Optimizing Quantum Long Short-Term Memory for Financial Time Series Forecasting

Mohamad Hagog, Alexander Kovacs, Leonard Niessen and Florian Eckstaller
LMU Munich

***Abstract*—The stock market serves as a dynamic arena for financial transactions, wherein investors strategically trade stocks, aiming to capitalize on price fluctuations for profit. Given the computational complexity and cost associated with predicting stock price movements, this paper undertakes a comparative analysis between classical and quantum machine learning methodologies to forecast financial data efficiently. Specifically, we investigate quantum Long-Short-Term-Memory models, optimizing them to discern the impact of various parameters on prediction quality. Our models are trained to forecast stock behavior for both one and ten days ahead. Remarkably, while the quantum approach demonstrates comparable performance to classical methods for one-day predictions, it excels in the ten-day forecast domain. These findings suggest the potential of quantum computing to enhance financial time series forecasting. However, it is imperative to note that the current limitations of quantum hardware remain prominent.**

I. Introduction

Traditionally, forecasting stock market trends has predominantly leaned on fundamental and technical stock analysis until the emergence of computational approaches employing Machine Learning (ML) models gained relevance. Among these models, some leverage artificial neural networks, enabling the model to glean patterns from data and autonomously tackle specific tasks on its own [1]. When training data is not size-constrained, the computational demands for model training can escalate significantly, resulting in even modern classical computers requiring excessive time to complete their tasks. Therefore, Quantum Machine Learning (QML) models appear to be a reasonable strategy to decrease computation time, since they can perform better than regular ML models on certain tasks with equivalent parameters [2].

In the past, promising attempts at financial time series forecasting have proven that there already exist classical solutions with acceptable performance such as ARIMA or recurrent neural networks like Long-Short-Term-Memory (LSTM) models [3]. Some of these models can be adapted to run on quantum hardware [4] and have been compared to a classical counterpart [5]. The common conclusion drawn from these comparisons is that the quantum approach does not consistently yield significantly higher-quality predictions; however, it does require less computational time for training[6].

Effectively training an artificial neural network is crucial for achieving good results. A lot of different parameters play an important role during training such as the input size, the learning rate or the circuit architecture of the model. We discuss the process of optimizing hyper-parameters in section V. Compared to previous studies on financial time series forecasting [7], we use additional data to train our model. Apart from the daily closing price of each stock, volume and other technical stock indicators are included to make the predictions as good as possible.

In this work, we analyze the results from training our quantum LSTM (QLSTM) models with different parameters and also compare the best performing one to a classical LSTM, a non-optimized quantum recurrent neural network (QRNN) and a simple baseline. The objective is to determine which parameter adjustments positively impact predictions and how the forecasting quality trades off with the computational time required for training. In our experiment, as described in Section V, we partition the financial data into training and testing sets before pre-processing it to fit the respective model. The training split is fed into a quantum circuit in order to let the model learn patterns from underlying data by comparing its own predictions with the actual values and then calculates the quality of its predictions with the help of a loss function. Afterwards, the testing split is fed into the same circuit to generate predictions which are then used to evaluate the model itself.

Regarding the evaluation in section VI, we compare each model using evaluation metrics such as the mean squared error (MSE) and accuracy score before discussing our findings in Section VII.

II. Background

A. Stock Predictions

Predicting stock prices based on historical data can be formulated as a time series forecast, consisting of sequential data and predicting the value of the next time step. Some of the many relevant data used for stock price predictions are everyday closing prices, price percentage changes, trading volume, fundamental business data like revenue and earnings or social sentiment. The stock market consists of many complex mechanisms that can be described by chaos theory as a nonlinear deterministic dynamical system [8], [9].

B. Deep Neural Networks

Deep neural networks have the ability to learn and model nonlinear, chaotic systems, therefore having great potential for solving financial prediction problems [10].

C. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a specific type of deep neural networks, designed to handle and model sequential data, by memorizing past inputs.

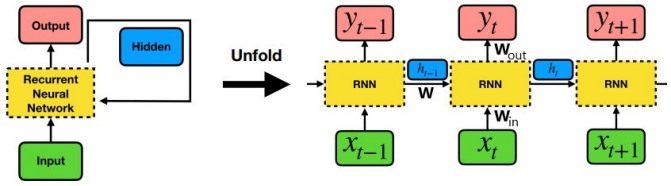


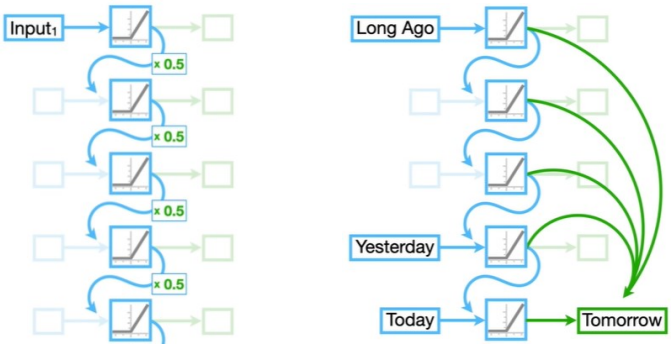
Fig. 1. [4] Each timestep, the next input value x_t is put into the RNN, it returns the outputs y_t and h_t, h_t gets fed as a new input in the next round t+1. The output of a RNN block can be described as

$$h_t = f(x_t \cdot W_{in} + h_{t-1} \cdot W)$$

$$y_t = h_t + W_{out}$$

where f is the activation function, and W_{in}, W_{out}, W are the weights, that need to be optimized.

RNNs possess a multi-layer architecture, where each layer comprises a recurrent block. The key lies in the feedback mechanism of hidden states, where the information from all previous time steps is retained, predictions at any given moment depend on both the current input data and the information about the history of all past elements in the sequence. This architecture allows RNNs to capture and utilize information about the temporal evolution of the data. These attributes make them well-suited for applications involving sequential dependencies, like a time series of stock data, where a sequence of past days is used to predict the next day. However, fundamental RNNs face a major problem: the weights are the same throughout the unfolding, which creates a problem called the vanishing gradient problem.



