Baltic J. Modern Computing, Vol. 14 (2026), No. 1, pp. 174–189
https://doi.org/10.22364/bjmc.2026.14.1.09

Exploring Hybrid Quantum-Classical Methods for Practical Time-Series Forecasting

Maksims DIMITRIJEVS, Mārtiņš KĀLIS, Iļja REPKO

Center for Quantum Computer Science, Faculty of Sciences and Technology, University of Latvia *

maksims.dimitrijevs@lu.lv, martins@kalis.lv, ilja.repko@gmail.com

ORCID 0000-0002-4225-7889, ORCID 0000-0001-9426-9497, ORCID 0009-0003-3154-1803

Abstract. Time-series forecasting is essential for strategic planning and resource allocation, and is important in areas like finance, supply chain management, and energy demand prediction. In such areas, small improvements in forecast quality can lead to significant economic and strategic benefits. It is important to assess the practicality of quantum-enhanced forecasting methods because quantum approaches can complement the strengths of classical models.
In this work, we explore two quantum-based approaches for time-series forecasting. The first approach utilizes a parameterized quantum circuit (PQC) model. The second approach employs variational quantum linear regression (VQLS), formulating the forecasting problem as a system of linear equations solved via quantum optimization. A comparative analysis is conducted to assess their practical applicability and performance.
In our experiments, the PQC with COBYLA optimization achieved the lowest test error rates, outperforming the classical baselines under the reported settings. VQLS formulation faced stability issues on general synthetic datasets.

Keywords: quantum, time-series, PQC, VQLS, forecasting

1 Introduction

Time-series forecasting is a critical tool in fields where accurate predictions of future values based on past observations can drive strategic decision-making and resource optimization (Fisher et al., 2024), (Masini et al., 2020). The fundamental objective in time-series forecasting is to develop a model that effectively captures underlying patterns in historical data and extrapolates them to future points (Fisher et al., 2024). One classical approach to this problem involves framing the prediction as a regression task, where a sliding window of previous time-steps is used to forecast the next value in

* Project conducted in collaboration with Forse AI (Sapiens.BI SIA)

the series (Masini et al., 2020), (Lara-Benítez et al., 2021). This approach naturally leads to a linear system, where past values in the time series are combined through a set of coefficients to produce the forecast. Solving this system by determining the optimal coefficients allows for efficient predictions, and thus, the time-series forecasting task becomes closely aligned with finding solutions to linear equations – a process that is computationally intensive for large datasets but offers promising advantages when adapted to quantum computing techniques (Harrow et al., 2009).

Quantum computing harnesses the principles of quantum mechanics to process information. Unlike classical computers that use bits (0 or 1), quantum computers utilize qubits. Qubits can exist in a superposition, representing both 0 and 1 simultaneously. This fundamental difference allows quantum computers to perform computations in parallel, potentially leading to significant speedups for certain problem classes. However, one has to measure the quantum state to learn any results, and the *interesting* results are only measured with some probability. The goal of quantum algorithms is to increase the probability that the *interesting* results are measured with high enough probability.

While the potential of quantum computing is considerable, current technology faces limitations. Currently, we are in the era of Noisy Intermediate-Scale Quantum (NISQ) devices (Preskill, 2018). These devices are susceptible to noise and have limited qubit counts, hindering large-scale calculations. However, even with these constraints, researchers are exploring algorithms and applications suitable for NISQ devices, such as hybrid quantum-classical approaches.

Algorithms such as the Harrow–Hassidim–Lloyd (HHL) algorithm offer an exponential speed-up over the best known classical algorithms, but they require large and fault-tolerant quantum computers that are not available currently (Harrow et al., 2009). Hybrid quantum-classical algorithms like the variational quantum linear solver (VQLS) (Bravo-Prieto et al., 2023) aim to mitigate the limitations of NISQ devices. These algorithms combine short-depth quantum circuits for cost function evaluation with classical optimization techniques, reducing the reliance on extensive quantum resources.

