Quantum Recurrent Neural Networks for Dynamical Systems

Yuan Chen
Computational and Data Science PhD Program
Middle Tennessee State University

August 15, 2024

Abstract

In this study, we explore the application of Quantum Recurrent Neural Networks in modeling dynamical systems, focusing on Quantum Long Short-Term Memory and Quantum Gated Recurrent Unit models. These quantum-enhanced models, which integrate Variational Quantum Circuits into classical architectures, are evaluated against their traditional counterpart. The models are tested on a range of dynamical systems, including the Van der Pol oscillator, coupled damped harmonic oscillators, the Lorenz system, and the Robertson chemical reaction system, with a specific test of Quantum Long Short-Term Memory focus on stiff systems. Our results demonstrate that quantum-based models consistently outperform their classical counterparts, particularly in handling stiff ordinary differential equations and chaotic systems. The quantum models exhibit faster convergence, better generalization, and improved stability in training, highlighting the potential of quantum machine learning in solving complex dynamical systems.

1 Introduction

Ordinary differential equations (ODEs) are foundational mathematical models that play a critical role across diverse domains, such as physics, biology, and economics. The numerical resolution of ODE systems is pivotal for understanding the dynamics of these varied

processes. To this end, numerous methodologies have been devised, including linear multi-step methods and Runge–Kutta methods. Nevertheless, the practical application of these methods frequently encounters the complexity of systems characterized by nonlinear and time-varying elements, alongside a pronounced sensitivity to initial conditions. Such complexities significantly challenge the efficacy of traditional analytical and numerical strategies, especially in terms of achieving desired levels of precision, stability, and computational efficiency. Given these considerations, there is a pressing need for more sophisticated techniques capable of navigating the intricate landscape of ODE systems.

In recent years, recurrent neural networks (RNNs) have achieved significant breakthroughs in processing sequential data. RNNs and their specific variants, such as Long Short-Term Memory (LSTM)[**1**] and Gated Recurrent Units (GRUs) [**2**], have proven to be efficacious in learning or forecasting complex, nonlinear challenges. The intrinsic nature of RNNs, which are capable of capturing temporal dependencies and learning from sequential data, makes them particularly suited for modeling dynamical systems.

Concurrently, the advent of variational quantum algorithms and circuits, pioneered by Mitarai et al. [**3**], has been a landmark development in the integration of quantum computing with machine learning. The study by Chen et al. [**4**, **5**]. innovated in the quantum machine learning domain by introducing Quantum LSTM (QLSTM) and Quantum GRU (QGRU), integrating variational quantum circuits (VQCs) with recurrent neural network architectures. However, the scenarios examined in Chen's studies only involved singular ODEs, which prompted us to explore the application of QRNN to systems of ODEs.

In this research, we explore the application of Quantum Long Short-Term Memory (QLSTM) and Quantum Gated Recurrent Unit (QGRU) models, based on the Variational Quantum Circuits (VQCs), to complex dynamical systems described by ODEs. Our approach involves a nuanced modification to the VQC, thereby refining its compatibility with QLSTM and QGRU models for their application in this context. Our focus is on the Van der Pol oscillator, two coupled damped harmonic oscillators, and the Lorenz equations.

2 Fundamental Quantum Computing Concepts

In this section, we introduce some basic concepts about quantum computing, including qubits, entanglements and some basic quantum gates.

2.1 Qubits

The core distinction between classical and quantum computing lies in their fundamental units of information: the classical bit and the quantum bit (qubit), respectively. In classical computing, a bit is a binary unit that can exist in one of two states, either 0 or 1. This binary nature underpins all classical computing processes, with data storage and computations executed through combinations of these binary states.

In contrast, the qubit, the fundamental unit of quantum computing, transcends this binary limitation. A qubit differs from a classical bit in its ability to exist in a superposition of states. Rather than being limited to a strict 0 or 1, a qubit can represent both 0 and 1 simultaneously, a phenomenon that is central to quantum computing's potential for processing complexity.

This unique property of qubits arises from the principles of quantum mechanics. Unlike classical bits, whose state is definite and observable, a qubit's state is described probabilistically. The actual state of a qubit is not determined until it is measured. Before measurement, a qubit exists in a superposition of the states $|0\rangle$ and $|1\rangle$, where the probabilities of these states are determined by quantum amplitudes. These amplitudes, which are complex numbers, dictate the likelihood of the qubit collapsing into either the $|0\rangle$ or $|1\rangle$ state upon measurement.

2.2 Superpositions

Superposition in a quantum system allows a qubit to exist in multiple states simultaneously until it is measured. This characteristic differentiates a quantum bit from a classical bit, which can only be in one state at any given time. In a single qubit system, the state of a qubit can be described as a linear combination of the basis states $|0\rangle$ and $|1\rangle$. Mathematically, this is represented

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1}$$

where α and β are complex numbers representing the probability amplitudes for the qubit being in state $|0\rangle$ and $|1\rangle$, respectively. According to the Born's rule, the probabilities of finding the qubit in either state upon measurement are $|\alpha|^2$ and $|\beta|^2$, with the condition that $|\alpha|^2 + |\beta|^2 = 1$.

For example, if $\alpha = \sqrt{0.3}$ and $\beta = \sqrt{0.7}$, the qubit has a 30% probability of being measured

in state $|0\rangle$ and a 70% probability of being in state $|1\rangle$.

Extending the concept of superposition to a two-qubit system, we find a more complex scenario. In a two-qubit system, the combined state can be represented as a superposition of all four possible states of the two qubits. The state vector for such a system is given by:

$$|\Psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle, \tag{2}$$

where α, β, γ, and δ are complex probability amplitudes corresponding to each of the four states. As per the normalization condition, the sum of the squares of the absolute values of these amplitudes must equal one:

$$|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1. \tag{3}$$

In both single and two-qubit systems, superposition allows quantum computers to process and encode information in ways that are fundamentally different from classical computers, enabling them to perform complex calculations more efficiently.

In traditional computing, gates are fundamental building blocks that process binary information, operating on bits that exist in one of two states and common logic gates include AND, OR, NOT, and XOR, each performing a specific logical operation for computational tasks.

Similarly, quantum computing has quantum gates, which manipulate qubits, such as the Hadamard, Pauli-X, Y, Z, and Controlled-NOT (CNOT) gates, operating on these qubits, enabling complex operations that can entangle qubits, and creating correlations between them that are essential for quantum computation's power. These gates are unitary, meaning they are reversible, a property that contrasts with some irreversible classical gates.

Here, we introduce two fundamental gates in quantum computing, the first one is the Hadamard gate, often used to create a superposition of states. The Hadamard gate acts on a single qubit and transforms it into a superposition of its basis states.

The Hadamard gate is represented by the following matrix:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{4}$$

This matrix operates on a qubit state vector, transforming it from a definite state ($|0\rangle$ or $|1\rangle$) into a superposition. When the Hadamard gate acts on the state $|0\rangle$ (represented by the vector $\begin{pmatrix}1\\0\end{pmatrix}$), the resulting state is:

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}. \tag{5}$$

Similarly, when it acts on the state $|1\rangle$ (represented by the vector $\begin{pmatrix}0\\1\end{pmatrix}$), the output is:

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \tag{6}$$

The Hadamard gate, therefore, creates an equal superposition of the $|0\rangle$ and $|1\rangle$ states.

2.3 Quantum Entanglement

Quantum entanglement is a phenomenon in which multiple qubits become interconnected in such a way that the state of one qubit cannot be described independently of the state of the other qubits, even when the qubits are separated by large distances.

Entanglement can be achieved using certain quantum gates, which is the second gates we introduce here, a Controlled-NOT (CNOT) gate. It is commonly used to entangle two qubits.

First, the Hadamard gate is applied to one of the qubits to create a superposition, as previously discussed. Then, the CNOT gate, which flips the state of the second qubit if the first qubit is in the $|1\rangle$ state, is applied. The operation of the CNOT gate can be described by the following matrix:

$$\text{CNOT} = \begin{pmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{pmatrix}. \tag{7}$$

This operation creates entanglement between the two qubits.

Consider two qubits, initially in the state $|00\rangle$. Applying a Hadamard gate to the first qubit creates the superposition $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ for the first qubit. Now, the combined state of the two qubits is $\frac{|00\rangle+|10\rangle}{\sqrt{2}}$. Applying a CNOT gate afterward results in the entangled state $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$, where the state of each qubit cannot be described independently of the other.

3 Quantum Recurrent Neural Networks

In this section, we introduce the concepts about two QRNNs model, we start from the variational quantum circuits (VQCs) to the real applicable models.

3.1 Variational Quantum Circuits (VQCs)

Variational Quantum Circuits (VQCs) are the unique part of the architecture of quantum recurrent neural networks. The specific VQC components used in this context are structured in three main parts: an encoding layer, a variational layer, and a quantum measurement layer. Figure 1 shows an example of a VQC.

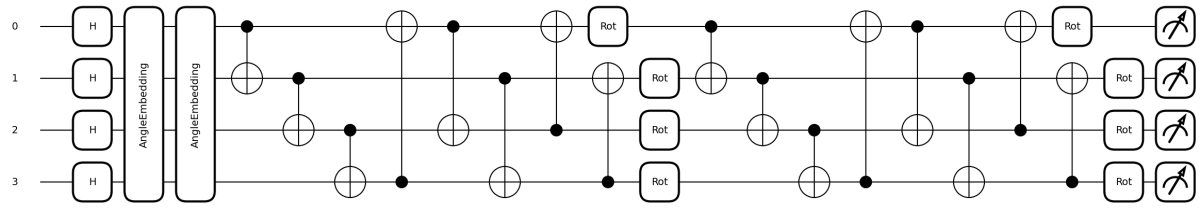


Figure 1: An example of VQC architecture with two layers of entanglements for QLSTM and QGRU.

The left-hand part of the figure is the encoding layer, which contains a Hadamard gate, two angle embedding layers working as R_y and R_z gates in the quantum concept. The middle part is the variational layer (entanglement layer). It is worth mentioning that the number of qubits (in this example is 4), and the number of variational layers (in this example is 2), can be modified to increase the capability to learn the data.

3.1.1 Encoding Layer

The encoding layer's primary function is to map classical data values into quantum amplitudes. It starts with initializing the circuit in the ground state and applying Hadamard gates to create an unbiased initial state. The state of an N-qubit quantum system can be represented as:

$$|\psi\rangle = \sum_{(q_1,q_2,\ldots,q_N)\in\{0,1\}^N} c_{q_1,q_2,\ldots,q_N}|q_1\rangle \otimes |q_2\rangle \otimes \ldots \otimes |q_N\rangle, \tag{8}$$

where $c_{q_1,\ldots,q_N} \in \mathbb{C}$ is the complex amplitude for each basis state and $\otimes$ stands for tensor product. Again, by the Born's rule, we know that:

$$\sum_{(q_1,q_2,\ldots,q_N)\in\{0,1\}^N} \|c_{q_1,\ldots,q_N}\|^2 = 1. \tag{9}$$

The encoding layer first transforms the input data into rotation angles to the rotation of each single qubit. We apply the Hadamard gate here and transform the initial state into a superposition, also called the unbiased state. After applying the Hadamard gate H N times (once to each qubit), we will obtain:

$$(H|0\rangle)^{\otimes N} = \left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)^{\otimes N}. \tag{10}$$

The resulting state after applying the Hadamard gate N times is a uniform superposition of all possible states of N qubits, which can be expressed as:

$$\frac{1}{\sqrt{2^N}}\left(|0\rangle^{\otimes N} + \cdots + |1\rangle^{\otimes N}\right). \tag{11}$$

Hence,

$$\frac{1}{\sqrt{2^N}}\sum_{i=0}^{2^N-1}|i\rangle. \tag{12}$$

Here, i is used as an index to sum over all possible states, where i represents the decimal equivalent of the binary numbers formed by the qubits. For example, for $N = 2$, the sum would run over $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$, which correspond to decimal 0, 1, 2, and 3, respectively.

The encoding process involves using two-angle encoding, where each data value is encoded to a qubit with a series of two gates, R_y and R_z. In this study, a template named *qml.templates.AngleEmbedding* is used to play the role of R_y and R_z.

The *qml.templates.AngleEmbedding* template effectively maps classical information onto the quantum states by applying specific rotation gates to each qubit in the system. This template can be customized for various aspects of the embedding, including selecting the axis of rotation R_x, R_y, or R_z. This adaptability makes the template well-suited for a broad spectrum of applications within the domain of quantum neural networks and other machine learning models.

After transforming the initial states into unbiased states, we will use $2N$ rotational angles from an N-dimensional input vector, $\vec{v} = (x_1, x_2, \ldots, x_N)$. For each component x_i of $\vec{v}$, in Chen's original paper, the two angles are calculated: $\theta_{i,1} = \arctan(x_i)$ for y-axis rotation and $\theta_{i,2} = \arctan(x_i^2)$ for z-axis rotation, where the rotations are effected through $R_y(\theta_{i,1})$ and $R_z(\theta_{i,2})$ gates, respectively.

Despite normalizing the input data before the encoding layer, this work introduces a distinct methodology. Rather than employing arctan functions, we select the sin and cos functions for the $R_y(\theta_{i,1})$ and $R_z(\theta_{i,2})$ gates, respectively. This entails setting $\theta_{i,1} = \sin(x_i)$ for the R_y gate and $\theta_{i,2} = \cos(x_i^2)$ for the R_z gate, diverging from conventional arctan applications. This method represents a preliminary attempt in practice and it can be improved in future work. More studies related to this aspect can be found in Mitarai's paper [**3**].

3.1.2 Variational Layer

The variational layer is the trainable part of the VQCs, consisting of parameterized unitary transformations. This section of the circuit includes multiple CNOT gates for qubit entanglement and unitary rotation gates controlled by some learnable parameters. It is worth mentioning that the variational layer can be repeated more than one time in practice to increase the number of parameters and the model's expressive capacity.

3.1.3 Quantum Measurement

Quantum measurement is used for extracting classical information from the quantum circuit. It involves measuring the qubits, which, due to the probabilistic nature of quantum systems, yield varying bit strings upon each measurement. The expectation value of an operator $\hat{O}$ for a state $|\psi\rangle$ is given by:

$$E[\hat{O}] = \langle\psi|\hat{O}|\psi\rangle. \tag{13}$$

The expectation values can be either calculated analytically in a quantum simulation or obtained through multiple samplings in practical quantum devices with specific noise models.

The VQC architecture described here is pivotal in the QLSTM and QGRU, as a quantum-enhanced approach to processing and learning from data.

3.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) particularly adept at learning from sequences of data. LSTMs are designed to overcome the limitations of traditional RNNs, especially issues related to long-term dependencies in data sequences. Figure 2 shows the structure of a single LSTM unit.

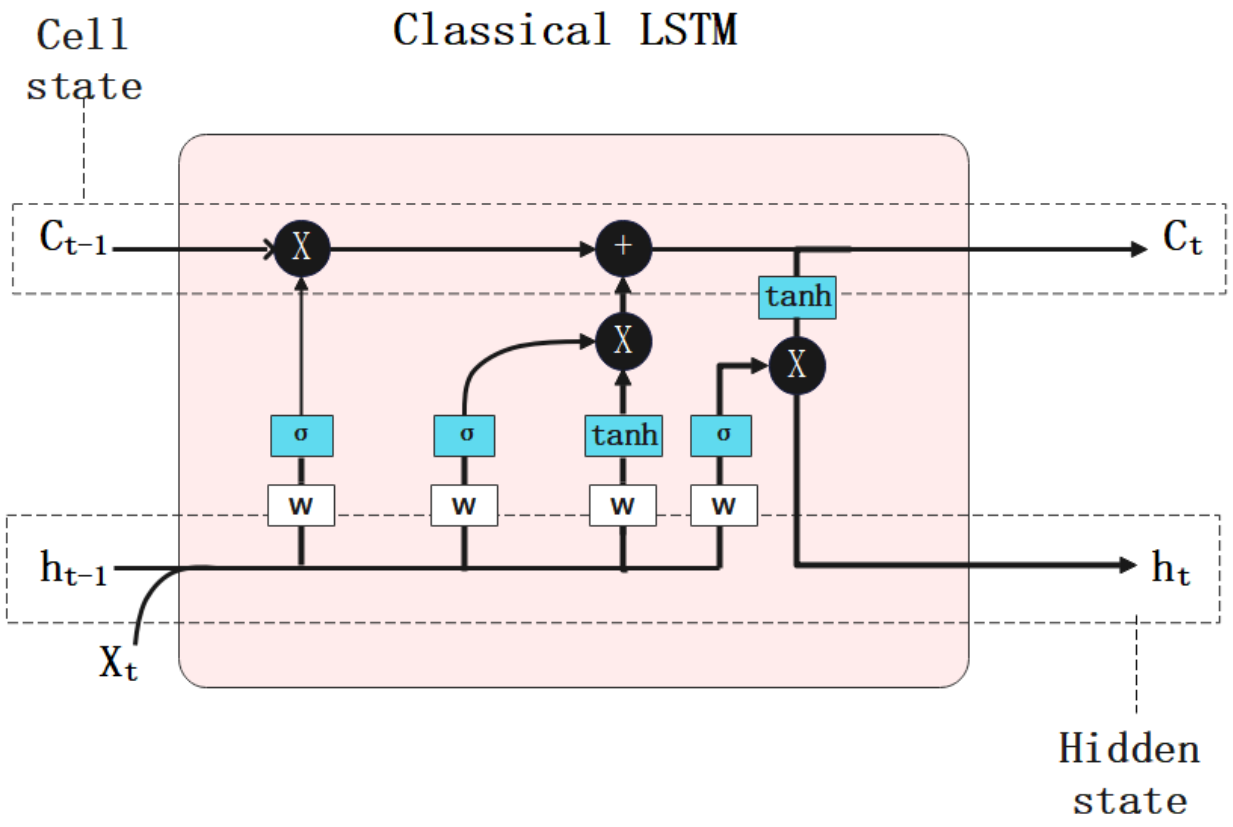


Figure 2: Structure of a single unit of classical LSTM.

An LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate. These components work together to regulate the flow of information into and out of the cell, and to decide which information to store and which to discard.

The key equations governing the operations within an LSTM unit are as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{14}$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{15}$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \tag{16}$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t, \tag{17}$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \tag{18}$$
$$h_t = o_t * \tanh(C_t). \tag{19}$$

Here, f_t, i_t, and o_t represent the activations of the forget, input, and output gates, respectively, at time t. σ denotes the sigmoid function, and $*$ represents element-wise multiplication. C_t is the cell state at time t, and h_t is the hidden state. W and b are the weights and biases associated with the respective gates and cell state updates.

3.3 Gated Recurrent Units (GRU)

Gated Recurrent Unit (GRU) is a variation of recurrent neural networks that aims to solve the vanishing gradient problem, similar to LSTM. GRUs simplify the architecture seen in LSTM by combining certain gates and states, which often results in more efficient training for certain types of problems. The structure of a single GRU is shown in Figure 3.