(a) Original. (b) Extended.

Fig. 2. [11] With each iteration, the past inputs get multiplied with W. Unfolding many times creates small numbers close to 0, causing the gradient to vanish, which makes it hard to optimize the parameters (a). This can be countered by extending the model with a second separate memory component (b).

D. Long Short-Term Memory

This problem can be avoided by upgrading to a LSTM neural network, that can capture long-term dependencies in sequential data. An LSTM cell has 3 inputs, the third being c_{t-1}, adding another memory path to the model. One functions as a short-term memory (h_t), the other as a long-term memory (c_t). This dual-memory structure erases the vanishing gradient problem and enhances numerical stability during training, resulting in more accurate predictions [4].

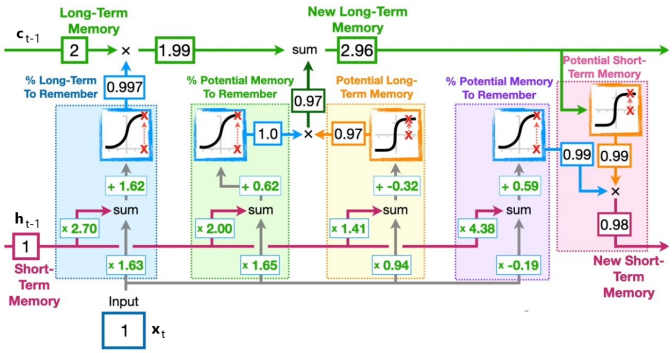


Fig. 3. [11] The blue block is called the forget gate and determines what percentage of the long-term memory remains. The green and yellow blocks form the input gate and decide, what is contributed to the long-term memory. The last two blocks form the output gate and compute a new short-term memory.

E. Quantum Computing

Quantum computing is an an emerging field of computer science that leverages the principles of quantum mechanics to enhance information processing. In contrast to traditional computers that represent data as binary digits or bits, quantum computers utilize quantum bits, or qubits [12]. These qubits are unique in that they can exist in multiple states simultaneously, a phenomenon known as superposition.

This capability of qubits allows quantum computers to process a vast amount of information more efficiently than classical computers. For instance, while a classical computer with n bits can store and process one out of 2^n possible combinations at a time, a quantum computer with the same number of qubits can represent and process all 2^n combinations concurrently [13].

The state of a qubit can be expressed as a linear combination of its base states $|0\rangle$ and $|1\rangle$, represented mathematically as

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

where α and β are complex coefficients that satisfy the equation

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

When a qubit in the state of $\alpha|0\rangle + \beta|1\rangle$ is measured, its superposition collapses into one of its possible states, either $|0\rangle$ or $|1\rangle$, with probabilities determined by the coefficients $|\alpha|^2$ and $|\beta|^2$ respectively [14]. The quantum system transitions from a superposition of states to an actual classical state where the observable's value is precisely known [15].

F. Variational Quantum Circuits

Variational Quantum Circuits (VQCs) build on the principles of quantum circuits, essential in quantum computing. Like

classical circuits, quantum circuits comprise quantum wires and gates. Each quantum wire carries a quantum bit (qubit), analogous to classical bits in classical circuits. Quantum gates perform unitary transformations on these qubits, mirroring the operations of classical gates on classical bits [13].

A VQC, also known as a parametrized quantum circuit, is a unique quantum algorithm that integrates a classical optimization process to encode solutions to specific problems in a quantum state. The process in VQC involves several key steps: initially, classical data is inputted and converted into quantum states via quantum gates. This is followed by the entanglement of qubits using controlled gates and their rotation through parameterized rotation gates. These operations, forming a layer, can be repeated multiple times with varying parameters [16].

After processing, the qubits are measured, and the outcomes are decoded into classical information. The circuit parameters are then refined through classical optimization techniques like gradient descent, aiming to optimize the algorithm's objective function. These adjustments are made iteratively [17].

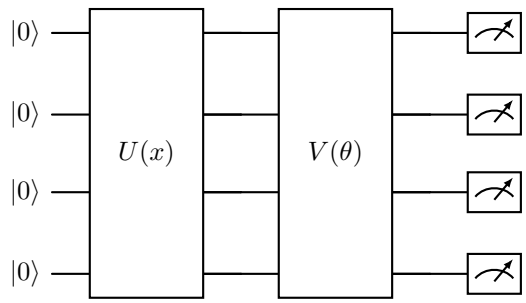


Fig. 4. Variational Quantum Circuit. The general VQC structure contains three main parts: the $U(x)$-block is the state preparation part, the $V(\theta)$-block is the variational part containing trainable parameters θ, and the quantum measurement layer.

A notable feature of VQCs is their resilience to quantum noise [18], making them particularly suitable for Noisy Intermediate-Scale Quantum era computers [19]. Additionally, VQCs have shown promising results in various machine learning applications, including classification [20] and natural language processing tasks [21].

III. Related Work

A considerable amount of research has investigated financial forecasting using computational approaches, particularly focusing on classical machine learning algorithms trained for time series prediction [22]. Notably, RNNs and their variants, such as LSTM or Gated Recurrent Unit, have demonstrated to be especially promising in this context [23] [24] [25].

Although numerous studies have highlighted the exceptional performance of these classical algorithms, this paper shifts its focus towards exploring Quantum Neural Networks (QNNs) as a potential way for surpassing the capabilities of classical neural networks. Quantum machine learning is currently a extensively researched area, with classical models being successfully translated into their quantum counterparts. Notably, the domain of QRNNs has witnessed significant attention, going beyond the mere translation of simple RNNs [4] [26].