The performance of both the HHL and VQLS algorithms depends on the condition number of the coefficient matrix of the linear system. There are no guarantees for the coefficient matrices of time-series problems to be well-conditioned. In fact, they are likely to have high condition numbers, leading to weak performance guarantees. However, many models in machine learning do not have strong theoretical performance guarantees, but perform well in practice nonetheless (Zhang et. al., 2021), (Halevy et. al., 2009). In this paper, we set out to test the practical performance of a quantum version of a linear system solver.

Summary of the main contributions of our paper:

- We implemented and evaluated two hybrid quantum-classical approaches for time-series forecasting: a parameterized quantum circuit (PQC) regressor and a VQLS-based regression model.
- We compared quantum models against the classical baselines (linear model, ridge regression, and small neural network) under similar experimental conditions.

- We provided a detailed analysis of model performance and test error for different metrics. This highlights the cases where quantum methods are competitive and where they face limitations.
- We discussed practical considerations such as optimization, reproducibility, and scalability. Therefore, we situate our findings within the broader context of quantum machine learning for time-series forecasting.

The remainder of the paper is organized as follows. Section 2 introduces the methods, including the PQC, the classical regression baselines, and the VQLS formulations. Section 3 presents the experimental setup and evaluation metrics, followed by the results and comparative analysis. Section 4 is the conclusion section that includes a comprehensive summary and a discussion of the findings.

The codes and results of our experiments are publicly available: `https://github.com/infenrio/pqc_and_vqls`.

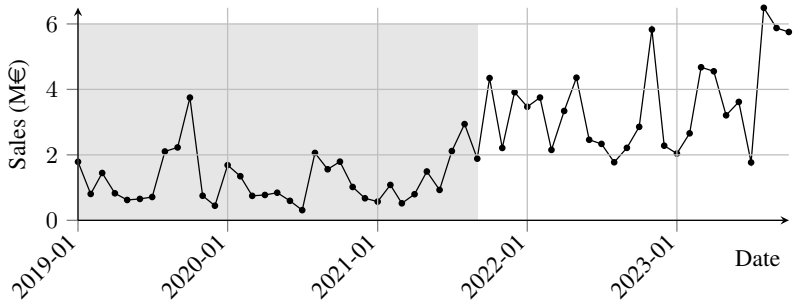
2 Methods

In this section, we introduce the methodological framework of our study. We describe the quantum models (PQC and VQLS) and the classical baselines, along with the data preprocessing steps, problem formulation, and some technical details of the implementation.

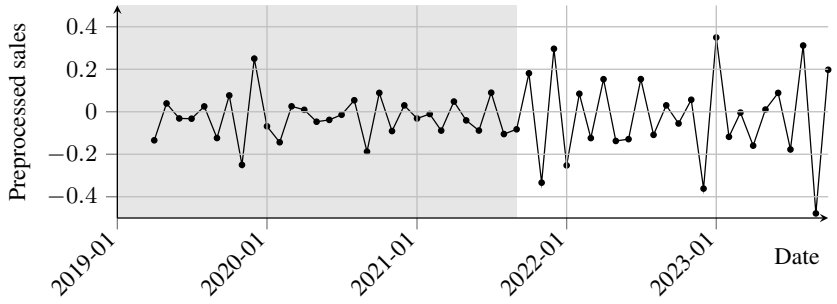
The study evaluated the performance of a PQC model (Benedetti et al., 2019), where two optimization algorithms, L-BFGS-B and COBYLA, were employed to find optimal parameters. The PQC's performance was compared against two classical baselines: a linear regression model and a simple deep learning model. The deep learning model consisted of two hidden layers, each with 12 units and ReLU activation, to mirror the limitations of the currently available quantum devices. The primary evaluation metric used across all models was the mean squared error (MSE).

The dataset used in the experiments was provided by *Forse AI*. It is a synthetic dataset designed to mimic typical sales data patterns analyzed by the company.

Standard data preprocessing steps were applied to ensure the data was suitable for both classical and quantum models. To address non-stationarity often present in time-series sales data, differencing was performed, which enhanced the stationarity of the dataset. This step is critical for improving the performance of models that assume stationarity in their inputs. Furthermore, the data was scaled to fall within a normalized range, a necessary condition to facilitate encoding on a quantum device, which operates effectively only on appropriately scaled data. This preprocessing ensures compatibility and optimal performance across all evaluated models.