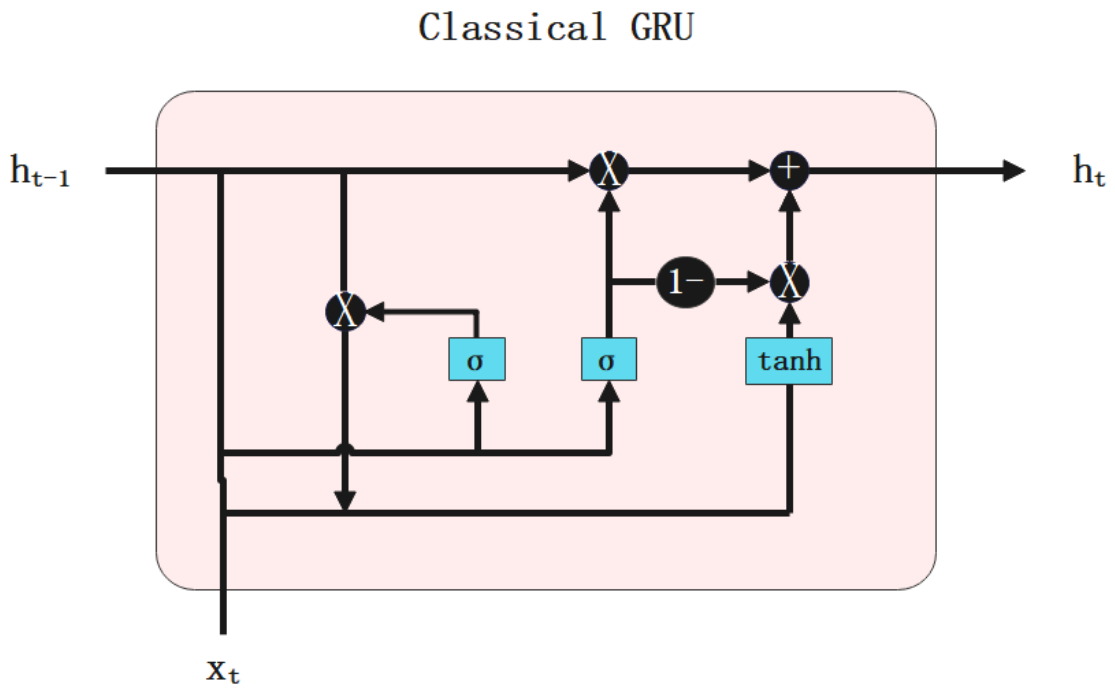


Figure 3: Structure of a single unit of classical GRU.

The GRU architecture is built around two gates: the update gate and the reset gate. These gates determine how much of the past information needs to be passed along to the future. The key equations defining a GRU are:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z), \tag{20}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r), \tag{21}$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t] + b), \tag{22}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t. \tag{23}$$

In Equations (20)–(23), z_t and r_t represent the activations of the update and reset gates at time t, respectively. h_t is the hidden state at time t, and $\tilde{h}_t$ is the candidate hidden state. W and b are the weights and biases associated with the respective gates and hidden state updates. The symbol $*$ denotes element-wise multiplication, and σ represents the sigmoid activation function.

GRUs provide an efficient alternative to LSTM and are particularly useful in modeling sequences where LSTM's complex structure may not be necessary.

We introduce a little bit about activation functions here, which are the *tanh* and *sigmoid* functions. On classical hardware, *tanh* and *sigmoid* functions are computed directly, introducing nonlinearity in neural networks crucial for learning complex patterns:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}, \tag{24}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{25}$$

3.4 Quantum Long Short-Term Memory (QLSTM)

Quantum Long Short-Term Memory (QLSTM) is a quantum-enhanced version of the traditional LSTM networks. QLSTM integrates VQCs into the LSTM architecture, aiming to leverage the computational advantages of quantum mechanics. The structure of a single QLSTM unit is shown in Figure 4.

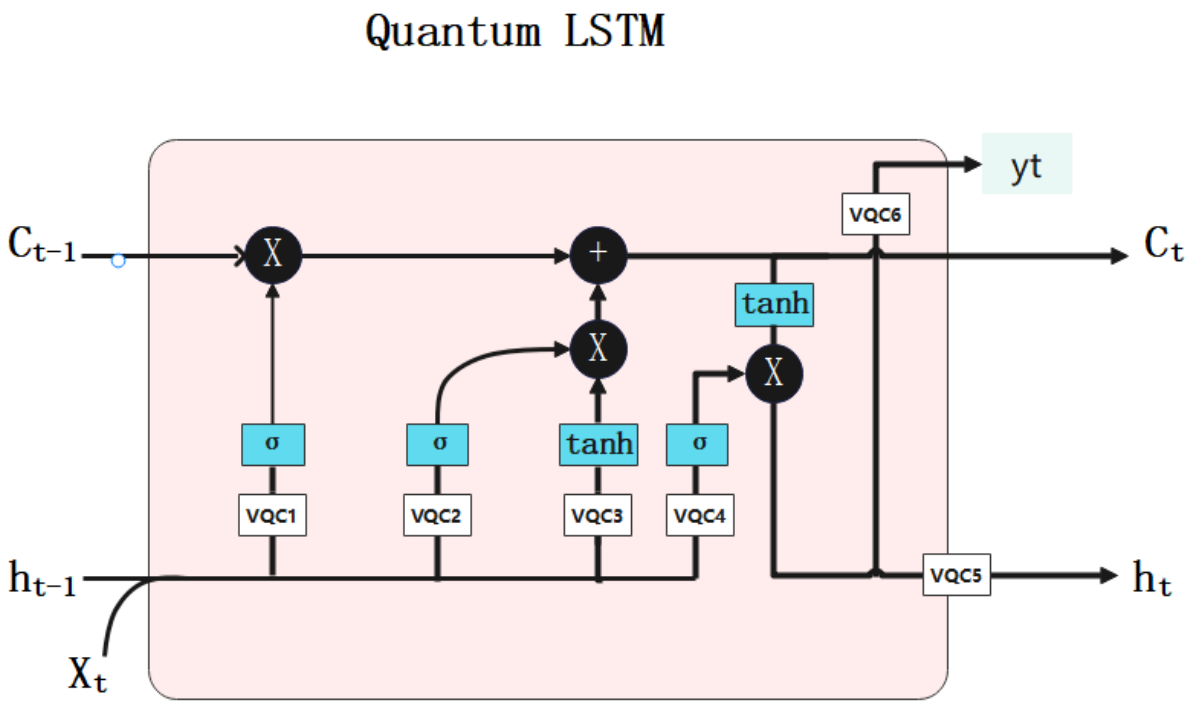


Figure 4: Structure of a single unit of QLSTM.

In a QLSTM, two key memory components are present: the hidden state h_t and the cell or internal state c_t. The functioning of a QLSTM cell can be mathematically described by the following equations:

$$f_t = \sigma(\text{VQC1}(v_t)), \tag{26}$$
$$i_t = \sigma(\text{VQC2}(v_t)), \tag{27}$$
$$\tilde{C}_t = \tanh(\text{VQC3}(v_t)), \tag{28}$$
$$c_t = f_t * c_{t-1} + i_t * \tilde{C}_t, \tag{29}$$
$$o_t = \sigma(\text{VQC4}(v_t)), \tag{30}$$
$$h_t = \text{VQC5}(o_t * \tanh(c_t)), \tag{31}$$
$$\tilde{y}_t = \text{VQC6}(o_t * \tanh(c_t)), \tag{32}$$
$$y_t = \text{NN}(\tilde{y}_t). \tag{33}$$

In Equations (26)–(33), σ represents the sigmoid function, and $*$ denotes element-wise multiplication. The input to the QLSTM cell at each time step is the concatenation v_t of the previous hidden state h_{t-1} and the current input vector x_t. The VQCs mentioned in the equations refer to Variational Quantum Circuits.

3.5 Quantum Gated Recurrent Unit (QGRU)

Quantum Gated Recurrent Unit (QGRU) represents an evolution of traditional GRU networks, integrating with VQCs. The structure of a single QGRU unit is shown in Figure 5.

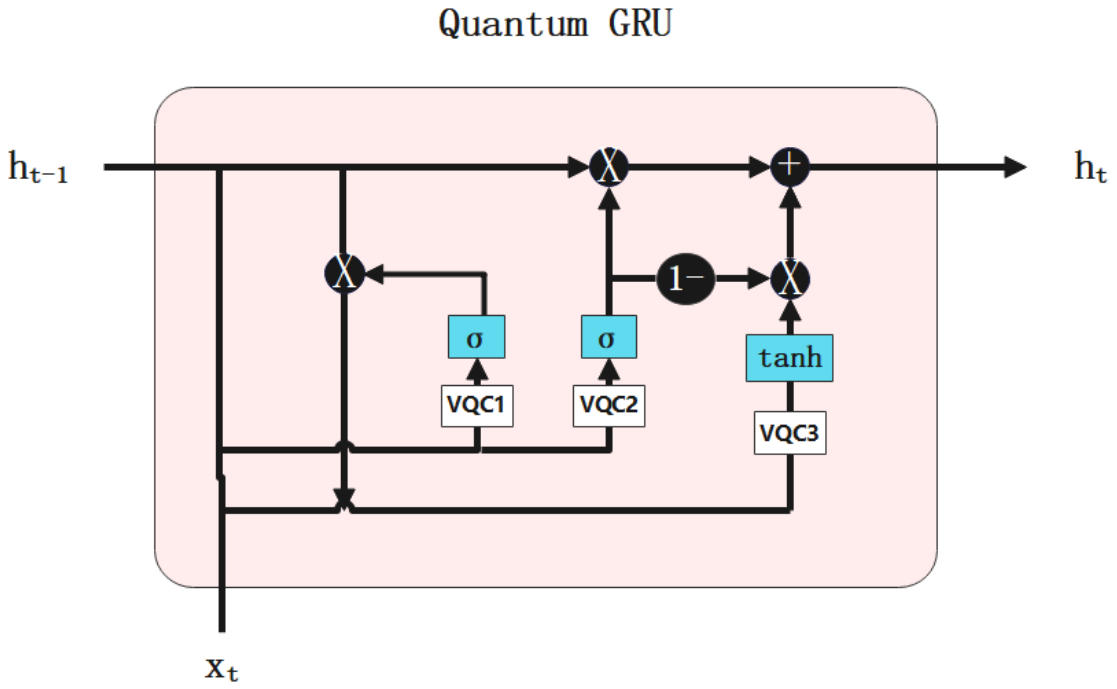


Figure 5: Structure of a single unit of QGRU.

A QGRU cell operates based on the following equations:

$$r_t = \sigma(\text{VQC1}(v_t)), \tag{34}$$
$$z_t = \sigma(\text{VQC2}(v_t)), \tag{35}$$
$$o_t = \text{cat}(x_t, r_t * H_{t-1}), \tag{36}$$
$$\tilde{H}_t = \tanh(\text{VQC3}(o_t)), \tag{37}$$
$$H_t = z_t * H_{t-1} + (1 - z_t) * \tilde{H}_t, \tag{38}$$
$$y_t = \text{NN}(H_t). \tag{39}$$

In Equations (34)–(39), r_t and z_t represent the reset and update gates of the QGRU at time t, respectively. H_t denotes the hidden state, and $\tilde{H}_t$ is the candidate hidden state. The input to the QGRU cell, v_t, is the concatenation of the previous hidden state H_{t-1} and the current input vector x_t. The VQCs are used to process the quantum aspects of the data.

4 Chapter 1: System of ODEs

In this chapter, we show the numerical results from system of ODEs, in this study we apply both QLSTM and QGRU to all the experiments, compared with their classical counterparts.

4.1 Hyperparameter Configuration

For the numerical experiments conducted in this study, specific hyperparameters were chosen to optimize the performance of the quantum models. Table 1 outlines the hyperparameter configuration used in the Van der Pol oscillator simulation and the simulation of two coupled damped harmonic oscillators:

Table 1: Hyperparameter configuration.

Hyperparameter	**QRNNs Value**	**RNNs Value**
Optimizer	RMSprop	Adam
Loss Function	MSE	MSE
Backend	default.qubit	-
Number of Qubits	4	-
Layer of Entanglements	4	-
Number of Data Points	250	250
Percentage of Train Set	67%	67%
Percentage of Test Set	33%	33%
Learning Rate	0.01	0.01

More specifically, we are using a single-layer model with hidden size 4, and the models are evaluating over 100 epochs. We are using the backend $default.qubit$ by PennyLane [**6**].

4.2 Van der Pol Oscillator Simulation

In the first experiment, we consider the Van der Pol oscillator, a classical example of a non-conservative oscillator with nonlinear damping. The oscillator is modeled by the following second-order differential equation:

$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x = 0. \tag{40}$$

The parameter μ, representing the nonlinearity and strength of the damping, is set to 1.0 in our simulations. This choice of μ provides a balance between linear and nonlinear dynamical behaviors, also making the system ideal for predicting a wide range of oscillatory patterns.

The initial conditions for the oscillator are chosen as $x_0 = 2.0$ and $y_0 = 0.0$, where x_0 and y_0 represent the initial position and velocity, respectively.

To numerically solve the Van der Pol differential equation, we convert it into a system of first-order equations:

$$\begin{aligned} \frac{dx}{dt} &= y, \\ \frac{dy}{dt} &= \mu(1 - x^2)y - x. \end{aligned} \tag{41}$$

The numerical solution is obtained over a time span of 0 to 50 s, discretized into 250 time steps.

Following the numerical solution of the differential equations, the resultant time series data of x and y are normalized and prepared for analysis. The data are reshaped to fit the input requirements of RNN models, enabling us to predict the Van der Pol oscillator's dynamics.

The behavior of the Van der Pol oscillator was examined through the LSTM, QLSTM, GRU, and QGRU models by evaluating their performance on the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) metrics.

Mean Absolute Error (MAE) is a measure of errors between paired observations. It is calculated as the average of the absolute differences between the predicted values and the actual values, disregarding the direction of the error. The MAE is given by the formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| . \tag{42}$$

Root Mean Square Error (RMSE) is a quadratic scoring rule that also measures the average magnitude of the error. It is the square root of the average of squared differences between

prediction and actual observation. The RMSE is given by the formula:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}, \tag{43}$$

where:

- n is the number of observations;
- y_i is the actual value of the i^{th} observation;
- $\hat{y}_i$ is the predicted value of the i^{th} observation.

Data from the numerical solution of the Van der Pol differential equations formed the basis for training and testing these models. A comparative analysis of the models' performance is summarized in Tables 2 and 3.

Table 2: Comparison of train and test MAE and RMSE for the state of x.

Model	**Train MAE**	**Test MAE**	**Train RMSE**	**Test RMSE**
LSTM	0.2601	0.2585	0.3002	0.2986
QLSTM	0.1411	0.1397	0.1648	0.1641
GRU	0.1597	0.1591	0.1845	0.1850
QGRU	0.0868	0.0902	0.1013	0.1031

Table 3: Comparison of train and test MAE and RMSE for the state of y.

Model	**Train MAE**	**Test MAE**	**Train RMSE**	**Test RMSE**
LSTM	0.3224	0.3294	0.3737	0.3828
QLSTM	0.1959	0.1851	0.2498	0.2384
GRU	0.3336	0.3375	0.4319	0.4384
QGRU	0.1473	0.1500	0.1931	0.1943

Table 2 shows that quantum-based models exhibit superior predictive performance on x value over classical models. Among the models compared, the GRU models also outshine the LSTM models in accuracy and QGRU model gives us the most accurate predictions.

Table 3 compares model performances on the y value. It is very interesting to see that the models produced more error on the prediction of the y values compared to the x values. Moreover, GRU models surpass LSTM models in terms of results, with the QGRU model achieving the highest accuracy among the evaluated models.

The example predictive results over epochs by the different models are shown in Figures 6-10.

It is observable that the Van der Pol oscillator exhibits periodic behavior. It can be seen that all the models caught the patterns well. It can be observed that the QRNN models, especially, learn significantly faster than the classical RNN models from the comparison through epoch 5. Again, the QGRU model shows great capability in learning the dynamics.

By comparing the training and test loss over the epochs, the QRNN models show more stable decrease than the classical RNN models and the QGRU converges faster than QLSTM.

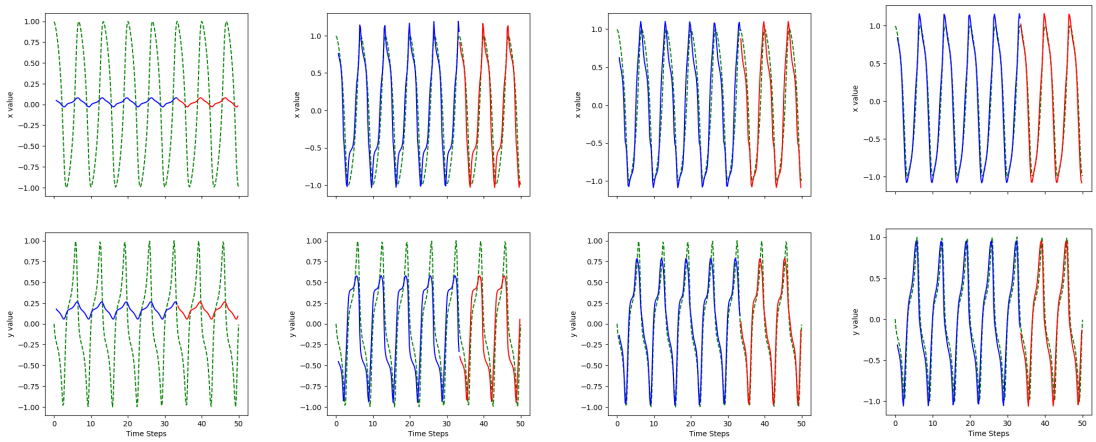


Figure 6: LSTM predictions over 100 epochs for the Van der Pol oscillator. Up: x; down: y; epoch: 5, 50, 70, 100; green dashed line: actual values; blue solid line: training predictions; red solid line: testing predictions.

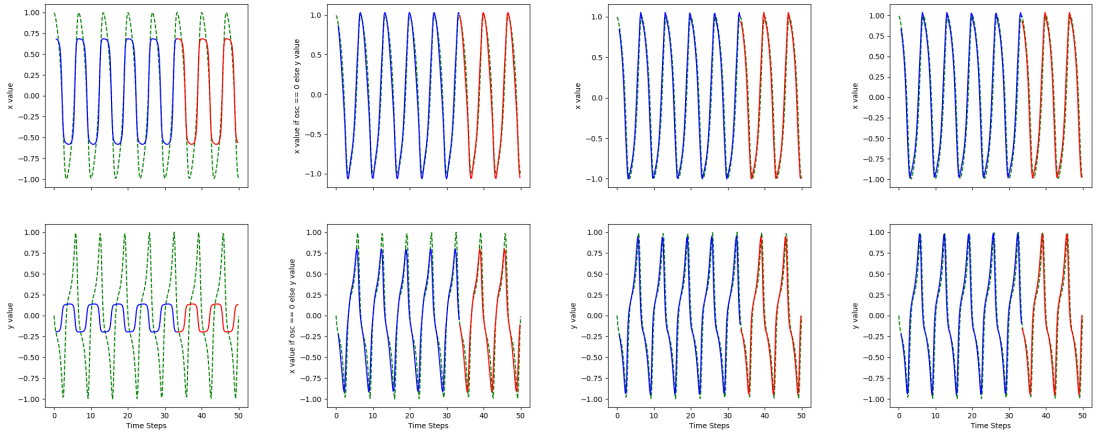


Figure 7: QLSTM predictions over 100 Epochs for the Van der Pol oscillator. Up: x; down: y; epoch: 5, 50, 70, 100; green dashed line: actual values; blue solid line: training predictions; red solid line: testing predictions.

