Milestone P1: Initial Experiment and Evaluation Setup

Benchmarking Optimization Methods for Quantum Recurrent Neural Networks

Erik Connerty Michael Stewart

October 16, 2025

Overview

This milestone presents the progress achieved in developing the experimental framework for benchmarking optimization techniques within Quantum Recurrent Neural Networks (QRNNs). The objective of this phase was to establish a reproducible environment, implement a functioning baseline model, and conduct preliminary experiments to demonstrate meaningful learning behavior and validate system performance before proceeding to large-scale evaluations.

1 System and Dataset Setup

The initial implementation was developed using the PennyLane framework due to its convenience in prototyping hybrid quantum-classical models. PennyLane's high-level abstraction enabled rapid experimentation with quantum layers and automatic differentiation, which was particularly useful in early stages of the project. However, as the complexity of the circuits increased—particularly with the introduction of mid-circuit measurements and non-unitary operations—PennyLane's execution overhead became a limiting factor. Each training update using SPSA required on the order of eight seconds (Apple M3 Max used), making iterative experimentation infeasible for the scope of this study.

To overcome these performance limitations, the model was reimplemented using a combination of **PyTorch** and **IBM Qiskit**. In this configuration, Qiskit serves as the quantum circuit backend while PyTorch handles the classical training loop and parameter management. This design choice enables seamless integration with standard deep learning workflows while allowing more precise control over the quantum execution pipeline. The migration led to a dramatic improvement in runtime efficiency: per-update times were reduced from

roughly eight seconds in PennyLane to approximately 0.33 seconds in the PyTorch–Qiskit implementation. While this speedup is promising, an error in the PyTorch configuration actually caused some of the weights in the QRNN model to not update, possibly making this execution time a bit faster than it would be in a proper implementation.

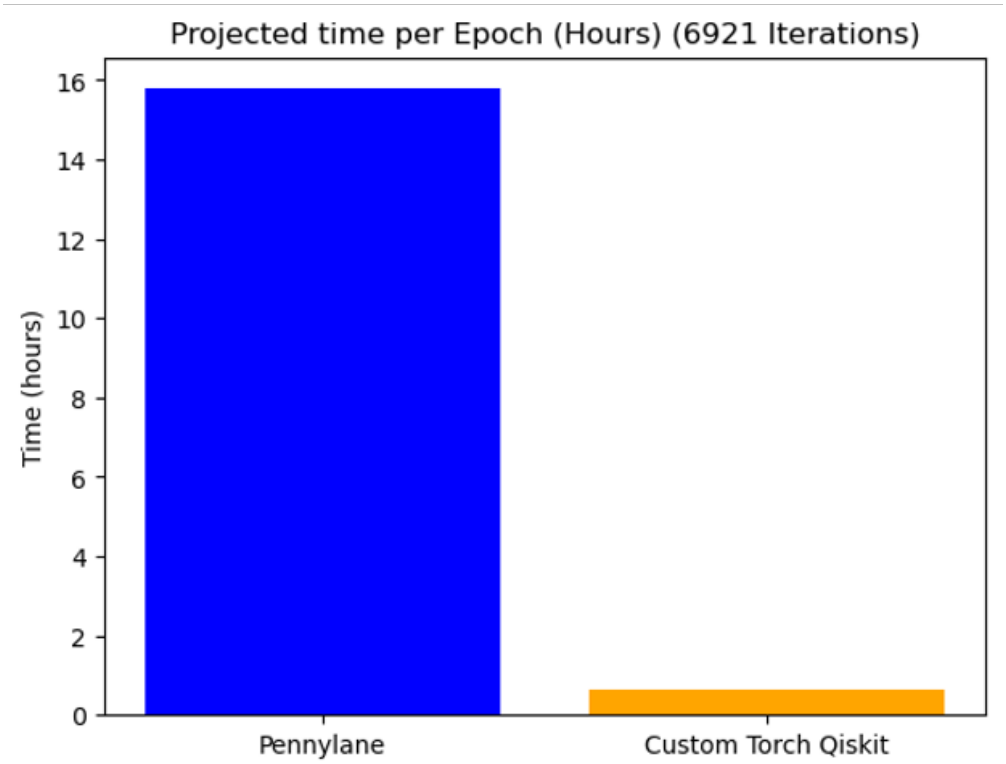

Figure 1: Comparison of training update times using SPSA between PennyLane and the PyTorch–Qiskit implementation. The latter achieves more than an order-of-magnitude speedup, enabling practical multi-epoch training. Due to an error in PyTorch configuration, the numbers for the torch-qiskit model are slightly off.