In related studies, researchers have addressed challenges akin to financial forecasting. For instance, in the work by Li et al. [27], a novel implementation of the QRNN is applied for sequential learning and assessed in the context of meteorological forecasting, text categorization, and stock price prediction. Comparative analyses against the classical counterpart of the QRNN reveal a substantial improvement in prediction accuracy. Furthermore, when contrasted with an alternative QNN, the performance of the QRNN surpasses that of the QNN, underscoring its enhanced predictive capabilities.

Moreover, existing research has explored solutions to the same problem under consideration in this paper by employing quantum neural networks. For instance, the work by Emmanoulopoulos et al. [6] addresses this challenge by comparing the efficacy of an implemented quantum neural network against that of a classical bidirectional long short-term memory neural network (BiLSTM). The outcomes, consistent with those discussed previously, demonstrate that their parameterized quantum circuit performs comparably to a classical BiLSTM, despite requiring significantly fewer parameters.

Similarly, noteworthy is the methodology presented in this work [5], wherein a Quantum Support Vector Machine is employed for forecasting future stock closing prices, deviating from the conventional use of QNNs. A distinctive feature of this approach is the incorporation of diverse stock price indicators, including Moving Averages, Average True Range, and Aroon, as inputs, offering a more comprehensive understanding of the stock market dynamics beyond relying solely on past stock closing prices.

Moreover, significant strides have been made in the fundamental research aimed at transposing LSTM architectures into the quantum domain, as elucidated by the work of the following paper [4]. In this seminal contribution, the authors explore a novel hybrid quantum-classical LSTM model. After its development, the model undergoes testing on mathematical functions, revealing substantial performance enhancements.

Building upon this foundational research, our work seeks to extend the applicability of such QLSTM models. Drawing inspiration from the aforementioned study, we aim to employ a QLSTM model for the analysis of time series data. By doing so, we intend to ascertain the efficacy of this model in the realm of time series forecasting, thereby contributing to the ongoing exploration of quantum-assisted methodologies in the field. The objective is to harness the demonstrated benefits of the hybrid model in mathematical functions and investigate its potential advantages when applied to the intricate domain of time series analysis.

IV. Methodology

A. Baseline

Establishing a baseline is fundamental for assessing model efficacy in financial time series analysis. The baseline, represented as a horizontal line projecting the last closing price from the training dataset, serves as an essential visionary, positing

that the future stock price will remain unchanged [28]. This approach is instrumental in setting a preliminary benchmark, ensuring that advanced models demonstrate superior predictive capabilities over this simplistic assumption [29].

B. Quantum Recurrent Neural Networks

To gain initial insights into addressing the problem of time series forecasting through Quantum Algorithms, we employ a methodology analogous to classical Machine Learning approaches. This Algorithm is subsequently adapted for its quantum analog. As elucidated in the preceding background section, leveraging recurrence constitutes a prevalent strategy for simultaneously processing multiple temporal steps [27] [26].

Similar to its classical counterpart, the model introduced herein represents a fundamental implementation within the realm of Quantum models. In the context of this paper, its primary purpose is to facilitate a comprehensive exploration of quantum models employing recurrence, while concurrently offering a substantive basis for comparison with our ultimate implementation.

The implemented QRNN utilizes recurrent cells composed of three distinct components: Data Encoding, the chosen Ansatz, and quantum measurement to yield an output.

In the Data Encoding phase, we adopt Angle Encoding as our encoding layer to ensure comparability and mitigate potential drawbacks inherent in this process and moreover to employ a standard and broadly used procedure. Angle Encoding involves representing input data as rotation angles of individual qubits [27].

The primary objective of the Ansatz is to establish entanglement among the qubits within the circuit. To achieve this, we employ an Ansatz that creates entanglement through two-qubit gates, as detailed in [30]. The complete Ansatz utilized in this study is depicted in Figure 5.

Similar to the data encoding strategy, the quantum measurement process aims for comparability with other quantum neural networks. Hence, we opt for a widely acknowledged and standard method of qubit measurement. The Pauli-Z measurement emerges as the most straightforward approach, determining whether an individual qubit is in the state of $|0\rangle$ or $|1\rangle$, as discussed in [27].

The finalized QRNN-Cell is illustrated in Figure 4. As previously elucidated, the last missing step is to stack the cells in a repetitive manner, culminating in the completion of the QRNN model [26].

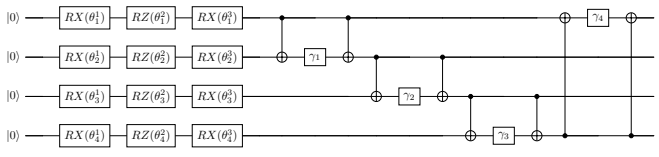


Fig. 5. QRNN-Ansatz.

C. Quantum Long Short-Term Memory

In this study, we propose an extension of classical LSTM networks into the quantum domain. This extension involves replacing the classical neural networks within the LSTM cells with VQCs. These VQCs serve dual purposes: feature extraction and data compression, aiming to leverage the unique computational advantages of quantum mechanisms [4].

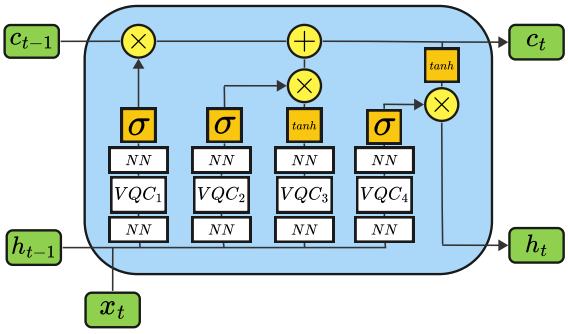


Fig. 6. QLSTM Cell With VQCs and pre- and post-processing NN

The core unit of our proposed QLSTM architecture is the QLSTM cell depicted in 6, which consists of a stack of VQC blocks. Specifically, a QLSTM cell contains four VQCs. The input to these VQCs is a scaled combination of the previous time step's hidden state h_{t-1} and the current input vector x_t. This scaling is performed by a classical linear neural network layer, aligning the inputs to the required four-qubit format, which we call v_t. The output from each VQC is a four-vector set, representing the Pauli Z expectation values for each qubit. These outputs are subsequently scaled using another classical linear neural network layer to achieve the desired output dimension, as depicted in 7. In our case, this dimension is 1, representing a predicted stock price. The measured values are then processed through nonlinear activation functions, specifically sigmoid and tanh.