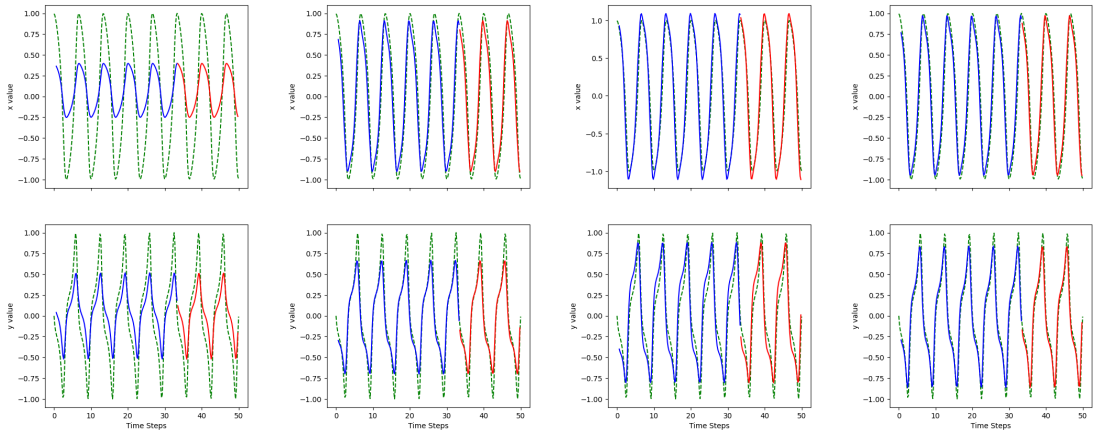


Figure 8: GRU predictions over 100 Epochs for the Van der Pol oscillator. Up: x; down: y; epoch: 5, 50, 70, 100; green dashed line: actual values; blue solid line: training predictions; red solid line: testing predictions.

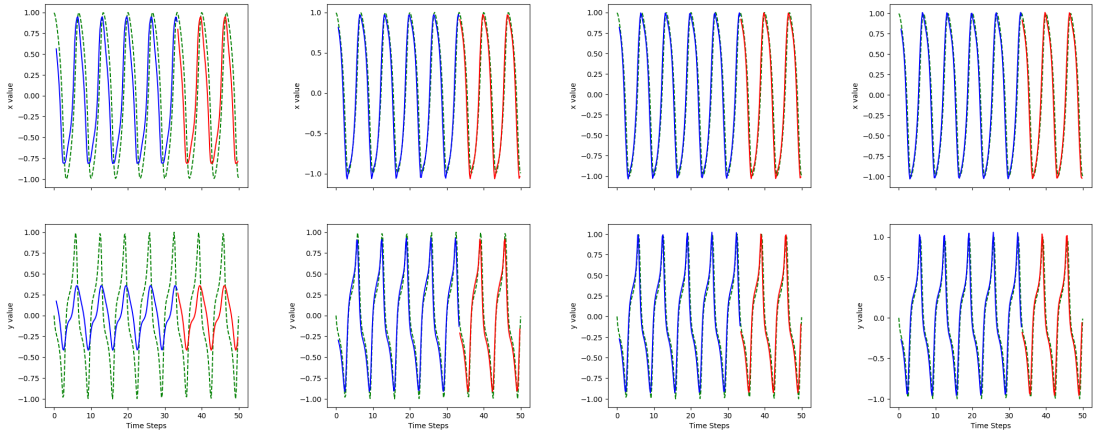


Figure 9: QGRU predictions over 100 Epochs for the Van der Pol oscillator. Up: x; down: y; epoch: 5, 50, 70, 100; green dashed line: actual values; blue solid line: training predictions; red solid line: testing predictions.

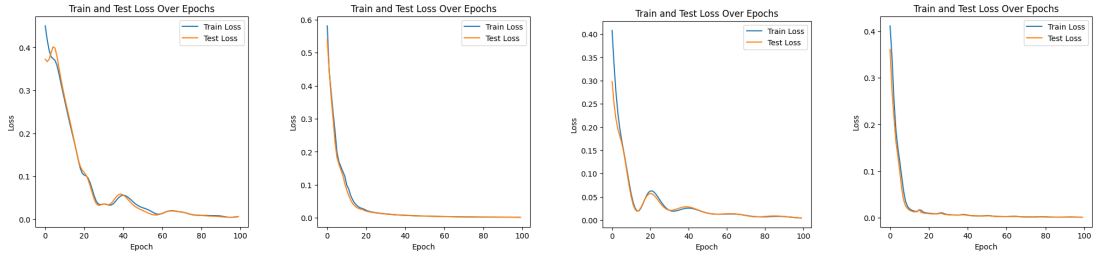


Figure 10: Experiment 1: Train and test losses for the models over 100 epochs. From left to right: LSTM, QLSTM, GRU, QGRU.

4.3 Two Coupled Damped Harmonic Oscillators Simulation

In this experiment, we study the capability the QRNN in learning the patterns in the system of two coupled harmonic oscillators. It is worth pointing out that this experiment actually is an extension of the prediction on a single damped harmonic oscillator by Chen [**34**, **35**]. This experiment extends our understanding not only from a single oscillator, but physical systems by introducing interactions between oscillatory motions. Such systems are paramount in numerous fields, including physics and engineering. The governing differential equations for two coupled damped harmonic oscillators, representing an extended model of the single oscillator, are as follows:

$$\begin{aligned}\frac{d^2\theta_1}{dt^2} &= -\frac{b_1}{m_1}\frac{d\theta_1}{dt} - \frac{g}{l_1}\sin(\theta_1) + \frac{k_c}{m_1}(\theta_2 - \theta_1),\\ \frac{d^2\theta_2}{dt^2} &= -\frac{b_2}{m_2}\frac{d\theta_2}{dt} - \frac{g}{l_2}\sin(\theta_2) + \frac{k_c}{m_2}(\theta_1 - \theta_2).\end{aligned} \tag{44}$$

Here, θ_1 and θ_2 represent the angular displacements of the first and second pendulums, respectively. The constants $g = 9.81$ m/s^2 (gravitational acceleration), $b_1 = b_2 = 0.15$ (damping factors), $l_1 = l_2 = 1.0$ m (lengths of the pendulums), $m_1 = m_2 = 1.0$ kg (masses of the pendulums), and $k_c = 0.05$ (coupling constant) define the system's characteristics. The initial conditions are set with angular displacements $\theta_1 = \theta_2 = 0$ and angular velocities $\dot{\theta}_1 = 3.0$, $\dot{\theta}_2 = 0.0$ rad/s.

Just like the Van der Pol example, we present a comparative analysis of the models' performance in predicting the dynamic behavior of both oscillators (again, with MAE and RMSE), which are summarized in Tables 4 and 5.

Table 4: Comparison of train and test MAE and RMSE for Oscillator 1.

Model	**Train MAE**	**Test MAE**	**Train RMSE**	**Test RMSE**
LSTM	0.5467	0.4371	0.7761	0.4922
QLSTM	0.5284	0.4763	0.6987	0.5374
GRU	0.3648	0.4396	0.4499	0.4941
QGRU	0.2585	0.2411	0.3587	0.2701

Table 5: Comparison of train and test MAE and RMSE for Oscillator 2.

Model	**Train MAE**	**Test MAE**	**Train RMSE**	**Test RMSE**
LSTM	0.2190	0.3597	0.2814	0.4248
QLSTM	0.1244	0.2591	0.1607	0.2885
GRU	0.1160	0.0858	0.1483	0.1006
QGRU	0.0794	0.0482	0.0949	0.0602

Like the results from the first experiment, in Tables 4 and 5, all the models successfully learn the dynamics, but the QGRU shows excellent predictive results, especially on the prediction of Oscillator 2. We noticed, first, that the predictions on Oscillator 2 are more accurate than Oscillator 1. Second, we noticed that the GRU models outshine LSTM models in general.

The example predictive results over the 100 epochs by the different models are shown in

Figures 11-15.

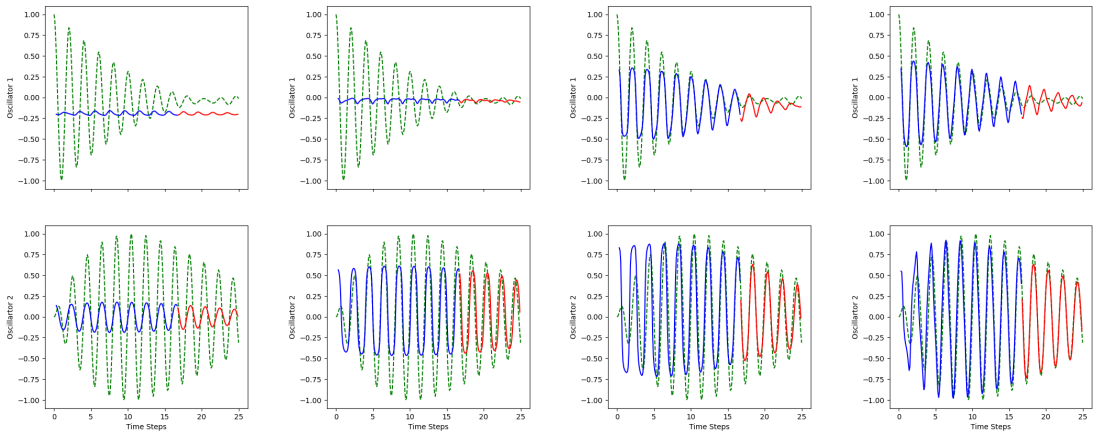


Figure 11: LSTM predictions over 100 epochs. Up: Oscillator 1; down: Oscillator 2; epoch: 5, 50, 70, 100; green dashed line: actual values; blue solid line: training predictions; red solid line: testing predictions.

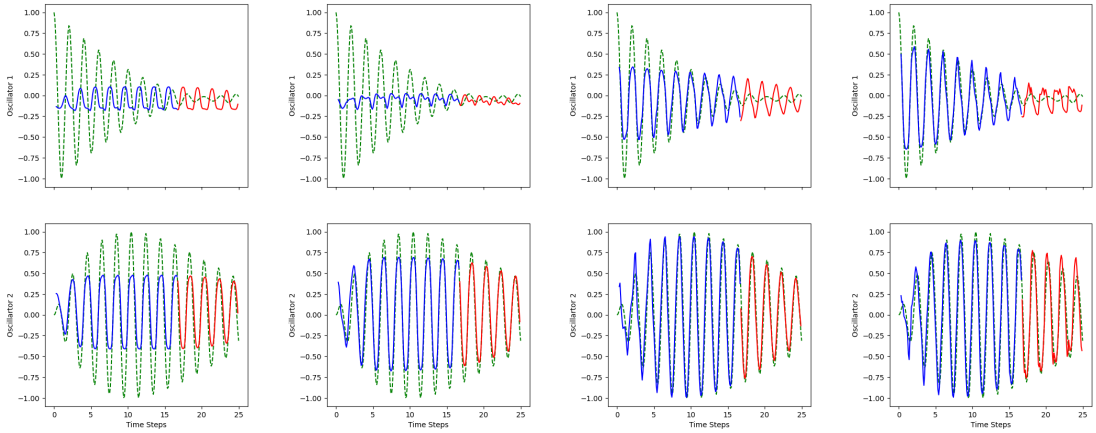


Figure 12: QLSTM Predictions over 100 epochs. Up: Oscillator 1; down: Oscillator 2; epoch: 5, 50, 70, 100; green dashed line: actual values; blue solid line: training predictions; red solid line: testing predictions.

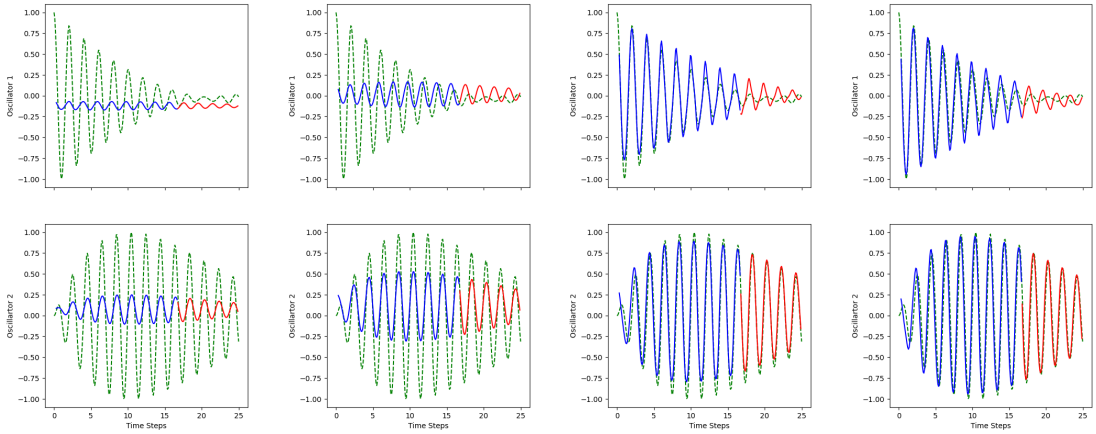


Figure 13: GRU Predictions over 100 epochs. Up: Oscillator 1; down: Oscillator 2; epoch: 5, 50, 70, 100; green dashed line: actual values; blue solid line: training predictions; red solid line: testing predictions.

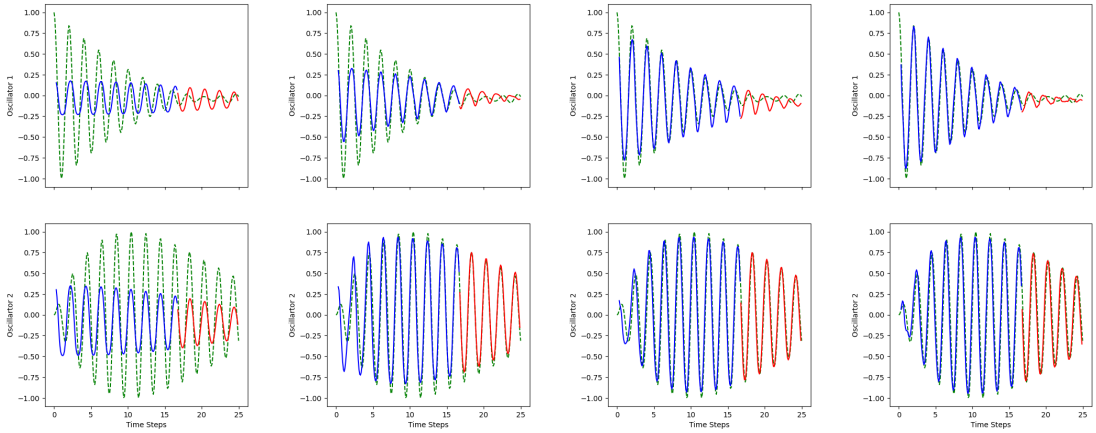


Figure 14: QGRU Predictions over 100 epochs. Up: Oscillator 1, down: Oscillator 2 epoch: 5, 50, 70, 100; green dashed line: actual values; blue solid line: training predictions; red solid line: testing predictions.

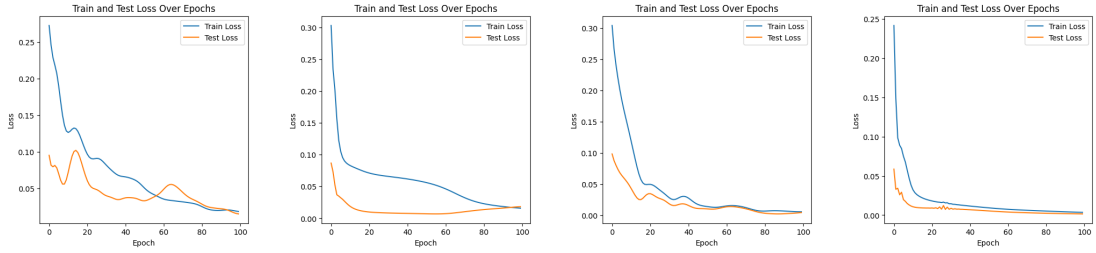


Figure 15: Experiment 2: train and test losses for the models over 100 epochs. From left to right: LSTM, QLSTM, GRU, QGRU.

Similar to the Van der Pol case, the GRU models converge faster than the LSTM models and the quantum-based models learn the pattern faster than the classical models (which can be seen by the comparison on the results on epoch 5). The quantum-based RNNs also show stabler decrease in loss. Here, we observe several points of interest: Firstly, there is a minor increase in the test loss in LSTM models from epoch 80 to 100, potentially indicative of mild overfitting. Although adjusting the learning rate could mitigate this, we maintain a constant learning rate to achieve a straightforward comparison between the models. Secondly, spikes are observed in the LSTM model's test loss. Furthermore, at epoch 5, the LSTM model exhibits an undershot in performance, which is not seen in the results of the other models.

From the previous two numerical experiments, the QRNN models demonstrate exceptional capability in learning patterns by system dynamics. In the last experiment, we apply the models to chaotic systems using a different approach.

4.4 System of Lorenz Equations

The Lorenz equations, fundamental in chaos theory, model the dynamics of atmospheric convection and are characterized by their chaotic nature for certain parameter values. The system is described by the following set of differential equations:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x), \\ \frac{dy}{dt} &= x(\rho - z) - y, \\ \frac{dz}{dt} &= xy - \beta z. \end{aligned} \tag{45}$$

In Equation 45, x, y, and z represent the system states, and the parameters σ, ρ, and β are crucial for the system's behavior. The chosen values $\sigma = 10.0$, $\beta = \frac{8}{3}$, and $\rho = 28.0$ are known to induce chaotic dynamics in the Lorenz system. These parameters represent the Prandtl number, normalized Rayleigh number, and certain physical dimensions of the convective cells, respectively. Their specific values make the system a classic example of chaos, making it a compelling subject for studying the predictive capabilities of quantum-enhanced and classical neural networks.

Data Preparation and Hyperparameter Configuration

Unlike the previous two experiments, we generated the data differently in the application of RNN models on Lorenz equations. We first randomly generated 10 datasets; we used 67% of them for training and the remaining 33% for testing. The shapes of the training and test datasets are as follows:

- Training dataset shapes:
  - Features: `torch.Size([3120, 10, 3])`;
  - Labels: `torch.Size([3120, 3])`.
- Testing dataset shapes:
  - Features: `torch.Size([780, 10, 3])`;
  - Labels: `torch.Size([780, 3])`.

Regarding the hyperparameter aspect, the previous two experiments demonstrate that QRNN models rapidly catch the trend of the dataset. Consequently, we aim to explore more about these models' proficiency in learning features with shorter sequence lengths, reduced data points, and fewer epochs.

We pick sequence length 10 for all the models and all the models will be running over 20 epochs instead of the 100 epochs in the previous two experiments.