All experiments in this milestone were performed on CPU-based simulations using the Qiskit Aer backend. This decision allows for full control over execution fidelity and reproducibility while deferring QPU runs until later phases, when the circuits and training configurations have stabilized. The chosen dataset for the preliminary experiments was the **Lorenz system**, a canonical chaotic time-series used extensively for benchmarking recurrent models. This system serves as a controlled environment for debugging and validating the hybrid pipeline before scaling up to the more complex **Kuramoto–Sivashinsky** system, which will be employed in subsequent milestones.

2 Baseline Implementation

The baseline model implemented for this milestone consists of a hybrid Quantum Recurrent Neural Network (QRNN) formulated as a custom PyTorch layer. A parameterized quantum circuit processes sequential inputs over time. The quantum circuit is executed through Qiskit Aerr, while the recurrent connections and overall model structure are managed by PyTorch. This architecture enables an end-to-end differentiable hybrid model, where classical optimizers can be used to train quantum parameters through either exact or approximate gradient estimation techniques.

Two optimization methods were prepared for comparison: the **Simultaneous Perturbation Stochastic Approximation (SPSA)** algorithm and the **finite-difference** gradient method. SPSA is an approximate gradient-based optimization scheme that evaluates gradients with only two circuit executions per update, regardless of the number of parameters, making it computationally efficient. The finite-difference method, in contrast, computes more accurate gradients but at the cost of additional circuit evaluations. Both approaches were integrated into the same training framework, allowing direct performance and accuracy comparisons under equivalent conditions.

It was initially reported that the **parameter-shift** method would be benchmarked, but initial experiments showed that this was impossible. Non-unitary operations in the quantum circuit are incompatible with this algorithm, so the **finite-differences** method replaced it. Asymptotically, both the parameter-shift and finite-differences method are in $O(PN)$, where P is the number of parameters and N is the number of optimization steps, so the comparisons are still valid vs the $O(N)$ SPSA algorithm.

3 Preliminary Experiment

A preliminary experiment was conducted using the Lorenz system to validate the functionality of the implemented pipeline and assess early learning behavior. The task involved one-step-ahead forecasting of the y and z components of the Lorenz trajectory using only the x component. A QRNN with eight qubits per recurrent step was used. The model was trained using the Adam optimizer with a learning rate of 10^{-3} and evaluated using mean squared error (MSE) as the primary metric, but plotted with root mean squared error (RMSE).

Training curves showed noisy convergence across epochs, demonstrating the inherent stochasticity of quantum systems. The final test RMSE achieved was approximately 0.116, demonstrating adequate performance. Later analysis however showed an error in the gradient calculation which meant the QRNN weights were not actually being updated. This also

meant that the finite-differences method was likely "training" much faster than it would normally, further complicating some of the earlier analysis. These issues will be addressed in the next milestone.