(a) Original data. Data in the shaded area is used for training and scaling adjustments. The rest is used for testing.



(b) Preprocessed data after applying differencing (change in sales vs. absolute sales) and scaling the differences to the range [-0.25, +0.25]. Note that test-time data can fall outside of this range.

Figure 1. Comparison of monthly sales (M€) before and after preprocessing from 2019 to 2023.

2.1 Parameterized quantum circuit

The core of the algorithm lies in the PQC, which consists of two primary components:

- *Feature map* encodes the features of each training sample as angles of rotation gates applied to the qubits. In the experiments, 12 previous time-steps are used as features, encompassing information on a full year.
- *Parameterized ansatz* is comprised of entangling gates (CNOT in this case) and parameterized single-qubit gates (R_X and R_Y). The parameters within these gates are what the algorithm adjusts during training to align the predicted values with the actual values from the training set.

The algorithm learns by iteratively updating the parameters of the ansatz to minimize the squared difference between the expected value of an observable (calculated using a Qiskit simulator) and the actual labels for each training sample.

PQC consists of 12 qubits, feature map encodes features through 12 rotations, ansatz has two layers, each consisting of one layer of two-qubit CNOT gates, and two different rotations on each qubit (see Figure 2). Ansatz has a circuit depth of 6 layers of gates and 48 angles that serve as learnable parameters.

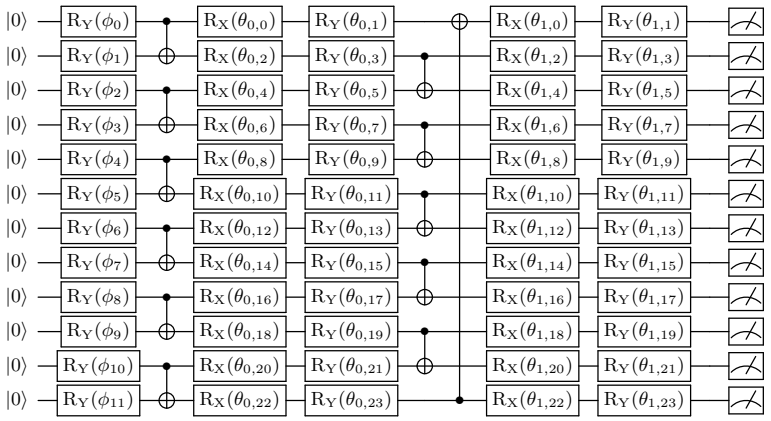


Figure 2. Quantum circuit with parameterized rotation gates. The first layer of gates encodes the data by performing a rotation $\mathrm{R_Y}(\phi_i)$ by the preprocessed data in Figure 1b. ϕ_i are non-trainable. The next two layers entangle the data using two-qubit CNOT operations and learn parameter $\theta_{i,j}$ values that minimize the loss function.

2.2 Classical regression for time-series prediction

To set up a time-series prediction problem using linear regression, suppose we have a time series with n data points, represented as:

$$\{y_t\}_{t=0}^{n-1} = \{y_0, y_1, y_2, \ldots, y_{n-1}\}$$

We want to use a sliding window approach to construct a matrix X and a vector y for supervised learning, where each row in A represents a sequence of past time steps (or "features") used to predict the next time step in y (the "label").

Step 1: Constructing the matrix X and vector y

1. **Choose a window size** m, which determines how many time steps we use to predict the next step.
2. **Form matrix** X: Each row in X is a sequence of m consecutive values from the time series, and there will be $n - m$ rows in total. Formally, $X \in \mathbb{R}^{(n-m)\times m}$ is given by:

$$X = \begin{bmatrix} y_0 & y_1 & \cdots & y_{m-1} \\ y_1 & y_2 & \cdots & y_m \\ y_2 & y_3 & \cdots & y_{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n-m-1} & y_{n-m} & \cdots & y_{n-2} \end{bmatrix}$$

3. **Form vector** y: For each row in X, the corresponding entry in y is the next time step in the sequence. Thus, $y \in \mathbb{R}^{(n-m)}$ is given by:

$$y = \begin{bmatrix} y_m \\ y_{m+1} \\ y_{m+2} \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Step 2: Setting Up the Linear System $Xw = y$ The goal is to find a vector $w \in \mathbb{R}^m$ that represents the coefficients used to linearly combine the values in each row of X to predict the corresponding value in y. This setup leads to the linear system:

$$Xw = y,$$

where:

- X is the matrix of input sequences (past values).
- w is the vector of coefficients (weights) that we want to determine.
- y is the vector of target values (next time steps in the time series).