Hyperparameter configurations for LSTM, QLSTM, GRU, and QGRU models are presented in Table 6:

Table 6: Hyperparameter configuration for classic models and quantum models.

Hyperparameter	**Quantum**	**Classic**
Optimizer	RMSprop	Adam
Loss Function	MSE	MSE
Backend	default.qubit	-
Number of Qubits	4	-
VQC Layer	4	-
Percentage of Train Set	67%	67%
Percentage of Test Set	33%	33%
Epochs	20	20
Learning Rate	0.01	0.01

4.5 Results Analysis

The performances of each model on the Lorenz system data are summarized in Tables 7 and 8. These tables detail the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) for training and testing datasets across three dimensions.

Table 7: Mean Absolute Error (MAE) for each model on test set.

Model	**X**	**Y**	**Z**
LSTM	2.0361	2.1232	2.2119
QLSTM	0.8820	0.9473	1.0206
GRU	1.1684	1.1719	1.1710
QGRU	0.4864	0.4723	0.4555

Table 8: Root Mean Square Error (RMSE) for each model on test set.

Model	**X**	**Y**	**Z**
LSTM	2.1736	2.2718	2.3714
QLSTM	1.2234	1.2723	1.3348
GRU	1.1783	1.1748	1.1740
QGRU	0.4971	0.4846	0.4745

According to the error metrics, quantum-based RNNs exhibit a marked increase in accuracy,

particularly the QGRU model in comparison to the classical GRU. Moreover, the LSTM model demonstrates notably poorer performance compared to the other models, potentially caused by its slower rate of convergence across epochs (again, only 20 epochs were used in this case).

To further present the models' predictions, we visualize their predictive outcomes across three dimensions, the trajectories and the losses over the epochs, in Figures 16-24.

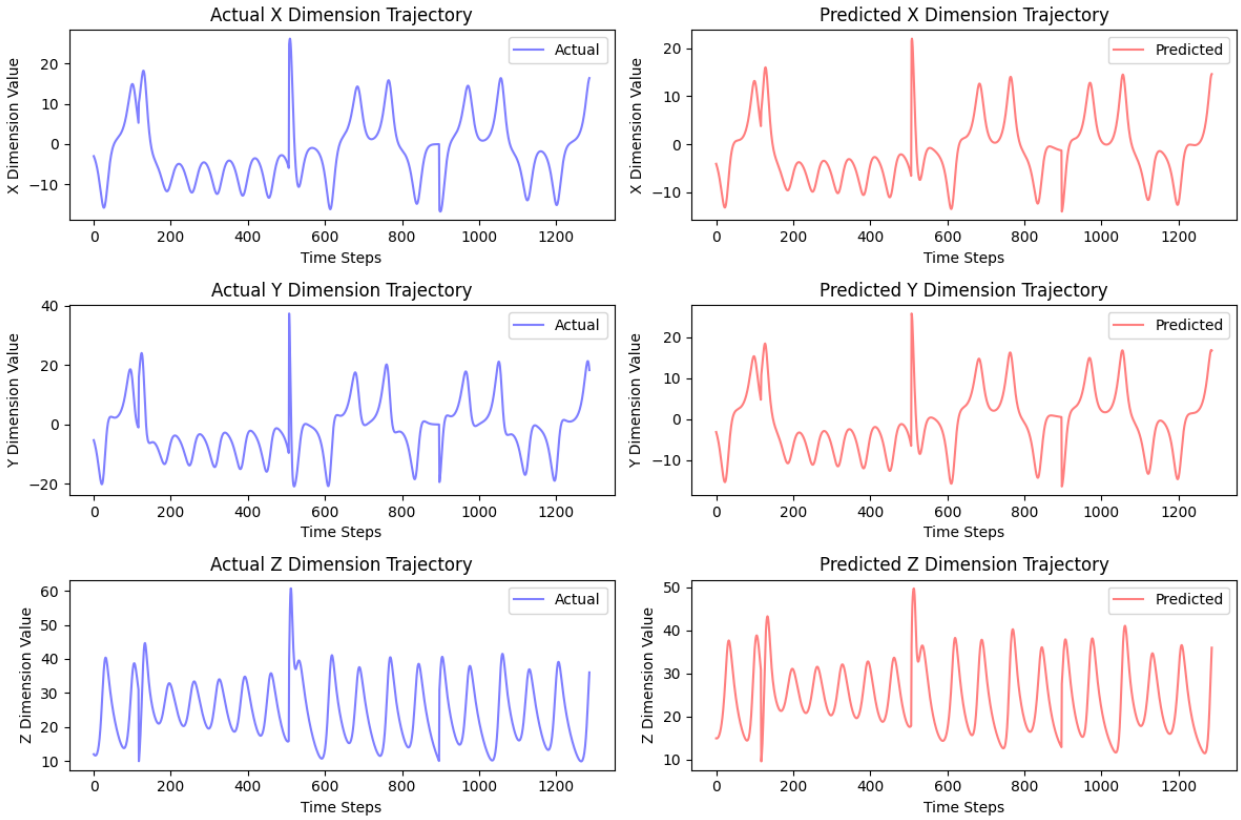


Figure 16: LSTM predictions for dimensions X (**up**), Y (**middle**), and Z (**bottom**).

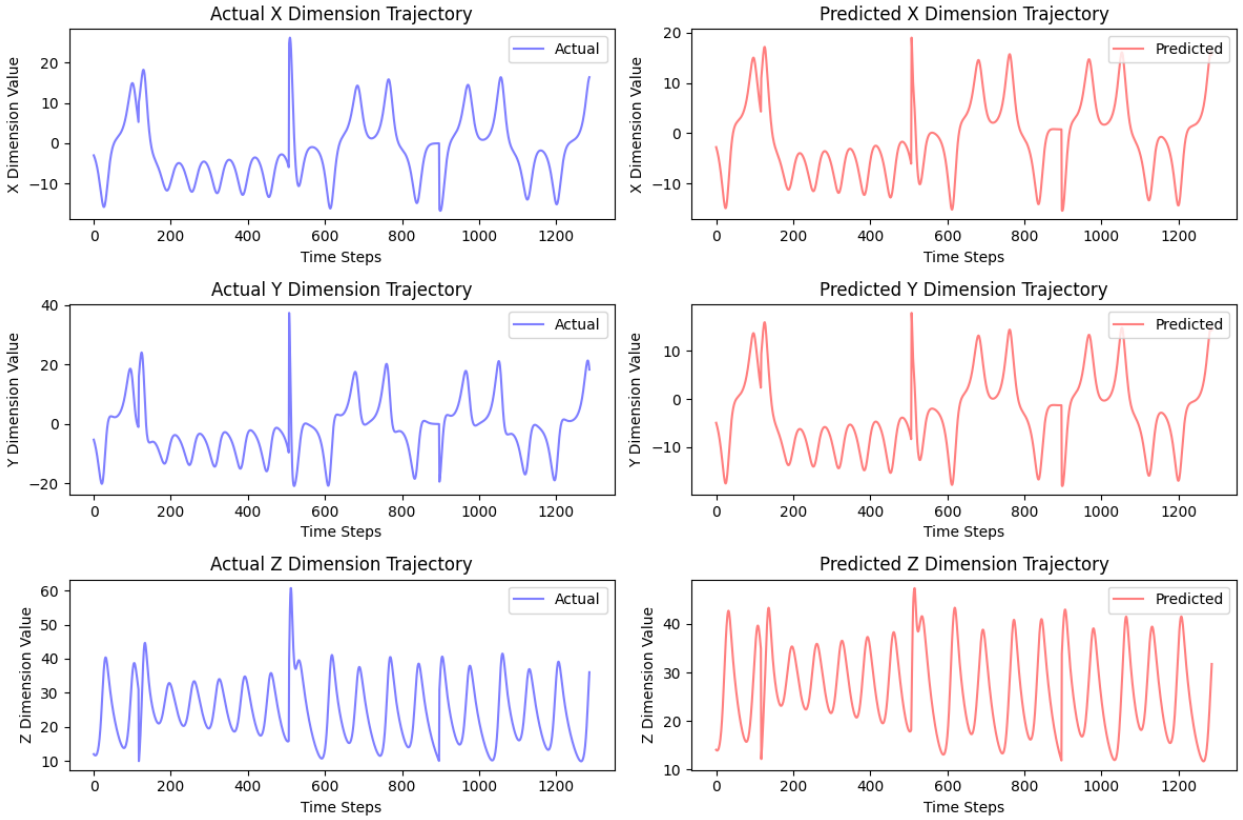


Figure 17: QLSTM predictions for dimensions X (**up**), Y (**middle**), and Z (**bottom**).

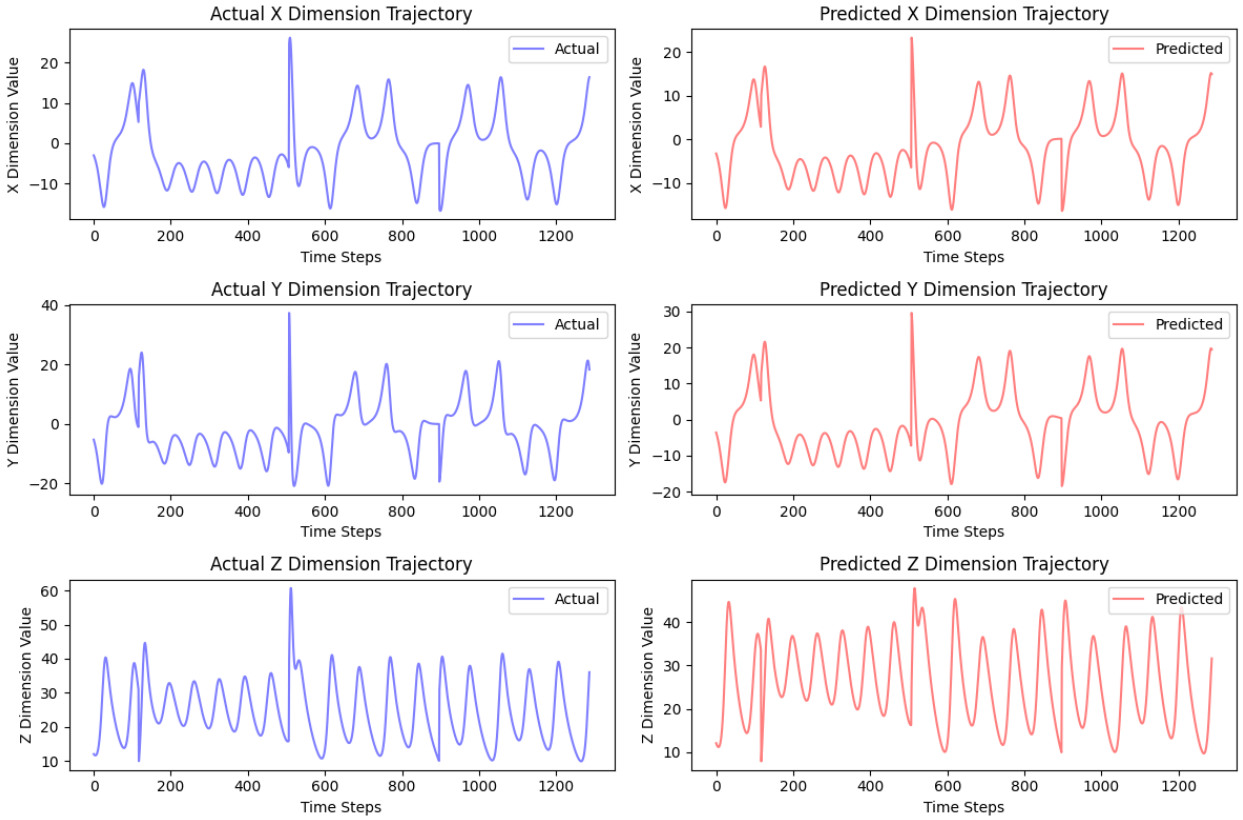


Figure 18: GRU predictions for dimensions X (**up**), Y (**middle**), and Z (**bottom**).

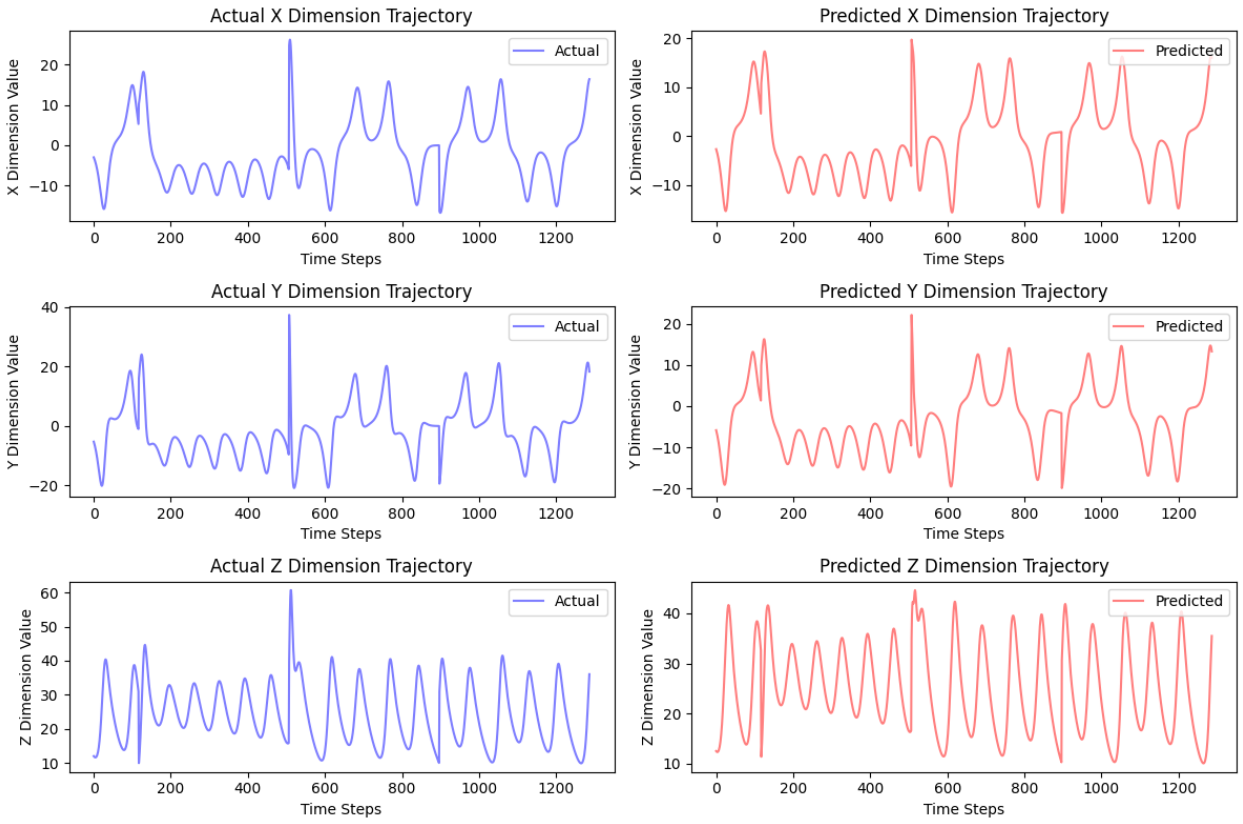


Figure 19: QGRU predictions for dimensions X (**up**), Y (**middle**), and Z (**bottom**).

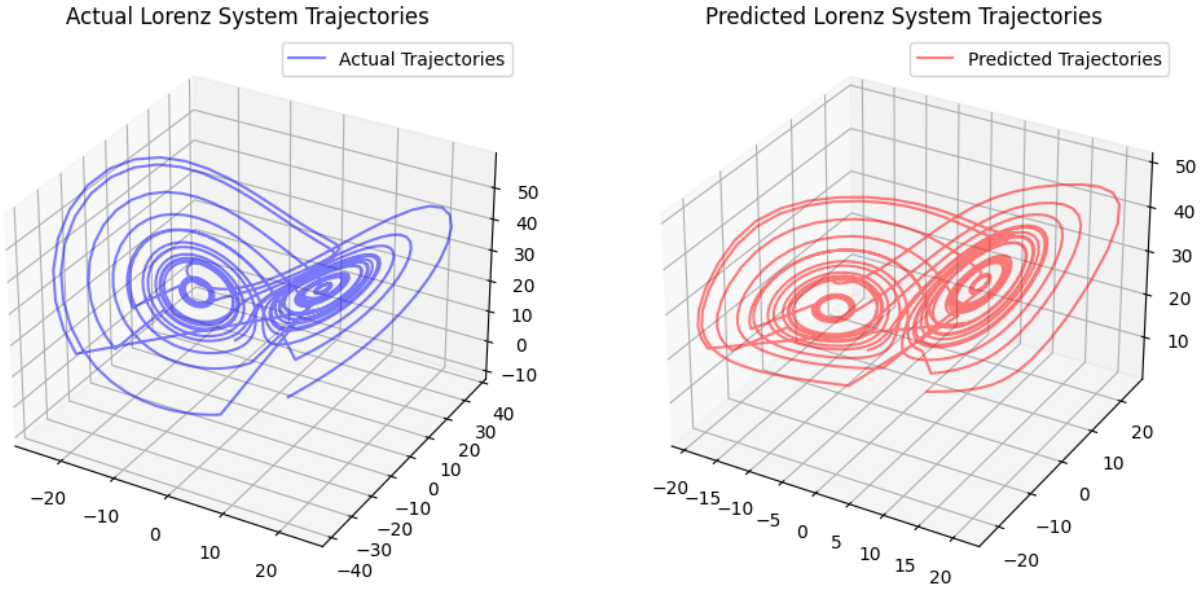


Figure 20: Predicted Lorenz system trajectory by LSTM model.

Firstly, regarding to the three-dimensional predictions, it is evident that all models correctly capture the dataset's trend. Nonetheless, a closer look at the figure scales reveals difficulties in predicting the spike at the figure's center for all the models (for LSTM in y and z dimensions, QLSTM in z, GRU in y and z, and QGRU in z). We believe this challenge could be solved by increasing the number of epochs, though the performance is deemed satisfactory for the current study.

Furthermore, the QGRU outperforms other models in trajectory plotting, showing a consistent decline in both training and test losses across epochs. In contrast, the classical RNNs display slight fluctuations in their loss plots. Although a spike is observed in the QLSTM's loss plot, quantum-based RNNs, in general, tend to show a smoother and more rapid reduction in losses than the classical models.