The mathematical formulation of a QLSTM cell is as follows:

$$y_t = \text{classical NN input layer}(v_t) \tag{3}$$

$$\text{forget}_t = \sigma(\text{classical NN output layer}(VQC_1(y_t))) \tag{4}$$

$$\text{input}_t = \sigma(\text{classical NN output layer}(VQC_2(y_t))) \tag{5}$$

$$\text{update}_t = \tanh(\text{classical NN output layer}(VQC_3(y_t))) \tag{6}$$

$$\text{output}_t = \sigma(\text{classical NN output layer}(VQC_4(y_t))) \tag{7}$$

$$c_t = (f_t \cdot c_{t-1}) + (i_t \cdot g_t) \tag{8}$$

$$h_t = o_t \cdot \tanh(c_t) \tag{9}$$

The operation of the QLSTM cell can be divided into three main blocks:

Forget Block: VQC_1 processes v_t, outputting a vector f_t in the range [0,1] via a sigmoid function. This vector determines

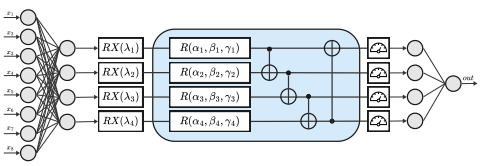


Fig. 7. QLSTM VQC with pre- and post-processing NN

the extent to which elements in the previous cell state c_{t-1} are retained or forgotten by element-wise multiplication.

Input and Update Block: This block decides what new information is added to the cell state. VQC_2 and VQC_3 process v_t, with their outputs determining which values are updated in the cell state. The result from VQC_2, after sigmoid activation, is multiplied element-wise by the new candidate cell state $\tilde{C}_t$, generated by VQC_3 after tanh activation, to update the cell state.

Output Block: After updating the cell state, VQC_4 processes v_t and, following sigmoid activation, determines the relevant values in the cell state c_t for the output. The cell state is then passed through a tanh function and multiplied element-wise by the output from VQC_4 to produce the final output or hidden state h_t.

D. Pseudo Code

Algorithm 1 Training and Testing of the Models

```
Initialize the model with its parameters
Initialize the loss function
Initialize Adam optimizer
Initialize train loader with stock data
Initialize test loader with stock data
for each epoch do
    for each stock do
        Set model to train mode
        for X_Batch, Y_Batch in train loader do
            Get model output from X_Batch
            Calculate loss from the output and Y_Batch
            Use early stopping
        end for
        Calculate average loss
        Apply linear decay learning rate scheduler
    end for
end for
Save model
Set model to evaluation mode
for X_Batch, Y_Batch in test loader do
    Get model output from X_Batch
    Calculate trend accuracy from the output and Y_Batch
end for
Calculate average test loss
```

V. Experimental Setup

A. Data Pre-Processing

For predicting future stock prices, we first fetch actual stock data of 28 stocks from the past two years via an Alpha Vantage API [31]. Four to eight different stock data are picked to predict stock prices. We set a train and test split of 70 percent, meaning the first 70 percent of the past 2 years is used for training our models and on the remaining 30 percent the models get tested. Then data needs to be normalized to fit a neural network. There are several scalers that can be used, we found the MinMaxScaler from scikit-learn [32] to work the best, which scales the data between -1 and 1. After that, data sequences of several days are created, so our models can predict the next day, by utilizing a sequence of a certain amount of past days. To make the training more efficient, the sequences get mixed and bound to batches of size 16, so 16 random sequences of several days are getting fed to the model at once. Further in depth explanation about the data and steps in the pre-processing will be given in the following two sections.

B. Stock Data

Stock price predictions require data that influence price movements in a certain way. The first three kinds of stock data below are daily stock data, that change every day, the rest are data about the company, influencing its value directly and therefore its stock price. These company data are reported every three months, so they only change every three months. The selected stock data (later input features) are as follows:

- **Stock Price**: Historical stock prices can be used to find certain pattern that may repeat themselves in the future.
- **Volume**: Amount of total shares traded in a day. Can be an indicator for bigger prize changes.
- **Percentage Change**: Relative price change over a day, indicating a trend in the positive or negative direction.
- **Revenue**: Generated income of a company, an indicator for the quality and popularity of the products made by a company.
- **EBIT**: Earnings Before Interest and Taxes, measures the success and efficiency of a company to produce at low costs.
- **Cashflow**: Flow of money in a company, expresses liquidity, and is the difference of income and expenditures. It indicates the ability to pay debts or finance new investments. A lower liquidity can increase the risk of insolvency.
- **Assets**: All resources a company owns, including buildings, machinery, equipment, money, shares and more. It contributes to the value of a company.
- **Long Term Debts**: Debts a company has to pay back in more than a year. Higher long term debts come at the expense of a company's profits, due to higher interests and can increase the risk of insolvency.

C. Hyperparameter Selection

Hyperparameter selection is crucial for optimizing learning efficiency in QNNs, such as QLSTMs and QRNNs. These externally configured parameters significantly influence model performance, convergence speed, and overall learning effectiveness. Appropriate hyperparameter tuning is key to navigating the complex optimization landscapes of QNNs, facilitating swift convergence to global optima and managing the balance between generalization and fitting, thereby mitigating overfitting and underfitting issues [33].

