 CONCLUSION

Our work demonstrates that QNN checkpointing needs a more sophisticated approach than storing just classical information, as NISQ-era quantum computers face unique challenges. The research shows that effective quantum checkpoints must include hardware-specific information alongside circuit parameters, optimizer states, and the number of shots. Our experiments reveal that different quantum hardware responds uniquely to optimization techniques, with noisier systems benefiting more from pruning, quantization, and freezing. Our proposed framework requires minimal storage (approximately 186.6KB for a 100-qubit QNN) and negligible computational overhead, enabling frequent checkpointing that enhances resilience and reproducibility in QNNs.

7 ACKNOWLEDGMENTS

This work was supported by the following NSF grants: CSR-2402328, CAREER-2338457, CSR-2406069, CSR-2323100, and HRD-2225201, as well as the DOE Quantum Testbed Finder Award DE-SC0024301 and NERSC DDR-ERCAP0034716.

REFERENCES

[1] Amira Abbas, David Sutter, Christa Zoufal, Aurélien Lucchi, Alessio Figalli, and Stefan Woerner. 2021. The power of quantum neural networks. *Nature Computational Science* 1, 6 (2021), 403–409.
[2] G. Aleksandrowicz, T. Alexander, P. Kl. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C.-F. Chen, J. M. Chow, A. D. Córcoles-Gonzales, A. J. Cross, A. W. Cross, J. Cruz-Benito, C. Culver, S. De La Puente González, E. De La Torre, D. Ding, et al. 2019. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2562111

[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.

[4] Kerstin Beer, Dmytro Bondarenko, Terry Farrelly, Tobias J Osborne, Robert Salzmann, Daniel Scheiermann, and Ramona Wolf. 2020. Training deep quantum neural networks. *Nature communications* 11, 1 (2020), 808.

[5] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M Sohaib Alam, Guillermo Alonso-Linaje, B AkashNarayanan, Ali Asadi, et al. 2018. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968* (2018).

[6] Jiajun Cao, Kapil Arya, Rohan Garg, Shawn Matott, Dhabaleswar K Panda, Hari Subramoni, Jérôme Vienne, and Gene Cooperman. 2016. System-level scalable checkpoint-restart for petascale computing. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 932–941.

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

[8] Li Deng. 2012. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142. https://doi.org/10.1109/MSP.2012.2211477

[9] Jay M Gambetta, Jerry M Chow, and Matthias Steffen. 2017. Building logical qubits in a superconducting quantum computing system. *npj quantum information* 3, 1 (2017), 2.

[10] Zhirui Hu, Peiyan Dong, Zhepeng Wang, Youzuo Lin, Yanzhi Wang, and Weiwen Jiang. 2022. Quantum neural network compression. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[11] IBM Quantum. 2025. Get QPU Information. https://docs.quantum.ibm.com/guides/get-qpu-information. Accessed: March 2025.

[12] Amena Khatun and Muhammad Usman. 2025. Quantum Transfer Learning with Adversarial Robustness for Classification of High-Resolution Image Datasets. *Advanced Quantum Technologies* 8, 1 (2025), 2400268.

[13] Juhyeon Kim, Joonsuk Huh, and Daniel K Park. 2023. Classical-to-quantum convolutional neural network transfer learning. *Neurocomputing* 555 (2023), 126643.

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Linsey Kitt and Myra B. Cohen. 2024. MorphQ++: A Reproducibility Study of Metamorphic Testing on Quantum Compilers. In *2024 39th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. 8–14.

[16] Ankit Kulshrestha, Xiaoyuan Liu, Hayato Ushijima-Mwesigwa, Bao Bach, and Ilya Safro. 2024. QAdaPrune: Adaptive Parameter Pruning For Training Variational Quantum Circuits. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Vol. 2. IEEE, 120–125.

[17] Andrea Mari, Thomas R Bromley, Josh Izaac, Maria Schuld, and Nathan Killoran. 2020. Transfer learning in hybrid classical-quantum neural networks. *Quantum* 4 (2020), 340.

[18] Avinash Maurya, Robert Underwood, M Mustafa Rafique, Franck Cappello, and Bogdan Nicolae. 2024. Datastates-llm: Lazy asynchronous checkpointing for large language models. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*. 227–239.

[19] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. 2018. Quantum circuit learning. *Physical Review A* 98, 3 (2018), 032309.

[20] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. 2021. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295* (2021).

[21] Bogdan Nicolae, Jiali Li, Justin M. Wozniak, George Bosilca, Matthieu Dorier, and Franck Cappello. 2020. DeepFreeze: Towards Scalable Asynchronous Checkpointing of Deep Learning Models. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. 172–181. https://doi.org/10.1109/CCGrid49817.2020.00-76

[22] Bogdan Nicolae, Adam Moody, Elsa Gonsiorowski, Kathryn Mohror, and Franck Cappello. 2019. VeloC: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 911–920. https://doi.org/10.1109/IPDPS.2019.00099

[23] Soronzonbold Otgonbaatar, Gottfried Schwarz, Mihai Datcu, and Dieter Kranzlmüller. 2023. Quantum Transfer Learning for Real-World, Small, and High-Dimensional Remotely Sensed Datasets. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 16 (2023), 9223–9230. https://doi.org/10.1109/JSTARS.2023.3316306

[24] Matteo Paltenghi and Michael Pradel. 2023. MorphQ: Metamorphic testing of the Qiskit quantum computing platform. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2413–2424.

[25] Qiskit Aer Developers. 2019-2025. *qiskit-aer*. https://github.com/Qiskit/qiskit-aer

[26] Gokul Subramanian Ravi, Kaitlin N Smith, Pranav Gokhale, and Frederic T Chong. 2021. Quantum computing in the cloud: Analyzing job and machine characteristics. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 39–50.

[27] Matthew Reagor, Christopher B Osborn, Nikolas Tezak, Alexa Staley, Guenevere Prawiroatmodjo, Michael Scheer, Nasser Alidoust, Eyob A Sete, Nicolas Didier, Marcus P da Silva, et al. 2018. Demonstration of universal parametric entangling gates on a multi-qubit lattice. *Science advances* 4, 2 (2018), eaao3603.

[28] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. 2014. Quantum support vector machine for big data classification. *Physical review letters* 113, 13 (2014), 130503.

[29] Rigetti Computing. 2025. Rigetti Quantum Processors. https://www.rigetti.com/what-we-build. Accessed: March 2025.

[30] Naoto Sasaki, Kento Sato, Toshio Endo, and Satoshi Matsuoka. 2015. Exploration of lossy compression for application-level checkpoint/restart. In *2015 IEEE international parallel and distributed processing symposium*. IEEE, 914–922.

[31] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. 2014. The quest for a quantum neural network. *Quantum Information Processing* 13 (2014), 2567–2586.

[32] Foteini Strati, Michal Friedman, and Ana Klimovic. 2025. PCcheck: Persistent Concurrent Checkpointing for ML. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 811–827.

[33] Jiyuan Wang, Qian Zhang, Guoqing Harry Xu, and Miryung Kim. 2021. QDiff: Differential Testing of Quantum Software Stacks. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 692–704. https://doi.org/10.1109/ASE51524.2021.9678792

[34] Meng Wang, Poulami Das, and Prashant J. Nair. 2024. Qoncord: A Multi-Device Job Scheduling Framework for Variational Quantum Algorithms. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 735–749. https://doi.org/10.1109/MICRO61859.2024.00060