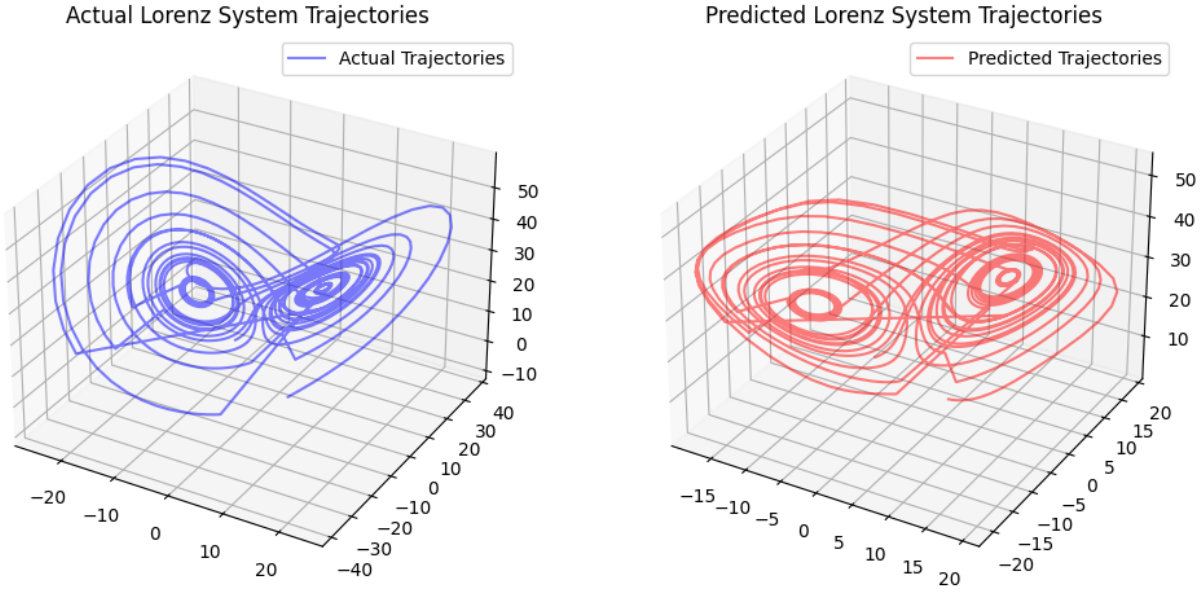


Figure 21: Predicted Lorenz system trajectory by QLSTM model.

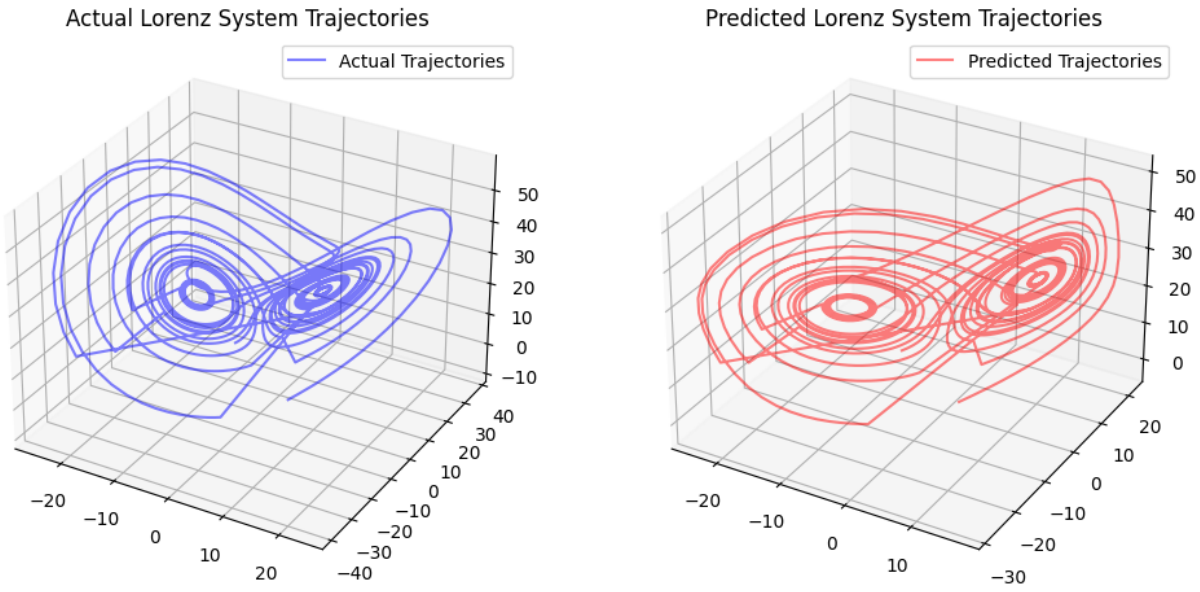


Figure 22: Predicted Lorenz system trajectory by GRU model.

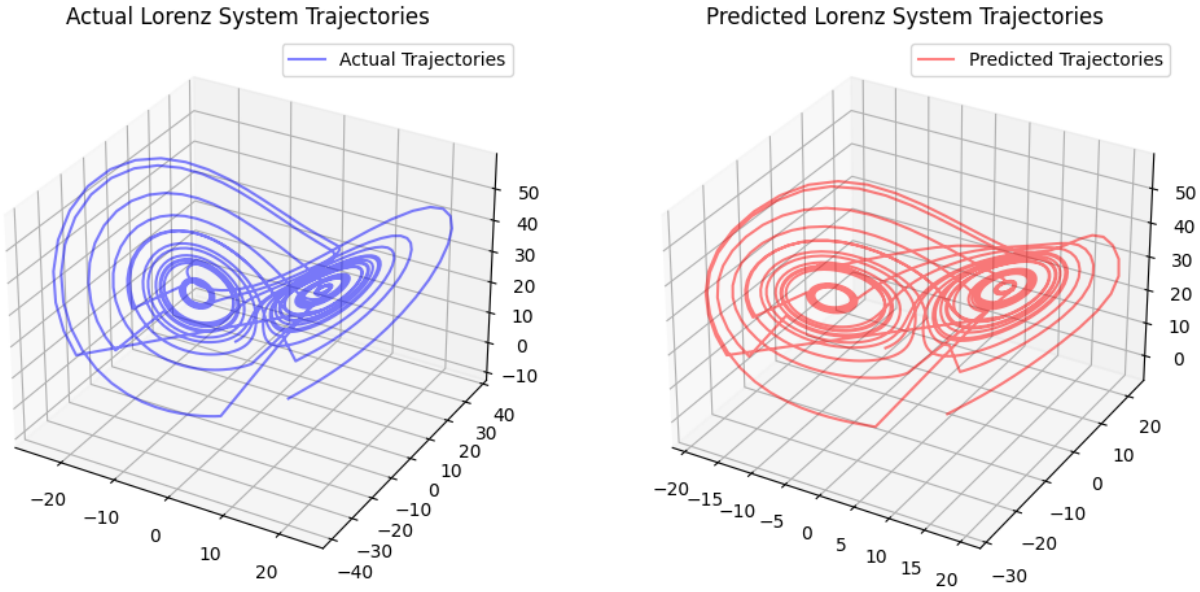


Figure 23: Predicted Lorenz system trajectory by QGRU model.

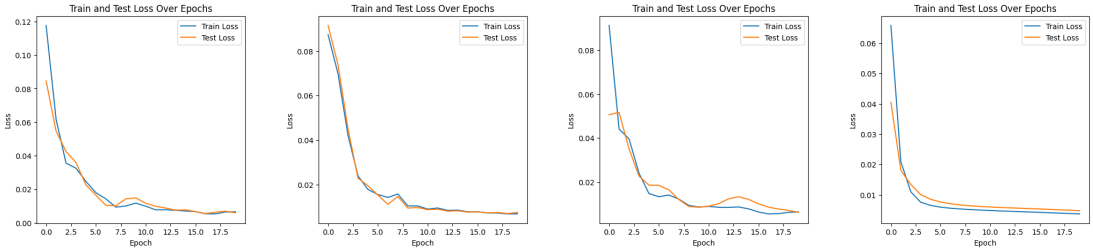


Figure 24: Experiment 3: Train and test losses for the models over 20 epochs. From left to right: LSTM, QLSTM, GRU, QGRU.

5 Chapter 2: Stiff Systems

In this chapter, we apply the QLSTM model to stiff ODE systems.

5.1 Introduction of Stiff Systems

In the context of ordinary differential equations (ODEs), especially stiff ODEs, the advantages of QLSTM networks become particularly relevant. Stiff ODEs are characterized by the presence of solutions with widely varying timescales, posing significant challenges for traditional numerical solvers that often require extremely small time steps to maintain stability. Recent advancements in machine learning have shown that Recurrent Neural Networks (RNNs) can effectively model the dynamics of such systems [**7**], [**8**], [**9**]. In particular, Echo State Networks (ESNs) have been successfully applied to chaotic time series prediction [**10**], demonstrating the potential of RNNs in capturing complex temporal patterns. The neural stiff ordinary differential equations [**11**] have further explored the application of neural networks to stiff ODEs, highlighting the potential of such hybrid approaches in addressing the unique challenges posed by these systems.

A system of ODEs is termed stiff when it exhibits widely varying timescales, which leads to rapid changes in some components while others evolve slowly. This discrepancy makes numerical integration challenging because explicit methods require extremely small time steps to maintain stability. Implicit methods, although more stable, are computationally intensive and complex to implement. Modeling stiff ODEs with neural networks (NN) introduces further complications due to the difficulty in capturing intricate dynamics and ensuring stable learning. To effectively develop neural network models for predicting unseen values in stiff

ODE systems, the model must strike a balance between fitting the training set and maintaining predictive accuracy on the test set. Overfitting to the training set can lead to poor predictive performance on the test set, highlighting the importance of designing models that generalize well beyond the data they were trained on.

Consider a system of ODEs given by:

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y}, t), \tag{46}$$

where $\mathbf{y}$ is the vector of dependent variables, and $\mathbf{f}$ is a vector-valued function describing the rate of change of $\mathbf{y}$ with respect to time t. Stiffness is characterized by the presence of eigenvalues with vastly different magnitudes. The Jacobian matrix of the system, which provides insight into the local behavior of the system, is defined as:

$$\mathbf{J} = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}, \tag{47}$$

where $\mathbf{J}$ is the matrix of partial derivatives of $\mathbf{f}$ with respect to $\mathbf{y}$. The eigenvalues λ_i of the Jacobian matrix $\mathbf{J}$ determine the system's stiffness. The stiffness ratio, R, is defined as:

$$R = \frac{|\lambda_{\max}|}{|\lambda_{\min}|}, \tag{48}$$

where $|\lambda_{\max}|$ and $|\lambda_{\min}|$ are the absolute values of the largest and smallest eigenvalues, respectively. A high stiffness ratio (usually greater than 1000) indicates severe stiffness, necessitating specialized solvers such as implicit methods or advanced approaches.

5.2 Little Adjustments on LSTM

Additionally, in the experiment in this chapter, we introduce another activation function, the Swish function (SiLU), which is a newer activation function defined as:

$$\text{swish}(x) = x \cdot \sigma(\beta x) = \frac{x}{1 + e^{-\beta x}}, \tag{49}$$

where β is either a constant or a trainable parameter depending on the model. When

$\beta = 1$, the function becomes equivalent to the Sigmoid Linear Unit (SiLU) [**12**], which was first proposed alongside the Gaussian Error Linear Unit (GELU) in 2016. The SiLU was later rediscovered in 2017 as the Sigmoid-weighted Linear Unit (SiL) function used in reinforcement learning [**13**]. The Swish function was then rediscovered over a year after its initial discovery, originally proposed without the learnable parameter β, implying β implicitly equaled 1. The Swish paper was subsequently updated to include the learnable parameter β, although researchers typically set $\beta = 1$ and do not utilize the learnable parameter β [**14**]. For $\beta = 0$, the function reduces to the scaled linear function $f(x) = x/2$. As $\beta \to \infty$, the sigmoid component approaches a 0-1 function pointwise, making Swish approach the Rectified Linear Unit (ReLU) function pointwise. Thus, Swish can be viewed as a smoothing function that nonlinearly interpolates between a linear function and the ReLU function.

The derivative of the Swish function is given by:

$$\text{swish}'(x) = \sigma(\beta x) + x \cdot \sigma'(\beta x) = \sigma(\beta x) + x \cdot \sigma(\beta x)(1 - \sigma(\beta x)), \tag{50}$$

The hyperbolic tangent (tanh) function, which outputs values in the range $(-1, 1)$, has been widely used due to its zero-centered output that aids in faster convergence during training. However, the Swish function has demonstrated several advantages over tanh. Swish provides a smoother, non-monotonic activation curve that facilitates better gradient flow through the network, reducing the likelihood of vanishing or exploding gradients. Additionally, Swish retains a portion of negative inputs, which enhances the neural network's expressiveness and capacity to model complex patterns. Some studies have shown that Swish often outperforms traditional activation functions like tanh and ReLU in various deep learning tasks [**14**], [**15**], making it a promising alternative for use in QLSTM networks.

Therefore, in this study, we replace the tanh function with the Swish function. The structure of a single unit of QLSTM with activation function Swish is shown in Figure 25:

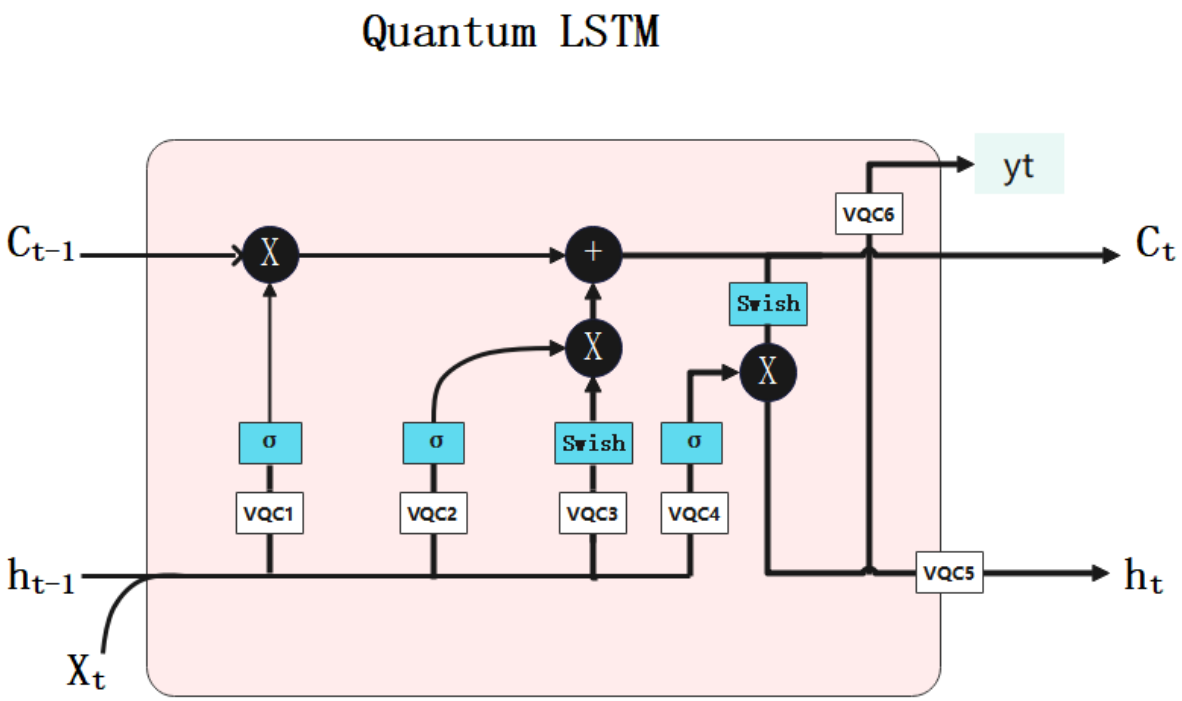


Figure 25: Structure of a single unit of QLSTM with the Swish function.

In a single QLSTM, two key memory components are present: the hidden state h_t and the cell or internal state c_t. The functioning of a QLSTM cell can be mathematically described as follows:

$$f_t = \sigma(\text{VQC1}(v_t)), \tag{51}$$
$$i_t = \sigma(\text{VQC2}(v_t)), \tag{52}$$
$$\tilde{C}_t = swish(\text{VQC3}(v_t)), \tag{53}$$
$$c_t = f_t * c_{t-1} + i_t * \tilde{C}_t, \tag{54}$$
$$o_t = \sigma(\text{VQC4}(v_t)), \tag{55}$$
$$h_t = \text{VQC5}(o_t * swish(c_t)), \tag{56}$$
$$\tilde{y}_t = \text{VQC6}(o_t * swish(c_t)), \tag{57}$$
$$y_t = \text{NN}(\tilde{y}_t). \tag{58}$$

The models are developed and executed using Python 3.8 in the Google Colab environment and we utilized PennyLane [**6**] as the quantum software base, employing the `default.qubit` backend for quantum computations. In the QLSTM component, we incorporated a 4-variational layer, 4-qubits model, within a single hidden layer framework. Additionally, a dense layer maps the hidden state output to the desired output size for both models. The RMSprop optimizer was used for training both the classical LSTM and the QLSTM models.

5.3 Experiment 1: Radioactive Decay

In the first experiment, we consider simple radioactive decay, which is a process that can be described by a differential equation, capturing the rate at which a quantity of radioactive material decreases over time. The rate of change of the number of radioactive atoms, N, can be modeled using the following ODE:

$$\frac{dN}{dt} = -\lambda_1 N - \lambda_2 N^2, \tag{59}$$

where λ_1 and λ_2 are decay constants. The term $-\lambda_1 N$ represents a first-order decay process, where the rate of decay is directly proportional to the number of atoms present. The additional term $-\lambda_2 N^2$ accounts for a second-order decay process, which might occur in certain complex decay scenarios where the decay rate depends on the square of the number of atoms. This equation illustrates how the decay process can be influenced by multiple mechanisms, making it a valuable model for understanding the dynamics of radioactive substances in various contexts.

We consider the following parameters for our ODE system: λ_1 is set to 0.1, representing the rate of the first process, while λ_2 is set to 50.0, representing the rate of the second process. For simplicity, $N_{\max}$ is assumed to be 1.0.

The eigenvalues of the system can be derived from the Jacobian matrix J. For simplicity, we assume the eigenvalues are given by:

$$\lambda_{\max} = -\lambda_1 - 2\lambda_2 N_{\max}, \tag{60}$$

$$\lambda_{\min} = -\lambda_1. \tag{61}$$

Here, $\lambda_{\max}$ is the most negative eigenvalue, occurring when $N = N_{\max}$, and $\lambda_{\min}$ is the least

negative eigenvalue, occurring when $N = 0$.

5.3.1 Stiffness Ratio Calculation

With $\lambda_1 = 0.1$, $\lambda_2 = 50.0$, and $N_{\max} = 1.0$, the calculated eigenvalues are:

$$\lambda_{\max} = -100.1, \tag{62}$$

$$\lambda_{\min} = -0.1. \tag{63}$$

The stiffness ratio R is then computed to be:

$$R = \frac{|-100.1|}{|-0.1|} = 1001.0. \tag{64}$$

5.3.2 Results

For the data generating (ODE solving) part, we used the Livermore Solver for Ordinary Differential Equations with Automatic method switching (LSODA) method within the Scipy library [**16**], [**17**], [**18**]. LSODA method is a highly effective numerical solver for handling stiff ODE systems. It dynamically switches between non-stiff (Adams) and stiff (Backward Differentiation Formula, BDF) methods based on the behavior of the solution. This adaptability makes LSODA particularly suitable for problems where the stiffness of the system can change over the integration interval. In stiff regions, LSODA employs implicit methods that are more stable and can take larger time steps without sacrificing accuracy, thereby significantly reducing computational cost.

For the first experiment, we select decay rate α 0.99, epsilon ϵ 1e-8 and learning rate lr 0.01. We consider the ODE over interval (1e-8, 1e1). The other hyperparamter selections for the first numerical experiment are shown in Table 9.