In our research, we have chosen a set of hyperparameters for the QLSTM and QRNN models, tailored to maximize their performance for our specific tasks. The selected hyperparameters are as follows:

- **Learning Rate (LR)**: a crucial parameter in gradient descent optimization, controls the step size towards the loss function's minimum. To achieve efficient and effective model convergence, the LR is initially set at 0.03 and linearly decreased to 0.001 across epochs using a PyTorch scheduler [34]. This approach ensures a balance between swift initial advancements and meticulous fine-tuning in subsequent phases.
- **Number of Qubits**: We utilize 4 and 8 qubits. This choice reflects a balance between computational tractability and the capability to represent complex features in the data.
- **Number of Variational Layers**: Set to 2 and 3, these layers form the core of the variational approach in our quantum circuits, allowing for a rich space of representable functions.
- **Rotations in VQC for QLSTM**: We use the BasicEntanglerLayers and StronglyEntanglingLayers from PennyLane [35], which provide an effective entanglement strategy for our quantum gates.
- **Rotations in VQC for QRNN**: We adopt the rotation sequence RX, RZ, RX, enabling a robust framework for feature extraction and transformation.
- **Optimizer**: The Adam optimizer [36] is chosen for its adaptive learning rate capabilities, enhancing the model's ability to converge.
- **Number of Epochs**: Set to 50, this number determines the maximum iterations over the entire dataset for training, ensuring adequate exposure to the data for learning. To optimize the number of epochs and to prevent overfitting, we use early cutting if the model does not learn anymore, more precisely if the loss does not decrease anymore. That means, if the loss decreases less than 0.0005 five times, the training terminates.
- **Lookback/Sequence Length**: Values of 5, 10 and 20 are selected, defining the number of previous time steps to consider for predicting the future value, crucial for capturing temporal dependencies in time series data.
- **Batch Size**: Set to 16, this parameter determines the batch size for training, influencing the gradient estimation and update steps in each epoch.
- **Scaler**: The MinMax scaler [32] is employed for feature scaling, normalizing the input data to aid in faster and more stable convergence of the model. The formula for transformation is given by the scikit-learn [32] documentation:

$$X_{std} = (X - X.min)/(X.max - X.min)$$

$$X_{scaled} = X_{std} * (max - min) + min$$

  where X is the actual value, X.min and X.max are the minimum and maximum of the feature input, min and max equal the scaling range.

The convergence, generalization, and overall performance of QLSTM and QRNN models are significantly influenced by these hyperparameters, making their selection a vital aspect of our research methodology.

D. Metrics

In this section, we describe the evaluation metrics used to assess the performance of our models in forecasting stock market trends. We employ commonly used metrics such as mean squared error (MSE) and trend accuracy score to quantify the predictive accuracy and performance of each model.

1) MSE: is a widely used metric to measure the average squared differences between the predicted values and the actual values. It is calculated as the average of the squared differences between the predicted values ($\hat{y}_i$) and the actual values (y_i) for all data points (n) in the dataset:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

where $\hat{y}_i$ represents the predicted value and y_i represents the actual value for the i^{th} data point.

A lower MSE indicates better predictive performance, as it signifies that the model's predictions are closer to the actual values.

2) Trend Accuracy Score: is a metric used to evaluate the correctness of the predicted trends in stock price movements. In this context, we define a correct trend prediction as one where the model correctly predicts whether the stock price will rise or fall.

To calculate the accuracy score, we first determine the predicted trend for each data point by comparing the predicted value to the previous actual value. If the predicted value is higher than the previous actual value and the stock price actually rises, or if the predicted value is lower and the stock price actually falls, then the prediction is characterized as correct.

The accuracy score is then calculated as the ratio of the number of correct predictions to the total number of predictions made by the model.

This metric provides insight into the model's ability to capture the overall direction of stock price movements.

E. Train Test Split of Stock Data

Dividing datasets into training and testing sets is essential in the data preprocessing phase for ML applications, including in the context of QNNs. This procedure is critical for multiple reasons:

1) **Model Evaluation:** Utilizing a test set allows for an unbiased assessment of the model post-training, facilitating the evaluation of its generalization to unseen data.
2) **Prevention of Overfitting:** Keeping a subset of data separate aids in ensuring the model does not overfit to the training data, where overfitting is characterized by the model learning the noise and fluctuations in the training data to the degree that it impairs performance on new data [37].

For our study, we have adopted the following strategy for the train test split:

- **Split Ratio:** The dataset is divided into 70% for training and 30% for testing. This split ensures that the model has sufficient data to learn from, while also providing a substantial amount of unseen data for testing.
- **Batch Processing:** The data, post-split, is divided into batches of size 16. This batch size is chosen to balance the computational load and the granularity of the gradient update process. During the training phase, these batches are shuffled randomly to improve generalization.

F. Simulators and Hardware

In this work, PennyLane [35], a prominent quantum computing framework, was employed to simulate quantum algorithms. PennyLane's integration capabilities between classical and quantum computations are particularly beneficial for developing and evaluating QRNN models, including QLSTM.

The utilization of PennyLane is instrumental for its provision of a controlled testing environment, facilitating the validation and refinement of our models absent the noise and constraints associated with actual quantum hardware. This feature is invaluable for the expedited prototyping and debugging of quantum algorithms, especially during the preliminary stages of quantum research where access to quantum hardware may be restricted [35].

Subsequent to the simulation and validation stages via PennyLane, the quantum algorithms were executed on genuine quantum hardware through IBM-Q [38]. The deployment on IBM-Q hardware is critical to assess the algorithms' real-world performance, considering factors like quantum noise and hardware imperfections.

The QLSTM architecture proposed in this study incorporates four VQCs, with each VQC corresponding to a distinct job on the IBM Quantum (IBM-Q) platform. Specifically, when the lookback hyperparameter is configured to 10, each VQC must be executed 40 times to facilitate the prediction for the subsequent (11th) day. Consequently, this results in a cumulative total of 160 executions across all VQCs.