Each element w_i in the vector w represents the weight or influence of the i-th time step in the input window on the prediction of the next time step. Once w is determined, it can be used to make predictions by applying it to new sliding windows from the time series.

Since X is not guaranteed to be invertible, we can set $A = X^T X$ and $b = X^T y$, and use the normal equation $Aw = b$ to solve for w. Vector w that is a solution to this system will minimize the squared error between Xw and y, providing the best linear combination of previous time steps for predicting the next time step.

2.3 Setting for PQC versus classical regression comparison

We compare the performance of PQC and classical regression. All models as loss function use the MSE.

For PQC, we use COBYLA and L-BFGS-B optimizers, each with the maximum number of iterations set to 1000. Quantum circuits use the default Estimator class object for simulation; therefore, circuits run on a noiseless state-vector simulator.

For initial comparison with linear regression, two basic classical models were used - the simple linear model and the simple neural network. Both imitate two layers with 12 parameters on each layer. The only stopping criteria defined is number of epochs equal to 1000. For initial comparison, we did not set seeds.

For further comparison, to ensure more fair conditions and replicability, we extended our experiments. First, we added one more classical model as an improvement for the simple linear model - ridge regression with L2 regularization. The regularization parameter α was tuned via 5-fold cross-validation over $\alpha \in [10^{-6}, 10^{2}]$. Second, we improved the classical neural network - it is trained with L2 weight regularization ($\lambda = 1 \times 10^{-4}$), dropout (0.2), and early stopping (patience = 20) to prevent overfitting. Third, additionally to MSE, we computed mean absolute error (MAE), symmetric

mean absolute percentage error (sMAPE), and mean absolute scaled error (MASE), as well as the number of iterations for each model during the training, and runtime. We also add confidence intervals (bootstrap) for MSE, MAE, and sMAPE. Lastly, we fixed seeds for our experiments to ensure replicability.

2.4 Variational quantum linear regression

In quantum computing, vectors need to be represented as quantum states, which must be normalized. Thus, we need to:

- Normalize the vector b so that it can be represented as a quantum state $|b\rangle$.
- Reformulate the task as finding a $|w\rangle$ (the quantum state corresponding to the vector w) such that $A|w\rangle$ is proportional to $|b\rangle$. This implies that $A|w\rangle = \lambda|b\rangle$ for some scalar λ.

2.4.1 Construction of matrix M and vector b Any matrix M acting on n qubits can be expressed as a linear combination of matrices M_i with coefficients α_i (Reznik and Oppenheim, 2004) as follows:

$$M = \sum_{i=0}^{4^n-1} \alpha_i M_i,$$

where $\alpha_i = \frac{\mathrm{Tr}(M_i \cdot M)}{2^n}$, and M_i is defined as the tensor product of Pauli matrices, $\mathcal{P}_{ij}$, for each qubit:

$$M_i = \bigotimes_{j=0}^{n-1} \mathcal{P}_{ij}.$$

The Pauli matrices are given by:

$$\sigma_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

The key idea is that M can be represented as a linear combination of the matrices M_i, with the basis M_i being all possible combinations of tensor products of the Pauli matrices. Formally, each $\mathcal{P}_{ij}$ is determined by the j-th digit of i in its quaternary (base-4) representation. Specifically, if i is written in base-4 as $i = c_{n-1}c_{n-2}\cdots c_1 c_0$, where $c_j \in \{0, 1, 2, 3\}$, then $\mathcal{P}_{ij} = \sigma_{c_j}$.