Hyperparameter	**QLSTM**	**LSTM**
Optimizer	RMSprop	RMSprop
Loss Function	MSE	MSE
Backend	default.qubit	-
Number of Qubits	4	-
VQC Layer	4	-
Hidden Layer	1	2
Number of Data Points	250	250
Percentage of Train Set	67%	67%
Percentage of Test Set	33 %	33 %
Epochs	100	100

Table 9: Hyperparameter Configuration for Experiment 1

We plotted the normalized predictive results for both LSTM and QLSTM models at 100 epochs, as shown in Figure 26. In these plots, the dashed line represents the ground truth values, the blue solid line indicates the training predictions and the red solid line shows the testing predictions.

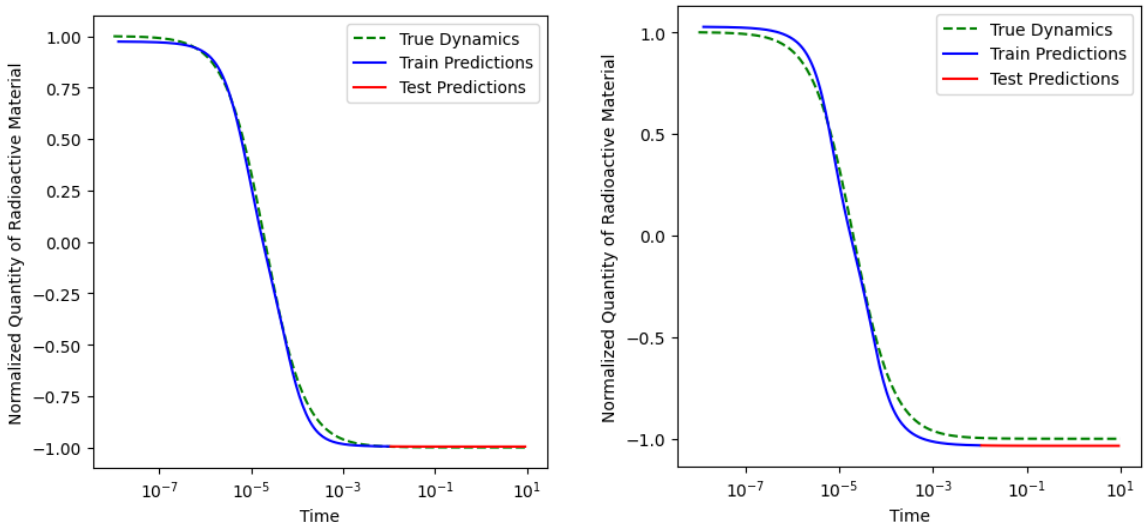


Figure 26: Experiment 1: Predictions QLSTM (left) vs LSTM (right), Train (blue) vs Test (red)

Both the LSTM and QLSTM successfully captured the overall trend. The QLSTM provided more accurate results, especially for the test predictions (red line), as observed by bare eyes.

Additionally, we plotted the losses over 100 epochs for both models in Figure 27, with blue representing the training loss and orange representing the testing loss.

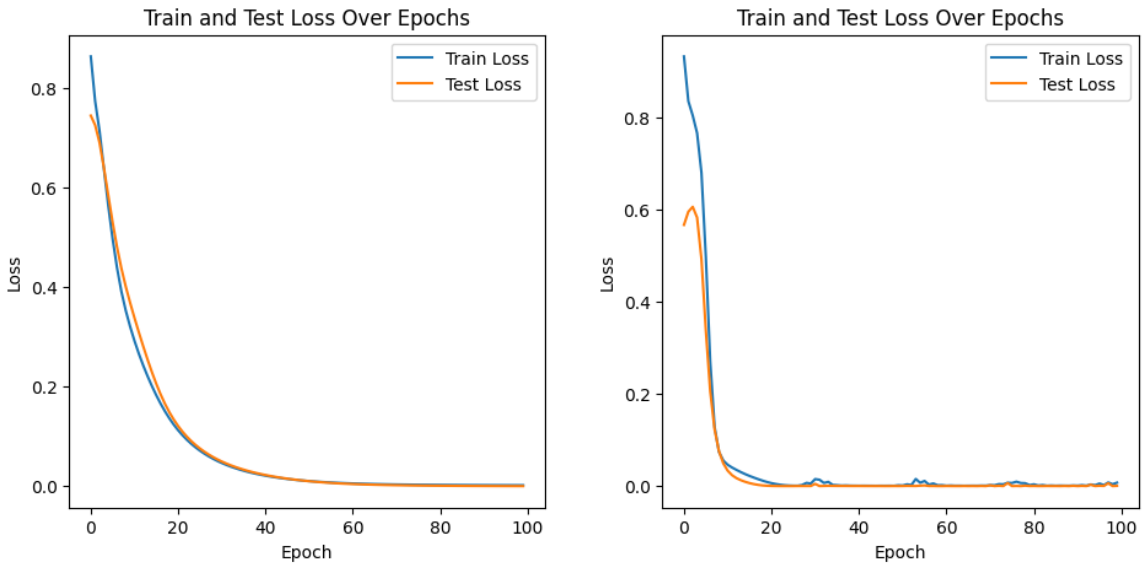


Figure 27: Experiment 1: Loss over 100 epochs QLSTM (left) vs LSTM (right), Train (blue) vs Test (orange)

The QLSTM generated smoother training and testing loss curves over time, while the LSTM exhibited some small spikes. This indicates that the QLSTM model achieves more stable convergence during training, leading to better generalization of the testing data. In this study, mean absolute error (MAE) and root mean squared error (RMSE) are used to evaluate the predictive results.

Metric	**QLSTM**		**LSTM**	
	Train	**Test**	**Train**	**Test**
MAE	10.1528	1.8403	25.7313	11.9552
RMSE	12.0770	1.8637	33.8402	11.9576

Table 10: Comparison of QLSTM and LSTM on Radioactive Decay

The comparison between QLSTM and LSTM on the radioactive decay task, as presented in Table 10, reveals that QLSTM outperforms the LSTM on both the training and testing sets. Specifically, the QLSTM achieves a significant lower Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) on the testing set, with values of 1.84 and 1.86, respectively, compared to the LSTM's 11.96 for both metrics. This underscores the potential advantages of

incorporating quantum elements into traditional neural network architectures for handling complex temporal dynamics. However, we need to test that on more complex problems, which lead to the system of ODEs in the following sections.

5.4 Experiment 2: Two Coupled Damped Harmonic Oscillators

In this section, we discuss the system of two coupled damped harmonic oscillators and compute the stiffness ratio. This system models the dynamics of two coupled pendulums with damping.

The system of differential equations describing the coupled damped oscillators is initially given in the form of second-order differential equations. To facilitate numerical analysis, this second-order system has been reduced into a first-order system by introducing new variables. Specifically, $\omega_1 = \dot{\theta}_1$ and $\omega_2 = \dot{\theta}_2$. This substitution transforms the original second-order equations into a set of first-order differential equations, which are easier to handle computationally.

The original second-order system is:

$$\frac{d^2\theta_1}{dt^2} = -\frac{b_1}{m_1}\frac{d\theta_1}{dt} - \frac{g}{l_1}\sin(\theta_1) + \frac{k_c}{m_1}(\theta_2 - \theta_1), \tag{65}$$

$$\frac{d^2\theta_2}{dt^2} = -\frac{b_2}{m_2}\frac{d\theta_2}{dt} - \frac{g}{l_2}\sin(\theta_2) + \frac{k_c}{m_2}(\theta_1 - \theta_2). \tag{66}$$

By introducing the new variables $\omega_1 = \dot{\theta}_1$ and $\omega_2 = \dot{\theta}_2$, we obtain the following first-order system:

$$\frac{d\theta_1}{dt} = \omega_1, \tag{67}$$

$$\frac{d\omega_1}{dt} = -\frac{b_1}{m_1}\omega_1 - \frac{g}{l_1}\sin(\theta_1) + \frac{k_c}{m_1}(\theta_2 - \theta_1), \tag{68}$$

$$\frac{d\theta_2}{dt} = \omega_2, \tag{69}$$

$$\frac{d\omega_2}{dt} = -\frac{b_2}{m_2}\omega_2 - \frac{g}{l_2}\sin(\theta_2) + \frac{k_c}{m_2}(\theta_1 - \theta_2). \tag{70}$$

By rewriting the system in terms of $\theta_1, \theta_2, \omega_1$, and ω_2, we convert the original second-order differential equations into a system of first-order differential equations. This transformation simplifies the numerical integration process and allows for the application of standard numerical solvers to analyze the system's dynamics and compute the stiffness ratio. The system is defined by the following parameters: The gravitational constant g is set to 9.81 m/s^2. The damping factors for the two oscillators, b_1 and b_2, are set to 200.0 and 0.1, respectively. The lengths of the pendulums, l_1 and l_2, are both set to 1.0 m. The masses of the pendulums, m_1 and m_2, are both set to 1.0 kg. The coupling constant k_c is set to 100.0.

The initial conditions for the system are: $\theta_1(0) = 0.0$, $\theta_2(0) = 0.0$, $\dot{\theta}_1(0) = 3.0$, and $\dot{\theta}_2(0) = 1.0$, where θ_1 and θ_2 are the initial angular displacements, and $\dot{\theta}_1$ and $\dot{\theta}_2$ are the initial angular velocities.

5.4.1 Stiffness Ratio Calculation

With the parameters given previously, the Jacobian matrix J at the equilibrium points $\theta_1 = \theta_2 = 0$ is:

$$J = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{g}{l_1} - \frac{k_c}{m_1} & -\frac{b_1}{m_1} & \frac{k_c}{m_1} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{k_c}{m_2} & 0 & -\frac{g}{l_2} - \frac{k_c}{m_2} & -\frac{b_2}{m_2} \end{bmatrix}.$$

Substituting the given parameters, the Jacobian matrix J becomes:

$$J = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -109.81 & -100.0 & 100.0 & 0 \\ 0 & 0 & 0 & 1 \\ 100.0 & 0 & -109.81 & -0.1 \end{bmatrix}.$$

Solving for the eigenvalues of J, we get:

$$\lambda_1 \approx -199.451, \quad \lambda_2 \approx -0.278 \pm 10.487i, \quad \lambda_3 \approx -0.0938.$$

Considering the real parts of the eigenvalues, the maximum and minimum absolute values

are $|\lambda_{\max}| \approx 199.451$ and $|\lambda_{\min}| \approx 0.0938$, respectively. Thus, the stiffness ratio is:

$$R = \frac{199.451}{0.0938} \approx 2127.256.$$

Given the significant disparity between the damping factors b_1 and b_2, and the coupling constant k_c, we can expect a high stiffness ratio.

5.4.2 Results

For Experiment 2, both models were run for 300 epochs on the interval (1e-4, 1e2). The other hyperparameters remain consistent with those used in Experiment 1. We plotted the predictions for both models in Figure 28 as well as the losses in Figure 29. Again, in the prediction plots, the dashed line represents the ground truth values, the blue solid line indicates the training predictions and the red solid line shows the testing predictions.

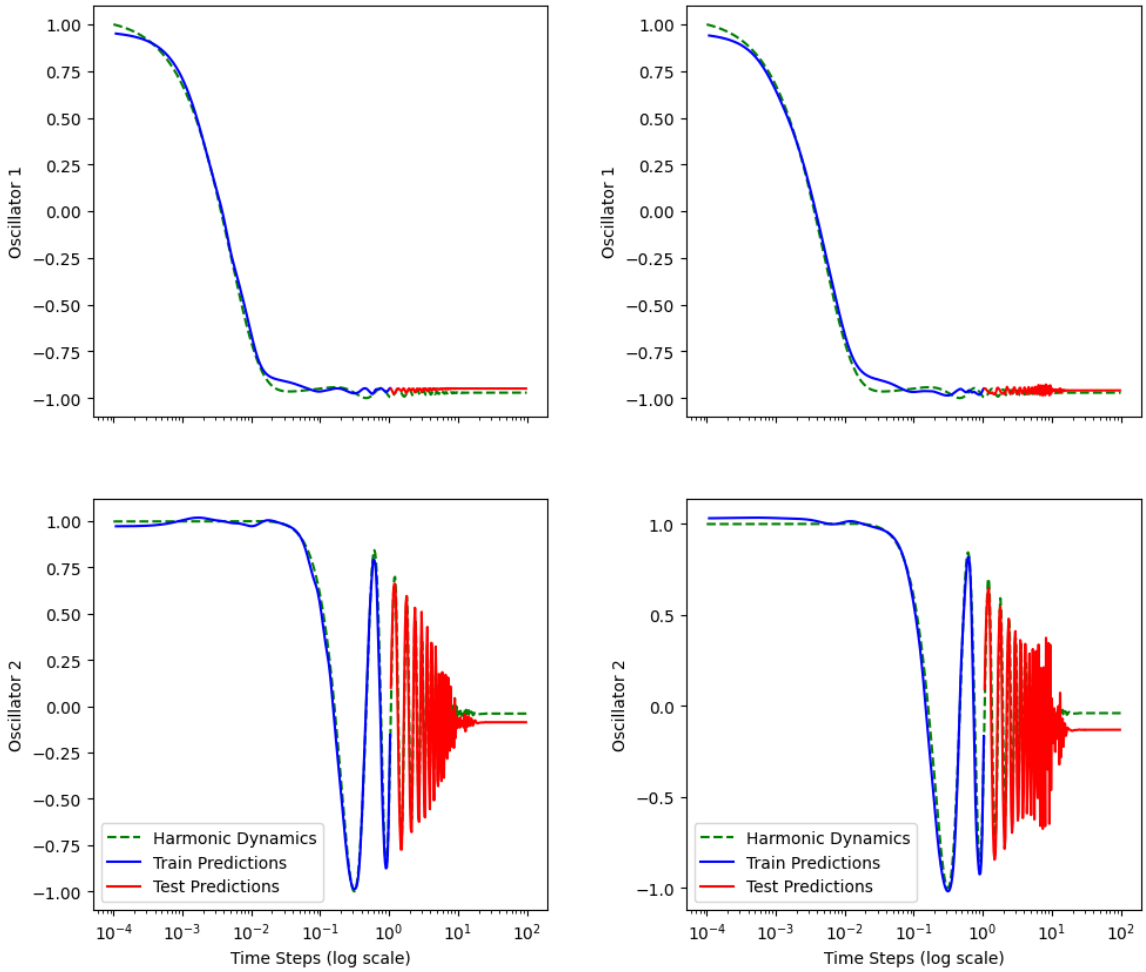


Figure 28: Experiment 2: Predictions QLSTM (left) vs LSTM (right), Top: Oscillator 1, Bottom: Oscillator 2

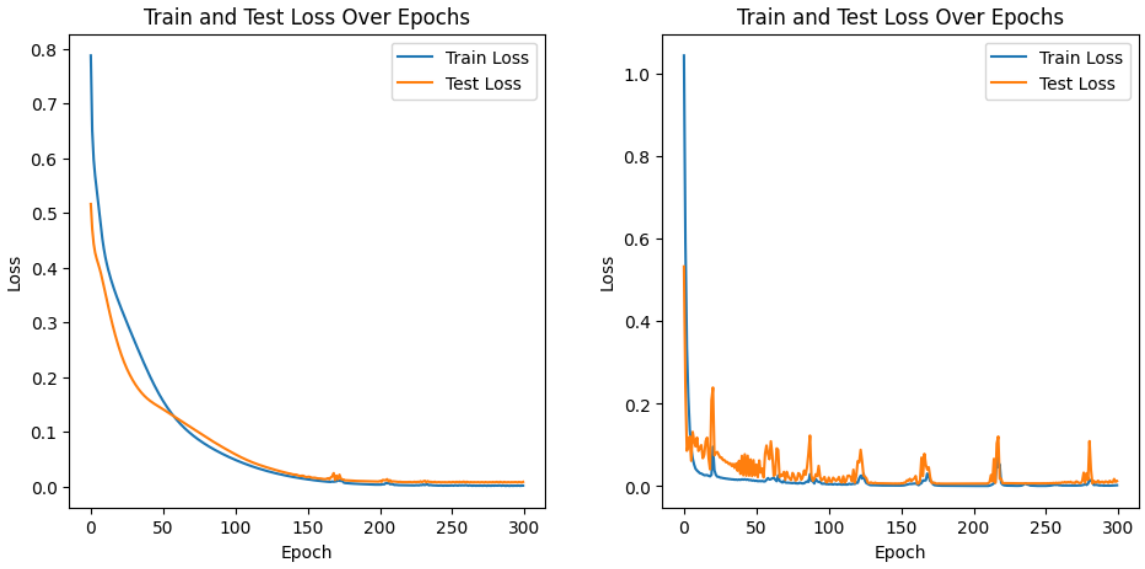


Figure 29: Experiment 2: Loss over 300 epochs QLSTM (left) vs LSTM (right), Train (blue) vs Test (orange)

The similar results are observed from experiment 2, QLSTM provided significantly better results compared to LSTM, especially for the test set (red). Meanwhile, QLSTM gives a stable decreasing loss plot compared to LSTM, significant vibrations and more spikes are visible in the loss plots by LSTM. Notably, there is a small spike between epochs 150 and 200 in the QLSTM plot, which means with the complex increasing, the model was affected in the predicting task.

Similar results were observed in Experiment 2, where the QLSTM model significantly outperformed the LSTM model, particularly on the test set (red). Additionally, the QLSTM model demonstrated a more stable decrease in the loss plot compared to the LSTM model, which exhibited notable vibrations and spikes. Notably, a small spike was observed between epochs 150 and 200 in the QLSTM loss plot, indicating that as the dynamics complexity increases, the prediction can indeed be affected. The error metrics for experiment 2 are summarized in Table 11 and Table 12; we compute them for Oscillator 1 and Oscillator 2, respectively.

Variable	**Train**		**Test**	
	MAE	**RMSE**	**MAE**	**RMSE**
OSC 1	0.0444	0.0530	0.0313	0.0335
OSC 2	0.0218	0.0327	0.1094	0.1500

Table 11: QLSTM Performance on Harmonic Oscillator

Variable	**Train**		**Test**	
	MAE	**RMSE**	**MAE**	**RMSE**
OSC 1	0.0504	0.0591	0.0235	0.0282
OSC 2	0.0344	0.0505	0.1367	0.1775

Table 12: LSTM Performance on Harmonic Oscillator

While LSTM showed slightly better results for the first oscillator in the test phase, with a lower MAE of 0.0235 and RMSE of 0.0282, QLSTM generally provided better overall performance. Specifically, for the second oscillator, QLSTM achieved a significantly lower training MAE of 0.0218 and RMSE of 0.0327, compared to LSTM. In the test phase, QLSTM also outperformed LSTM with a testing MAE of 0.1094 and RMSE of 0.1500 for the second oscillator. This indicates that QLSTM not only fits the training data more effectively but also generalizes reliable results to unseen data, especially for the complex vibrations in Oscillator 2.

We test the models' performance in larger systems with rapid change in the following sections.

5.5 Experiment 3: Robertson System

In this section, we describe the Robertson system, a well-known stiff ODE system introduced by [**19**]. This system models a set of chemical reactions with vastly differing reaction rates.