For the experimental setup, the QLSTM was trained on the "ibm_brisbane" quantum computer, receiving input tensors comprising four features across 16 batches, thereby constituting a 16×4 tensor structure. Since IBM-Q infrastructure does not inherently support batch processing, a method was devised to mitigate the need to execute the VQCs 640 times (40 executions x 16 batches). This was achieved by parallelizing 16 VQCs within the available 127 qubits, effectively reducing the operational footprint to 64 qubits per VQC execution.

Despite this optimization, the experimental phase encountered significant delays attributed to the extensive queue times on the IBM-Q platform, aggravated by the requirement to run 400 VQCs. This figure is derived from the necessity to process 10 batches over a 10-day lookback period for each of the four VQCs. As of the submission date of this paper, the prolonged queuing times have precluded the acquisition of experimental results.

VI. Evaluation

A. Training Evaluation

To evaluate the training progression of our optimized QLSTM model, we monitor the mean loss across all stocks with respect to the number of epochs. Each epoch represents a complete training cycle on the entire stock dataset, which promotes model generalization and decreases the potential for overfitting to a single stock. This approach ensures that the model's learning is robust and applicable across diverse stock behaviors.

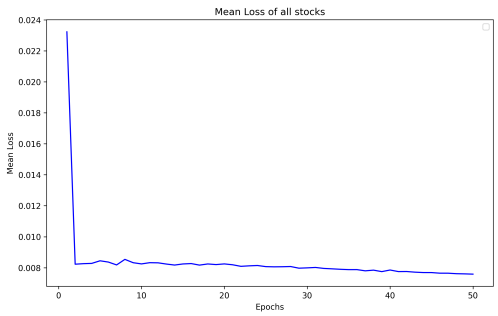


Fig. 8. Mean Loss of all stocks: This plot shows the trend of minimizing the mean loss across all stocks during the training process over 50 epochs. The loss decreases sharply initially and then levels off, indicating the QLSTM model is learning and stabilizing.

B. Models' Parameters Comparison

We have adjusted a few parameters before training our QLSTM model in order to make assumptions on which parameter has what effects on the results. 32 models with slight changes in parameters have been trained and tested throughout the experiment.

Among the various architectures considered, the 4.2 architecture with a lookback of 5 exhibited the highest mean trend accuracy across all seeds and stocks. This version of the QLSTM model was configured with 4 qubits, 3 variational layers, and 8 input features, utilizing strongly entangling layers. The efficacy of our model is illustrated in Figures 10, 11, and 12, where it is benchmarked against alternative architectures

ID	Variational Layer	n_Qubits	n_Variational Layers	Input Size
1.1	BasicEntanglerLayers	4	2	4
1.2	BasicEntanglerLayers	4	3	4
1.3	BasicEntanglerLayers	8	2	4
1.4	BasicEntanglerLayers	8	3	4
2.1	StronglyEntanglingLayers	4	2	4
2.2	StronglyEntanglingLayers	4	3	4
2.3	StronglyEntanglingLayers	8	2	4
2.4	StronglyEntanglingLayers	8	3	4
3.1	BasicEntanglerLayers	4	2	8
3.2	BasicEntanglerLayers	4	3	8
3.3	BasicEntanglerLayers	8	2	8
3.4	BasicEntanglerLayers	8	3	8
4.1	StronglyEntanglingLayers	4	2	8
4.2	StronglyEntanglingLayers	4	3	8
4.3	StronglyEntanglingLayers	8	2	8
4.4	StronglyEntanglingLayers	8	3	8

Tab. I. Variational Layers and their Parameters

utilized in our work. Interestingly, although employing 8 input features might intuitively lead to more accurate predictions, the choice of the lookback parameter demonstrated the opposite effect. Most models performed better with a lookback of 5 compared to 10 for one-day predictions, primarily due to the risk of overfitting associated with higher lookback values during training. A similar trend was observed when comparing the number of qubits, where an increase in qubits often resulted in lower-quality predictions.

Additionally, increasing the number of variational layers generally yielded positive effects on the model's performance. However, the difference between 2 and 3 layers did not significantly stand out in the comparison. The subsequent heatmap illustrates how the best-performing model (4.2, lookback 5) on the right side compares to slightly adjusted architectures. The values depicted in the heatmap represent the percentage of correct trend predictions on a specific stock across all 5 seeds.

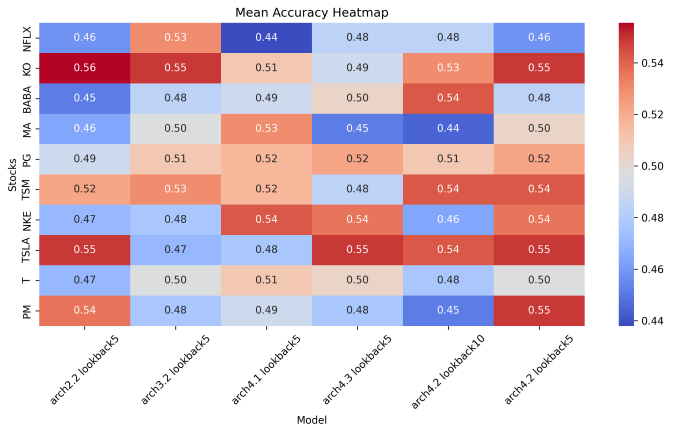


Fig. 9. Heatmap with accuracy for different architectures on different stocks

C. Algorithms' Performance for one-day and ten-day Predictions

We compare our best model (4.2 lookback 5 VI-B) to the QRNN, a classical LSTM and the baseline. The following

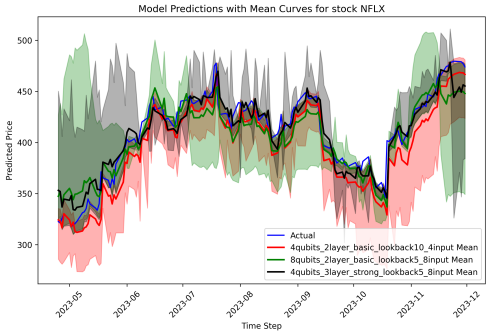


Fig. 10. Netflix Inc.