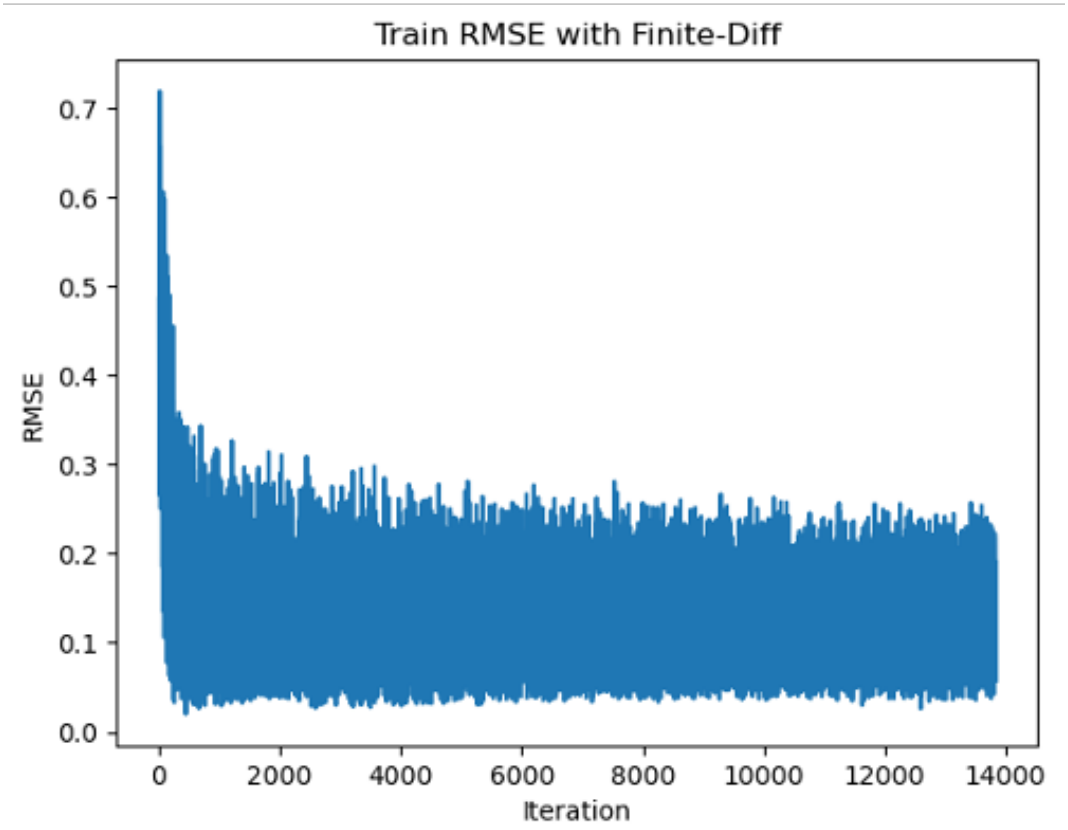

Figure 2: Training RMSE versus iteration number for the Lorenz forecasting task using finite-differences. An error in the PyTorch config caused some of the QRNN's weights to not update for this run, so performance could be improved further.

These preliminary demonstrate the viability of the hybrid QRNN implementation and confirm that the hybrid PyTorch layer can be constructed.

4 Summary and Next Steps

In this milestone, we established a functional and efficient experimental pipeline for QRNN benchmarking. Transitioning from PennyLane to a PyTorch–Qiskit framework yielded a substantial computational speedup that enables scalable experimentation. The baseline QRNN demonstrated meaningful predictive behavior on the Lorenz dataset, confirming the model's correctness and the stability of the hybrid training loop.

The next phase of work will expand these experiments to the Kuramoto–Sivashinsky

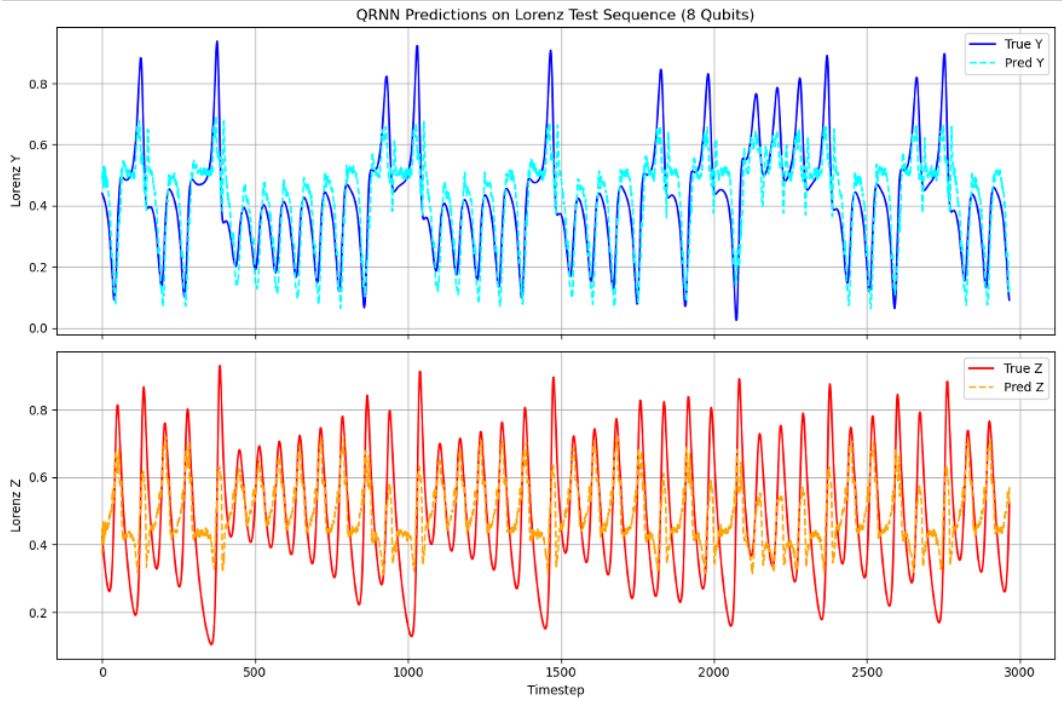

Figure 3: Predicted versus true Lorenz trajectories showing the QRNN's ability to reproduce short-term chaotic dynamics.

system, a more demanding benchmark that will better expose the trade-offs between SPSA and finite-difference optimization. Additionally, a GPU implementation and more rigorous comparisons will be added to give faster training times and quantify the computational advantages and performance of each optimization method. We also aim to improve the predictive accuracy of the model by tweaking the architecture. These next experiments will form the basis for our next milestone.