The Robertson system consists of three coupled nonlinear differential equations given by:

$$\frac{dy_1}{dt} = -0.04y_1 + 10^4 y_2 y_3, \tag{71}$$

$$\frac{dy_2}{dt} = 0.04y_1 - 10^4 y_2 y_3 - 3 \times 10^7 y_2^2, \tag{72}$$

$$\frac{dy_3}{dt} = 3 \times 10^7 y_2^2, \tag{73}$$

where $y_1(t)$, $y_2(t)$, and $y_3(t)$ represent the concentrations of different chemical species over time.

The initial conditions for the system are:

$$y_1(0) = 1, \quad y_2(0) = 0, \quad y_3(0) = 0.$$

5.5.1 Stiffness Ratio Calculation

To compute the stiffness ratio, we analyze the Jacobian matrix J of the system at a specific state. The Jacobian matrix for the Robertson system is given by:

$$J = \begin{bmatrix} -0.04 & 10^4 y_3 & 10^4 y_2 \\ 0.04 & -10^4 y_3 - 6 \times 10^7 y_2 & -10^4 y_2 \\ 0 & 6 \times 10^7 y_2 & 0 \end{bmatrix}.$$

Using the final state of the system after simulation, we compute the Jacobian matrix and its eigenvalues. The eigenvalues of J provide insights into the system's dynamics and stiffness.

Given the final state $[y_1, y_2, y_3]$, we calculate the eigenvalues:

$$\lambda_1, \lambda_2, \lambda_3.$$

The stiffness ratio R is then computed as:

$$R = \frac{\max(|\lambda_1|, |\lambda_2|, |\lambda_3|)}{\min(|\lambda_1|, |\lambda_2|, |\lambda_3|)}.$$

Upon solving the Robertson system using the LSODA method, the final state of the system was determined to be $[y_1, y_2, y_3] = [1.0, 2.0 \times 10^{-7}, 2.5 \times 10^{-6}]$. Substituting these values into the Jacobian matrix, we get:

$$J = \begin{bmatrix} -0.04 & 2.5 \times 10^{-2} & 2.0 \times 10^{-3} \\ 0.04 & -2.5 \times 10^{-2} - 1.2 \times 10^1 & -2.0 \times 10^{-3} \\ 0 & 1.2 \times 10^1 & 0 \end{bmatrix}.$$

Solving for the eigenvalues of this Jacobian matrix, we obtain:

$$\lambda_1 \approx -4.1978 \times 10^3, \quad \lambda_2 \approx 2.4113 \times 10^{-16}, \quad \lambda_3 \approx -8.9238 \times 10^{-3}.$$

The stiffness ratio is then computed as:

$$R = \frac{\max(|\lambda_1|, |\lambda_2|, |\lambda_3|)}{\min(|\lambda_1|, |\lambda_2|, |\lambda_3|)} = \frac{4.1978 \times 10^3}{2.4113 \times 10^{-16}} \approx 1.7409 \times 10^{19}.$$

5.5.2 Results

Firstly, the hyperparameters for the experiment 3 are detailed in Table 13. Both models used a decay rate α of 0.99, an epsilon ϵ of 1e-8, and a learning rate lr of 0.001. The lower learning rate was chosen to prevent overfitting. The Robertson system was considered over the interval (1e-4, 1e4). To improve prediction results, more data points were used with a different train-test split ratio (80-20). Both models were trained for 250 epochs.

Hyperparameter	**QLSTM**	**LSTM**
Optimizer	RMSprop	RMSprop
Loss Function	MSE	MSE
Backend	default.qubit	-
Number of Qubits	4	-
VQC Layer	4	-
Hidden Layer	1	2
Number of Data Points	1000	1000
Percentage of Train Set	80%	80%
Percentage of Test Set	20%	20%
Epochs	250	250

Table 13: Hyperparameter Configuration for Experiment 3

The predictive results are shown in Figure 30, and the loss plots are presented in Figure 31.

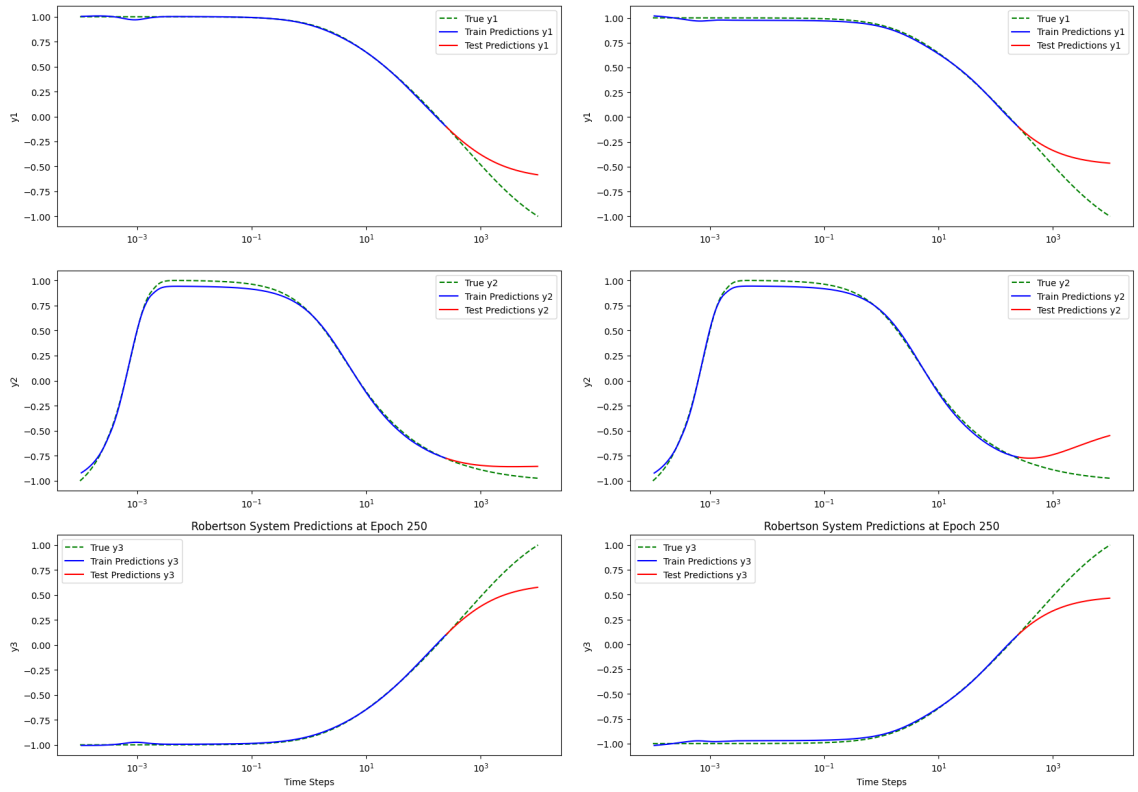


Figure 30: Experiment 3: Predictions QLSTM (left) vs LSTM (right), Train (blue) vs Test (red), Top: y_1, Middle: y_2, Bottom: y_3

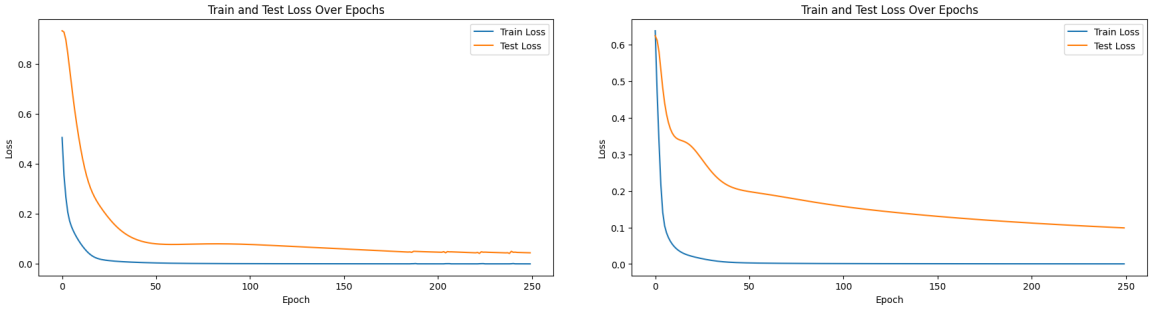


Figure 31: Experiment 3: Loss over 250 epochs QLSTM vs LSTM, Train (blue) vs Test (orange)

Both models demonstrated good fit to the training set and provided comparable predictions. However, QLSTM exhibited slightly better performance for each species, particularly for y_2. With the lower learning rate, both models produced smooth loss curves, but QLSTM showed a more stable decrease and faster convergence.

We also have the error metrics for experiment 3 in Table 14.

Error Metrics	QLSTM		LSTM	
	Train	Test	Train	Test
MAE	0.0060	0.0627	0.0090	0.1041
RMSE	0.0094	0.0832	0.0109	0.1250

Table 14: Robertson Chemical Reaction QLSTM vs LSTM

In this experiment, the QLSTM model outperformed the LSTM model across all metrics. For the training set, the QLSTM achieved a MAE of 0.0060 and a Root Mean Square Error RMSE of 0.0094, compared to the LSTM's MAE of 0.0090 and RMSE of 0.0109. Similarly, for the test set, the QLSTM achieved a MAE of 0.0627 and a RMSE of 0.0832, whereas the LSTM recorded a higher MAE of 0.1041 and RMSE of 0.1250. With the comparable prediction, QLSTM always has the better performance in dynamics learning.

5.6 Experiment 4: Plasma System

For the last section, we apply both models on a large stiff system, the plasma dynamics system, which models the behavior of a plasma with multiple ion species and varying parameters.

The system of differential equations describing the plasma dynamics is:

$$\frac{dv_e}{dt} = -\frac{e}{m_e}E - \nu_e v_e, \tag{74}$$

$$\frac{dv_i1}{dt} = \frac{e}{m_i1}E - \nu_i1 v_i1, \tag{75}$$

$$\frac{dv_i2}{dt} = \frac{e}{m_i2}E - \nu_i2 v_i2, \tag{76}$$

$$\frac{dE}{dt} = -\frac{e}{\epsilon_0}(n_i1 v_i1 + n_i2 v_i2 - n_e v_e), \tag{77}$$

$$\frac{dT_e}{dt} = -\gamma_e T_e, \tag{78}$$

$$\frac{dT_i1}{dt} = -\gamma_i1 T_i1, \tag{79}$$

$$\frac{dT_i2}{dt} = -\gamma_i2 T_i2, \tag{80}$$

where v_e, v_i1, and v_i2 are the velocities of the electron, ion species 1, and ion species 2, respectively. E is the electric field, and T_e, T_i1, and T_i2 are the temperatures of the electron, ion species 1, and ion species 2, respectively. The plasma dynamics system involves the following parameters:

e is the electron charge, set to 1.602×10^{-19} C.

m_e is the electron mass, set to 9.109×10^{-31} kg.

m_i1 is the mass of ion species 1 (hydrogen ion), set to 1.673×10^{-27} kg.

m_i2 is the mass of ion species 2 (helium ion), set to 3.345×10^{-27} kg.

ϵ_0 is the permittivity of free space, set to 8.854×10^{-12} F/m.

ν_e is the electron collision frequency, set to 10^8 s^{-1}.

ν_i1 is the ion species 1 collision frequency, set to 10^5 s^{-1}.

ν_i2 is the ion species 2 collision frequency, set to 5×10^4 s^{-1}.

n_e is the electron density, set to 10^{18} m^{-3}.

n_i1 is the ion species 1 density, set to 10^{18} m^{-3}.

n_i2 is the ion species 2 density, set to 5×10^{17} m^{-3}.

γ_e is the electron thermal relaxation rate, set to 10^6.

γ_i1 is the ion species 1 thermal relaxation rate, set to 10^5.

γ_i2 is the ion species 2 thermal relaxation rate, set to 5×10^4.

The initial conditions for the system are:

$$v_e(0) = 0.1, \quad v_i1(0) = 0.1, \quad v_i2(0) = 0.1, \quad E(0) = 1, \quad T_e(0) = 1, \quad T_i1(0) = 1, \quad T_i2(0) = 1.$$

5.6.1 Stiffness Ratio Calculation

To compute the stiffness ratio, we analyze the Jacobian matrix J of the system. The Jacobian matrix for the plasma dynamics system is given by:

$$J = \begin{bmatrix} -\nu_e & 0 & 0 & -\frac{e}{m_e} & 0 & 0 & 0 \\ 0 & -\nu_{i1} & 0 & \frac{e}{m_{i1}} & 0 & 0 & 0 \\ 0 & 0 & -\nu_{i2} & \frac{e}{m_{i2}} & 0 & 0 & 0 \\ \frac{e}{\epsilon_0} n_e & -\frac{e}{\epsilon_0} n_{i1} & -\frac{e}{\epsilon_0} n_{i2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\gamma_e & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\gamma_{i1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\gamma_{i2} \end{bmatrix}.$$

Given the final state $[v_e, v_{i1}, v_{i2}, E, T_e, T_{i1}, T_{i2}]$, we got the following eigenvalues:

$$\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7.$$

The stiffness ratio R is then computed as:

$$R = \frac{\max(|\lambda_1|, |\lambda_2|, |\lambda_3|, |\lambda_4|, |\lambda_5|, |\lambda_6|, |\lambda_7|)}{\min(|\lambda_1|, |\lambda_2|, |\lambda_3|, |\lambda_4|, |\lambda_5|, |\lambda_6|, |\lambda_7|)}.$$

Upon solving the plasma dynamics system, the final state of the system was determined to be $[v_e, v_{i1}, v_{i2}, E, T_e, T_{i1}, T_{i2}] = [0.1, 0.05, 0.02, 1000, 200, 100, 50]$.

Solving for the eigenvalues of the Jacobian matrix at this final state, we obtain:

$$\begin{aligned} \lambda_1 &\approx -4.9966 \times 10^7 \pm 5.6429 \times 10^{10} i, \\ \lambda_2 &\approx -1.6188 \times 10^5, \\ \lambda_3 &\approx -5.608 \times 10^4, \\ \lambda_4 &\approx -1.0 \times 10^6, \\ \lambda_5 &\approx -1.0 \times 10^5, \\ \lambda_6 &\approx -5.0 \times 10^4. \end{aligned}$$

The stiffness ratio is then computed as:

$$R = \frac{\max\left(|\lambda_1|, |\lambda_2|, |\lambda_3|, |\lambda_4|, |\lambda_5|, |\lambda_6|\right)}{\min\left(|\lambda_1|, |\lambda_2|, |\lambda_3|, |\lambda_4|, |\lambda_5|, |\lambda_6|\right)} = \frac{5.6429 \times 10^{10}}{5.0 \times 10^{4}} \approx 1.1286 \times 10^{6}.$$

In this experiment, we used RADAU [**20**] as the ODEs solver, which is an implicit Runge-Kutta approach. It achieves high accuracy and stability, particularly through A-stability and L-stability, which are important to very stiff problems. RADAU's specialized stability properties make it particularly effective for consistently stiff systems, providing computational acceleration and enhanced performance in our study.

5.6.2 Results

When we are applying the models to the Plasma system, we notice it is very easy to overfitting even with a small enough learning rate (in this case 0.001), so we introduced a Dropout Layer [**21**] into the both models. A dropout layer is a regular technique used to prevent overfitting during the training process. It dropout works by randomly "dropping out" a fraction of neurons during each training iteration. This means that in each forward pass, certain neurons are ignored, which helps in making the model less sensitive to specific neuron weights and thus, improves generalization.

Mathematically, if $\mathbf{h}$ is the input to the dropout layer, and $\mathbf{r}$ is a random vector of Bernoulli trials (each element is 0 with probability p and 1 with probability $1-p$), the output $\mathbf{h}'$ after applying dropout is given by:

$$\mathbf{h}' = \mathbf{h} \odot \mathbf{r}, \tag{81}$$

where $\odot$ denotes the element-wise multiplication. The dropout technique is particularly effective in reducing the risk of overfitting, leading to better performance on unseen data.

Therefore, we considered the plasma system over the interval (1e-9, 1e-7) and both models equipped with a 20% dropout layer were trained for 200 epochs. Besides that, the other hyperparameters are remain the same as in experiment 3.

Finally, we got the predictions for experiment 4 in Figure 32 and losses in Figure 33.

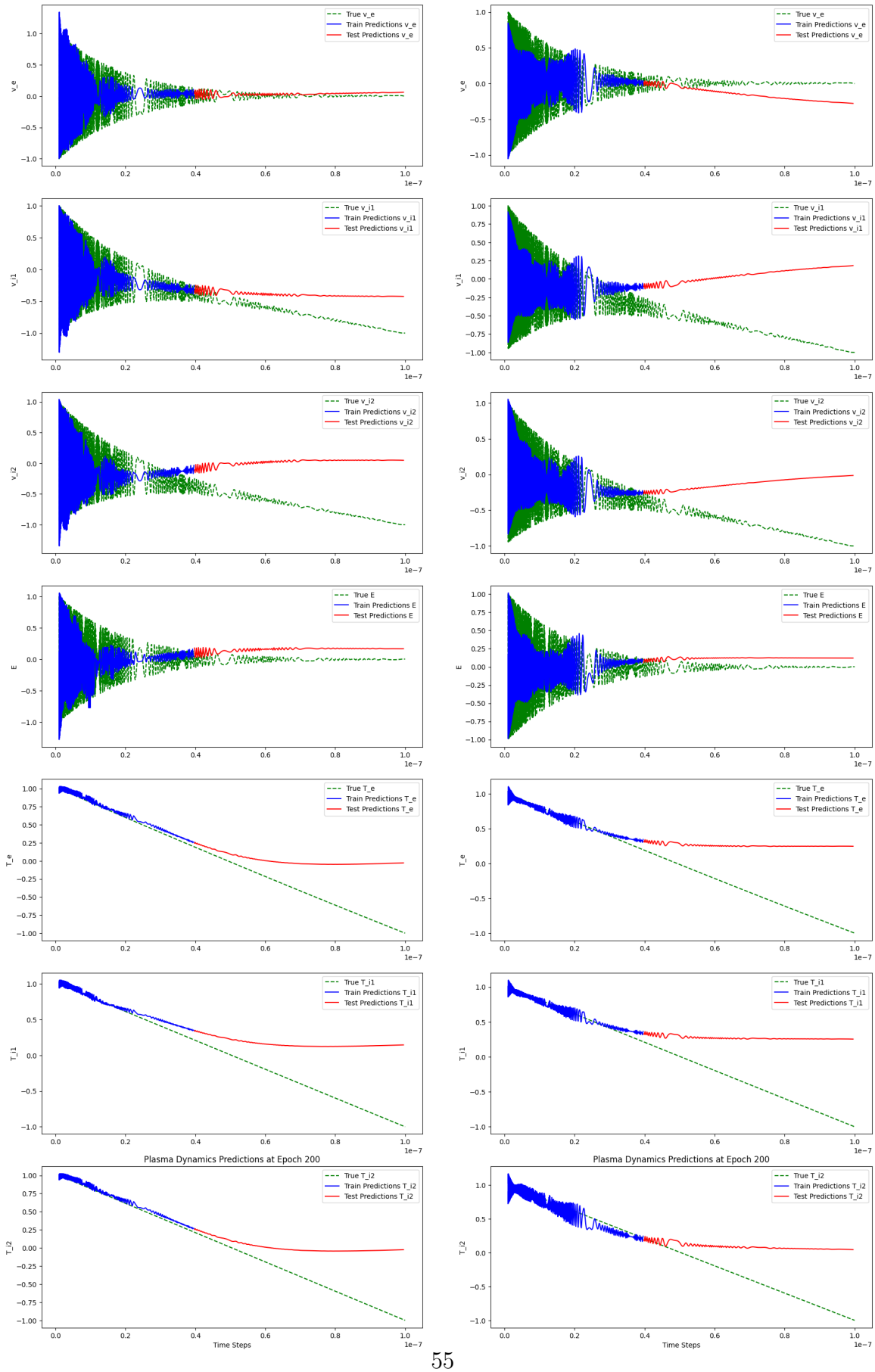


Figure 32: Experiment 4: Predictions QLSTM (left) vs LSTM (right), Variables from top to the bottom: $v_e, v_{i1}, v_{i2}, E, T_e, T_{i1}, T_{i2}$