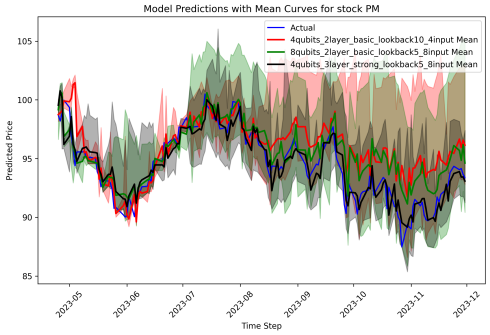


Fig. 11. Philip Morris International Inc.

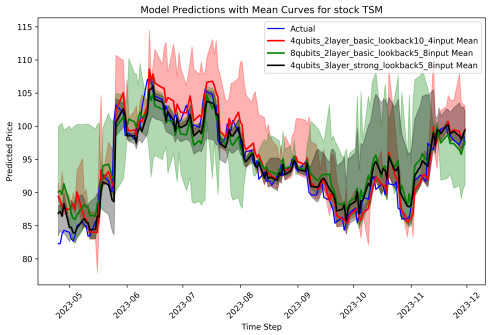


Fig. 12. Taiwan Semiconductor Mfg. Co. Ltd.

plots show the models' predictions of the Philip Morris International Inc. stock as an example 13, 14, 15.

Overall, the QLSTM and classical LSTM networks exhibited comparable performance across all evaluated stocks, achieving a low MSE loss. However, both models achieved an average trend accuracy around 50 percent, indicating that their predictive capability for trend direction does not substantially surpass that of random chance with respect to this metric. Notably, the QLSTM outperformed the LSTM in ten-day forecast horizons, manifesting reduced loss figures. Nevertheless, the LSTM demonstrated a generally similar trend prediction capacity. In contrast, the QRNN and the established baseline model underperformed, incurring higher losses.

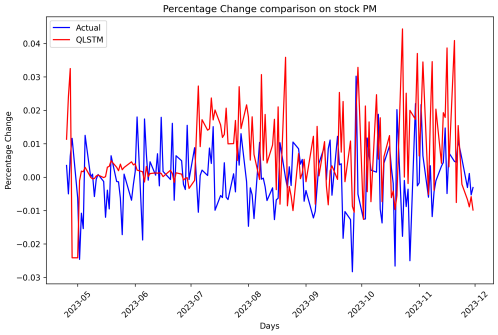


Fig. 13. Percentage Change of Philip Morris Stock: A comparison between the actual and QLSTM-predicted percentage change in Philip Morris stock over time.

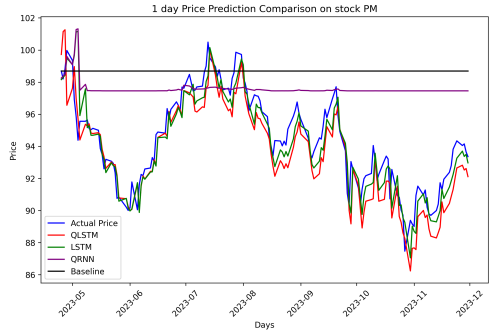


Fig. 14. 1-day prediction performance of all the models

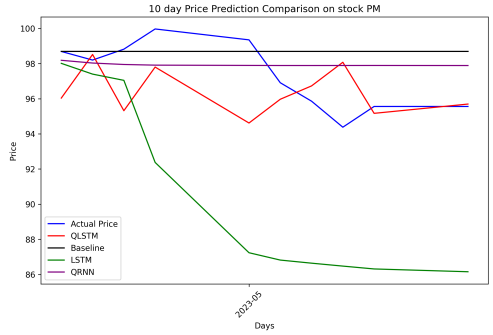


Fig. 15. 10-day future prediction performance of all the models

VII. Conclusion

In this study, our primary focus has been on the implementation and optimization of a QLSTM model. The foundational model is constructed to address the challenges associated with sequential forecasting. We systematically explore various architectures, examining the impact of key parameters, including the number of qubits utilized for input, the number of inputs for predicting future time steps, and variations in the entanglement layer, among other hyper-parameters. Through our investigation, we offer insights into how these choices influence the overall performance and accuracy of the QLSTM model.

A retrospective analysis of our Evaluation section reveals subtle distinctions among the different architectures, with certain modifications demonstrating marginal but discernible benefits to prediction outcomes. Despite the modest nature of these observed changes, they underscore the sensitivity of the model to specific parameter configurations.

This paper serves as a foundational exploration of the optimization of distinct QML models built for the challenges of sequential learning. While we have examined a multitude of hyper-parameters, there remains ample opportunity for further exploration. This research can be viewed as an initial step in the refinement of QML models for efficient time series prediction. Moreover, the methodology presented here can be extended to other models designed to address the problem of time series prediction, opening avenues for continued investigation into additional parameters and their impact on model performance.

Author Contributions

Mohamad contributed to Section II, IV, V, VI. Leonard contributed to Section I, V, VI and abstract. Alex contributed to Section II, V, VI. Florian contributed to Section III, IV VII.

References

[1] Christian Janiesch, Patrick Zschech, and Kai Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, April 2021.

[2] Javier Alcazar, Vicente Leyton-Ortega, and Alejandro Perdomo-Ortiz. Classical versus quantum models in machine learning: Insights from a finance application, 2020.