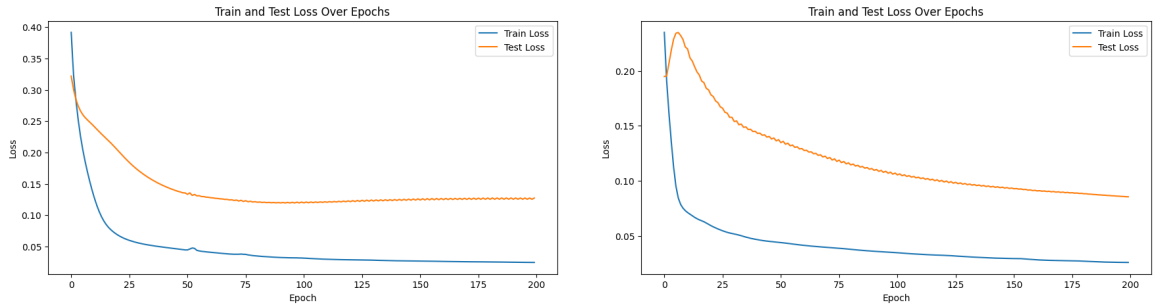


Figure 33: Experiment 4: Loss over 200 epochs QLSTM (left) vs LSTM (right)

Variable	**Train**		**Test**	
Error Metrics	**MAE**	**RMSE**	**MAE**	**RMSE**
v_e	0.5754	0.7002	0.1398	0.1611
v_{i1}	0.5550	0.6795	0.8276	0.9508
v_{i2}	0.5824	0.7198	2.0525	2.1623
E	0.6115	0.7365	0.4832	0.5058
T_e	0.0956	0.1145	1.1007	1.3830
T_{i1}	0.1121	0.1455	1.5118	1.7696
T_{i2}	0.0911	0.1081	1.0628	1.3614

Table 15: QLSTM Performance on Plasma Dynamics

Variable	**Train**		**Test**	
Error Metrics	**MAE**	**RMSE**	**MAE**	**RMSE**
v_e	0.7759	0.9633	0.4629	0.5178
v_{i1}	0.7785	0.9400	2.1164	2.2735
v_{i2}	0.7386	0.9175	1.5655	1.7553
E	0.7989	0.9670	0.3896	0.4142
T_e	0.1230	0.1531	1.8143	2.0767
T_{i1}	0.1216	0.1510	1.8058	2.0738
T_{i2}	0.2329	0.2814	1.2710	1.5858

Table 16: LSTM Performance on Plasma Dynamics

From the above figures, we can see that this system is really hard to predict and QLSTM did give some significant superior results, measuring visually, for instance, v_e (row 1) and v_{i1} (row 2). QSLTM also learnt well in training set, especially for the temperature fitting part

(row 5-7). The QLSTM model demonstrates a smoother and more stable decrease in loss over epochs compared to the LSTM model as well.

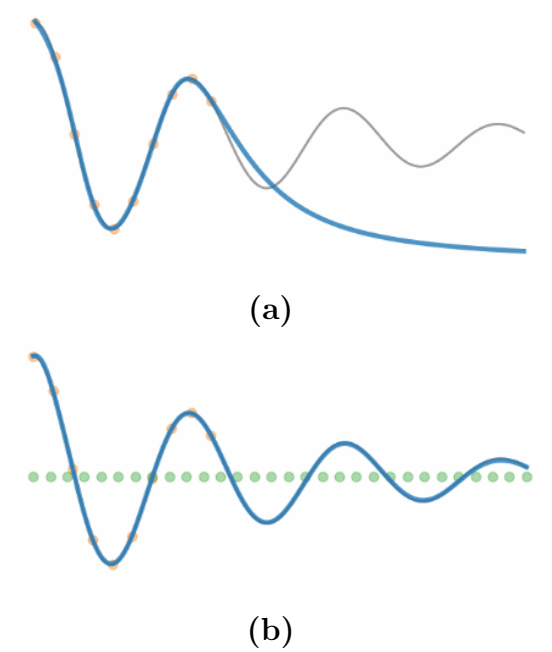
The comparison of QLSTM and LSTM performances on plasma dynamics, as shown in Tables 15 and 16, reveals distinct advantages for the QLSTM model. For the training phase, QLSTM consistently achieves MAE and RMSE across most variables. Specifically, QLSTM attains a notably lower MAE of 0.5754 and RMSE of 0.7002 for v_e compared to LSTM's MAE of 0.7759 and RMSE of 0.9633. Similarly, in the test phase, QLSTM demonstrates superior performance, particularly evident in v_{i1} and v_{i2}, with test MAEs of 0.8276 and 2.0525 respectively, as opposed to LSTM's higher errors. Notably, QLSTM outperforms LSTM significantly in the T_{i2} variable for the test set, achieving a lower MAE of 1.0628 compared to LSTM's 1.2710. These results confirmed the effective fitting in training data for QLSTM as well as offering enhanced predictive accuracy to unseen data in the problem of complex plasma dynamics.

6 Chapter 3: Recent Studies and future Path

In this section we discuss some recent researches and potential steps, including the current studies, some realization and general quantum machine learning.

6.1 QRNNs

In this work for qualifying exam, we explored QRNNs for learning dynamical systems, utilizing a single-layer model. However, to gain more accurate insights, it's essential to conduct a comprehensive comparison with a deeper model featuring more layers and more complex structure. Given that the data is sequential (time series), the model struggled to accurately predict the unseen test dataset for some problem, is due to the simplicity of the MSE loss function employed in this study. To achieve better results, our next step will involve incorporating a Physics-Informed structure [**22**] to enhance the model's ability to learn the dynamics effectively. The following figure is one of my favorite figure to show why do we need Physics-Informed structure, which generated by Ben Moseley [**23**]. In this figure, we can see that the additional physics loss training location in the loss function tries to ensure that the solution learned by the network is consistent with the known physics.



(a)

(b)

Figure 34: Comparison between Neural Network (NN) (a) and Physics-Informed Neural Network (PINN) (b).

Additionally, we aim to incorporate real-life case studies involving partial differential equations (PDEs). These will include models such as the SEIR (Susceptible-Exposed-Infectious-Recovered) epidemiological model, the multi-dimensional Black-Scholes model for American type option pricing which will be boundary free, and some Computational Fluid Dynamics (CFD) models.

6.2 Variational Quantum Algorithm (VQA)

This is the task what we have been working on during the August and I would like to give a comprehensive introduction to this project. We are considering fractional order of PDE in this project, this project is motivated by the study by Leong et al. [**24**].

First, we recall the definition of the Caputo fractional derivative:

$$D_t^\alpha u(t,x) = \frac{1}{\Gamma(n-\alpha)} \int_0^t \frac{\partial^n u(s,x)}{\partial s^n} \frac{ds}{(t-s)^{\alpha-n+1}}, \tag{82}$$

where Γ denotes the Gamma function and $0 < \alpha < 1$ is the fractional order.

To explain it better, we use diffusion equation as an example:

$$D_t^\alpha u(t,x) = \frac{\partial^2 u(t,x)}{\partial x^2}, \tag{83}$$

where D_t^α is the Caputo fractional derivative of order α with respect to time t, and $\frac{\partial^2 u(t,x)}{\partial x^2}$ is the spatial derivative.

To numerically solve the fractional PDE, the first-order finite difference approximation of the Caputo derivative of order $0 < \alpha \leq 1$ is used:

$$D_t^\alpha u(t_k, x_n) \approx g_{\alpha,\tau} \sum_{j=1}^{k} w_j^{(\alpha)} (u_{k-j+1}^n - u_{k-j}^n), \tag{84}$$

where $u_k^n = u(t_k, x_n)$, $g_{\alpha,\tau} = \tau^{-\alpha}/\Gamma(2-\alpha)$, and $w_j^{(\alpha)} = j^{1-\alpha} - (j-1)^{1-\alpha}$.

The discrete sum for the Caputo derivative is rearranged as:

$$D_t^\alpha u(t_k, x_n) \approx g_{\alpha,\tau} \left[u_k^n - w_k u_0^n + \sum_{j=1}^{k-1} (w_{j+1} - w_j) u_{k-j}^n \right], \tag{85}$$

and the spatial derivative is approximated by:

$$\frac{\partial^2 u(t_k, x_n)}{\partial x^2} \approx \frac{u_k^{n+1} - 2u_k^n + u_k^{n-1}}{h^2}. \tag{86}$$

The resulting finite difference scheme for the fractional diffusion equation is iterated in time as:

$$(1 - a\delta) u_k^n = w_k u_0^n - \sum_{j=1}^{k-1} (w_{j+1} - w_j) u_{k-j}^n, \tag{87}$$

where $a = g_{\alpha,\tau}^{-1} h^{-2}$ and $\delta u_n = u_{n+1} - 2u_n + u_{n-1}$.

Since we are approximating the integral, we need all the previous time steps to calculate the next time steps, that introduce the quantum part. In order to solve the discretized PDEs,

the iterative fractional diffusion equation is rewritten using Dirac notation:

$$A|\tilde{u_k}\rangle = |\tilde{f_{k-1}}\rangle, \tag{88}$$

where $|u_k\rangle$ represents the quantum state at time step k, and $|f_{k-1}\rangle$ is a linear combination of previous states. The quantum state at a given time step k, denoted as $|u_k\rangle$, is approximated using an ansatz—a parameterized quantum circuit. The ansatz is constructed from a series of parameterized unitary operations $R(\theta)$ and entangling operations W:

$$|u_k\rangle := \prod_{i=1}^{l} W_{l-i+1} R(\theta_{k,l-i+1})|0\rangle, \tag{89}$$

where θ_k is the set of gate parameters at time step k, l is the number of layers in the ansatz, and $|0\rangle$ is the initial state of the qubits. The ansatz in this case is showing in the following figure.

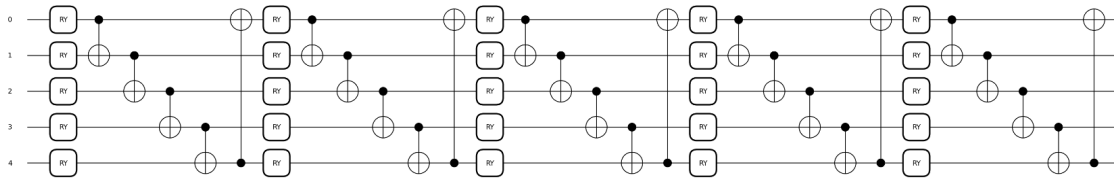


Figure 35: A 5 qubits 5 layers Ansatz

The unnormalized quantum state $|\tilde{u_k}\rangle$ is scaled by a norm factor r_k, leading to the expression:

$$|\tilde{u_k}\rangle := r_k \prod_{i=1}^{l} W_{l-i+1} R(\theta_{k,l-i+1})|0\rangle, \tag{90}$$

where r_k is determined through the optimization process.

and the cost function we used is inspired by the work by Sato et al. [**25**]:

$$C_k(\theta_k) = -\frac{1}{2}\left(\frac{\langle u_k|\tilde{f_{k-1}}\rangle^2}{\langle u_k|A|u_k\rangle}\right). \tag{91}$$

$$|\tilde{f_{k-1}}\rangle = w_k r_0 |u_0\rangle - \sum_{j=1}^{k-1} \Delta w_j r_{k-j} |u_{k-j}\rangle, \tag{92}$$

which is minimized to find the optimal parameters θ_k for the quantum circuit.

I could introduce the more details about the algorithm during the presentation, if someone has interest in it. We aim to extend the problem into higher dimension and more complex problems. Additionally, this project can be run on a real quantum computer, the application of Hamiltonian decomposition on Matrix A make it possible. The most difficult part of achieving this project is the measuring of overlapping part $\langle u_k | \tilde{f_{k-1}} \rangle$ and expectation terms $\langle u_k | A | u_k \rangle$, some algorithms have been proposed to measure these term, such as Hadamard test, SWAP test and so on. However, could these implements work well with PennyLane [**6**]? If not, we are also learning and testing cirq [**26**].

7 Discussion and Conclusion

The results of our experiments indicate a clear advantage in using QRNNs over classical RNNs for modeling dynamical systems, particularly when dealing with stiff ODEs and chaotic systems. The quantum models, specifically QLSTM and QGRU, not only demonstrated superior predictive accuracy but also exhibited faster convergence and more stable training behavior.

In the context of stiff systems, the results from the Robertson chemical reaction system and the plasma dynamics system emphasize the robustness of QRNNs. The QLSTM model, in particular, showed significant improvements in both MAE and RMSE metrics over the LSTM model, indicating its effectiveness in managing the wide range of timescales typical in stiff ODEs.

Therefore, make a quick summary of the current study, QRNNs could give us promising results in advancing the field of dynamical systems modeling. more further research needed.Future work could explore the application of these models to even more complex systems and investigate the scalability of QRNNs as quantum computing technology continues to evolve.

8 References

References

[Hochreiter and Schmidhuber(1997)] Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

[Chung et al.(2014)] Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv* **2014**, *arXiv:1412.3555*.

[Mitarai et al.(2018)] Mitarai, K.; Negoro, M.; Kitagawa, M.; Fujii, K. Quantum circuit learning. *Phys. Rev. A* **2018**, *98*, 032309. https://doi.org/10.1103/PhysRevA.98.032309.

[Chen et al.(2020a)] Chen, S.Y.-C.; Yoo, S.; Fang, Y.-L.L. Quantum Long Short-Term Memory; Computational Science Initiative, Brookhaven National Laboratory: Upton, NY, USA, 2020. https://doi.org/10.2172/1630680.

[Chen et al.(2020b)] Chen, S.Y.-C.; Fry, D.; Deshmukh, A.; Rastunkov, V.; Stefanski, C. Reservoir Computing via Quantum Recurrent Neural Networks. *arXiv* **2020**, *arXiv:2211.02612v1*.

[Bergholm et al.(2022)] Bergholm, V.; Izaac, J.; Schuld, M.; Gogolin, C.; Ahmed, S.; Ajith, V.; Alam, M.S.; Alonso-Linaje, G.; Narayanan, B.A.; Asadi, A.; et al. PennyLane: Automatic differentiation of hybrid quantum-classical computations. *arXiv* **2022**, *arXiv:1811.04968*. https://doi.org/10.48550/arXiv.1811.04968.

[Gajamannage et al.(2023)] Gajamannage, K.; Jayathilake, D.I.; Park, Y.; Bollt, E.M. Recurrent neural networks for dynamical systems: Applications to ordinary differential equations, collective motion, and hydrological modeling. *Chaos* **2023**, *33*, 013109. https://doi.org/10.1063/5.0088748.

[Fu et al.(2019)] Fu, Y.; Saab, S., Jr.; Ray, A.; Hauser, M. A Dynamically Controlled Recurrent Neural Network for Modeling Dynamical Systems. *arXiv* **2019**, *arXiv:1911.00089*.

[Niu et al.(2019)] Niu, M.Y.; Horesh, L.; Chuang, I. Recurrent Neural Networks in the Eye of Differential Equations. *arXiv* **2019**, *arXiv:1904.12933*.

[de la Fraga et al.(2023)] de la Fraga, L.G.; Ovilla-Martínez, B.; Tlelo-Cuautle, E. Echo state network implementation for chaotic time series prediction. *Microprocess. Microsyst.* **2023**, *103*, 104950. https://doi.org/10.1016/j.micpro.2023.104950.

[Kim et al.(2021)] Kim, S.; Ji, W.; Deng, S.; Ma, Y.; Rackauckas, C. Neural Stiff Ordinary Differential Equations. *arXiv* **2021**, *arXiv:2107.10979*.

[Hendrycks and Gimpel(2016)] Hendrycks, D.; Gimpel, K. Gaussian Error Linear Units (GELUs). *arXiv* **2016**, *arXiv:1606.08415*.

[Elfwing et al.(2017)] Elfwing, S.; Uchibe, E.; Doya, K. Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning. *arXiv* **2017**, *arXiv:1702.03118v3*.

[Ramachandran et al.(2017)] Ramachandran, P.; Zoph, B.; Le, Q.V. Searching for Activation Functions. *arXiv* **2017**, *arXiv:1710.05941v2*.

[Szandała(2021)] Szandała, T. Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. *arXiv* **2021**, *arXiv:2109.14545*.

[hindmarsh1983odepack] Hindmarsh, A.C., 1983. ODEPACK, A Systematized Collection of ODE Solvers. Scientific Computing, pp. 55-64.

[petzold1983automatic] Petzold, L., 1983. Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations. SIAM Journal on Scientific and Statistical Computing, 4(1), pp. 136-148.

[virtanen2020scipy] Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, I., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors, 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17, pp. 261-272.

[robertson1966] Robertson, H. H., 1966. The solution of a set of reaction rate equations. Numerical analysis: an introduction. Academic Press. pp. 178–182.

[hairer1991solving] Hairer, E., Nørsett, S.P., and Wanner, G., *Solving Ordinary Differential Equations I: Nonstiff Problems*, Springer-Verlag, 1991.

[srivastava2014dropout] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R., Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929-1958, 2014.

[PINN2019] M. Raissi, P. Perdikaris, G. E. Karniadakis,Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *Journal of Computational Physics*, 2019.

[githubBen] https://github.com/benmoseley/harmonic-oscillator-pinn/blob/main/

[fractionalpde] Fong Yew Leong, Dax Enshan Koh, Jian Feng Kong, Siong Thye Goh, Jun Yong Khoo, Wei-Bin Ewe, Hongying Li, Jayne Thompson, Dario Poletti; Solving fractional differential equations on a quantum computer: A variational approach. AVS Quantum Sci. 1 September 2024; 6 (3): 033802. https://doi.org/10.1116/5.0202971

potentialpde25 ki Sato, Ruho Kondo, Satoshi Koide, Hideki Takamatsu, and Nobuyuki Imoto. Variational quantum algorithm based on the minimum potential energy for solving the Poisson equation. Physical Review A, 104(5):052409, 2021.

[cirq] Cirq Developers. "Cirq". Zenodo, May 29, 2024. https://doi.org/10.5281/zenodo.11398048.