[3] Shun Liu, Kexin Wu, Chufeng Jiang, Bin Huang, and Danqing Ma. Financial time-series forecasting: Towards synergizing performance and interpretability within a hybrid machine learning approach, 2023.

[4] Samuel Yen-Chi Chen, Shinjae Yoo, and Yao-Lung L. Fang. Quantum long short-term memory, 2020.

[5] Naman S, Gaurang B, Neel S, and Aswath Babu H. The potential of quantum techniques for stock price prediction, 2023.

[6] Dimitrios Emmanoulopoulos and Sofija Dimoska. Quantum machine learning in finance: Time series forecasting, 2022.

[7] Zhengmeng Xu, Yujie Wang, Xiaotong Feng, Yilin Wang, Yanli Li, and Hai Lin. Quantum-enhanced forecasting: Leveraging quantum gramian angular field and cnns for stock return predictions, 2023.

[8] Igor Klioutchnikov, Mariia Sigova, and Nikita Beizerov. Chaos theory in finance. *Procedia Computer Science*, 119:368–375, 2017. 6th International Young Scientist Conference on Computational Science, YSC 2017, 01-03 November 2017, Kotka, Finland.

[9] Ramon Lawrence. Using neural networks to forecast stock market prices. *University of Manitoba*, 333(2006):2013, 1997.

[10] J. B. Heaton, N. G. Polson, and J. H. Witte. Deep learning in finance, 2018.

[11] Josh Starmer. Long short-term memory (lstm), clearly explained. Accessed: 2024-04-02.

[12] N.S. Yanofsky and M.A. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, 2008.

[13] M. Homeister. *Quantum Computing verstehen: Grundlagen – Anwendungen – Perspektiven*. Computational Intelligence. Springer Fachmedien Wiesbaden, 2018.

[14] D. McMahon. *Quantum Computing Explained*. IEEE Press. Wiley, 2007.

[15] M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[16] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. Variational quantum circuits for deep reinforcement learning. *IEEE Access*, 8:141007–141024, 2020.

[17] Xanadu PennyLane Team. Variational circuit - pennylane, 2022.

[18] Samuel Yen-Chi Chen, Chih-Min Huang, Chia-Wei Hsing, Hsi-Sheng Goan, and Ying-Jer Kao. Variational quantum reinforcement learning via evolutionary optimization. *Machine Learning: Science and Technology*, 3(1):015025, feb 2022.

[19] John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, aug 2018.

[20] Samuel Yen-Chi Chen, Chih-Min Huang, Chia-Wei Hsing, and Ying-Jer Kao. An end-to-end trainable hybrid classical-quantum classifier. *Machine Learning: Science and Technology*, 2(4):045021, sep 2021.

[21] Riccardo Di Sipio, Jia-Hong Huang, Samuel Yen-Chi Chen, Stefano Mangini, and Marcel Worring. The dawn of quantum natural language processing, 2021.

[22] Jan G. De Gooijer and Rob J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443–473, 2006. Twenty five years of forecasting.

[23] Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning : A systematic literature review: 2005–2019. *Applied Soft Computing*, 90:106181, 2020.

[24] Alaa Sagheer and Mostafa Kotb. Time series forecasting of petroleum production using deep lstm recurrent networks. *Neurocomputing*, 323:203–213, 2019.

[25] Peter T. Yamak, Li Yujian, and Pius K. Gadosey. A comparison between arima, lstm, and gru for time series forecasting. In *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, ACAI '19, page 49–55, New York, NY, USA, 2020. Association for Computing Machinery.

[26] Johannes Bausch. Recurrent quantum neural networks, 2020.

[27] Yanan Li, Zhimin Wang, Rongbing Han, Shangshang Shi, Jiaxin Li, Ruimin Shang, Haiyong Zheng, Guoqiang Zhong, and Yongjian Gu. Quantum recurrent neural networks for sequential learning, 2023.

[28] Rob J Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2018.

[29] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, 2020.

[30] B. Kraus and J. I. Cirac. Optimal creation of entanglement using a two-qubit gate. *Physical Review A*, 63(6), May 2001.

[31] Alpha vantage api. https://www.alphavantage.co/, 2024. Accessed: 04.02.2024.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[33] Honghe Jin. Hyperparameter importance for machine learning algorithms, 2022.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[35] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M. Sohaib Alam, Guillermo Alonso-Linaje, B. AkashNarayanan, Ali Asadi, Juan Miguel Arrazola, Utkarsh Azad, Sam Banning, Carsten Blank, Thomas R Bromley, Benjamin A. Cordier, Jack Ceroni, Alain Delgado, Olivia Di Matteo, Amintor Dusko, Tanya Garg, Diego Guala, Anthony Hayes, Ryan Hill, Aroosa Ijaz, Theodor Isacsson, David Ittah, Soran Jahangiri, Prateek Jain, Edward Jiang, Ankit Khandelwal, Korbinian Kottmann, Robert A. Lang, Christina Lee, Thomas Loke, Angus Lowe, Keri McKiernan, Johannes Jakob Meyer, J. A. Montañez-Barrera, Romain Moyard, Zeyue Niu, Lee James O'Riordan, Steven Oud, Ashish Panigrahi, Chae-Yeun Park, Daniel Polatajko, Nicolás Quesada, Chase Roberts, Nahum Sá, Isidor Schoch, Borun Shi, Shuli Shu, Sukin Sim, Arshpreet Singh, Ingrid Strandberg, Jay Soni, Antal Száva, Slimane Thabet, Rodrigo A. Vargas-Hernández, Trevor Vincent, Nicola Vitucci, Maurice Weber, David Wierichs, Roeland Wiersema, Moritz Willmann, Vincent Wong, Shaoming Zhang, and Nathan Killoran. Pennylane: Automatic differentiation of hybrid quantum-classical computations, 2022.

[36] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[37] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[38] IBM. Quantum computing. https://research.ibm.com/quantum-computing, 2023. Accessed: 2024-02-04.