However, in reality, we need at most n^2 non-zero terms of M (Werner, 2001).

For instance, for $i = 6$ and $n = 7$, the base-4 representation of i is 0000012. Therefore:

$$\mathcal{P}_{60} = \mathcal{P}_{61} = \mathcal{P}_{62} = \mathcal{P}_{63} = \mathcal{P}_{64} = \sigma_0, \quad \mathcal{P}_{65} = \sigma_1, \quad \mathcal{P}_{66} = \sigma_2.$$

Vector b is normalized to be represented as a quantum state $|b\rangle$. To obtain $|b\rangle$ from b, each element of b should be divided by the square root of the sum of squares of the elements of b. We initialize the quantum state $|b\rangle$ by using Qiskit initialization functionality, which resembles the transpilation of quantum gates.

2.4.2 Reformulating the Task as Finding a State Vector $|w\rangle$ As discussed in Section 2.2, for input X and y, we aim to solve the equation for the vector w:

$$Aw = b,$$

where $A = X^T X$ and $b = X^T y$. In the quantum setting, the solution for w is typically found in terms of a unit vector:

$$M\,|w\rangle = |b\rangle\,,$$

where $|w\rangle = \frac{w}{\|w\|}$ and $|b\rangle = \frac{b}{\|b\|}$, and M represents the matrix decomposition of A in terms of Pauli operators, i.e., $M = \sum_i \alpha_i M_i$.

The problem is that we cannot directly compute w from the quantum state $|w\rangle$, as the algorithm only provides the unit vector $|w\rangle$, and the norm $\|w\|$ is unknown. Although the vector $|w\rangle$ will be solved by VQLS using the decomposition of A, to find w, we can rescale the answer $|w\rangle$ using the following relation

$$\frac{A\,|w\rangle}{\|A\,|w\rangle\,\|} \approx |b\rangle$$

Multiplying both sides of this equation by $\|b\|$, we get b on the right side:

$$A \cdot \underbrace{\frac{\|b\|}{\|A\,|w\rangle\,\|}\,|w\rangle}_{w} \approx b$$

Thus, the solution for w is:

$$w \approx \frac{\|b\|}{\|A\,|w\rangle\,\|}\,|w\rangle$$

,where $|w\rangle$ is the result obtained from the VQLS and A input matrix.

2.4.3 Parameterized ansatz We did VQLS experiments with 2, 3, and 4 qubits, that represent time series of length n of 4, 9, and 16. For each size of experiment, we used the corresponding ansatz. An ansatz is a PQC, whose parameters can be adjusted to generate a wide range of different quantum transformations and quantum states. An example of an ansatz for 2 qubits is depicted on Figure 3.

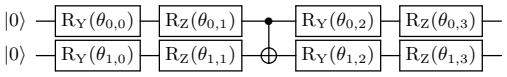


Figure 3. Quantum circuit for parameterized ansatz with 2 qubits ($n = 4$).

2.4.4 Cost Function The goal of the VQLS is to find the optimal parameter vector φ for the ansatz state $|\psi\rangle = M\,|x(\varphi)\rangle$, where $|x(\varphi)\rangle$ is the state after the ansatz and M decomposition of input matrix. We define the cost function similarly to the one in (Bravo-Prieto et al., 2023), but with a slight modification

$$C = \langle\psi|\psi\rangle - \langle\psi|b\rangle\langle b|\psi\rangle$$

Since $|\psi\rangle$ is a unit vector, the first term simplifies to 1, yielding:

$$C = 1 - |\langle b|\psi\rangle|^2$$

This expression can be rewritten as:

$$\langle b|\psi\rangle = \langle b|M\,|x(\varphi)\rangle\rangle = \langle b|\sum_i \alpha_i M_i|x(\varphi)\rangle = \sum_i \alpha_i \langle b|M_i|x(\varphi)\rangle.$$

To evaluate each term in the sum, we use the Hadamard test. This means that the cost function is computed as a sum over all M_i, where for each term, the following procedure is performed: first, the ansatz state $|x(\varphi)\rangle$ is prepared; then, the corresponding M_i transformation is applied. Afterward, the transformation that prepares the unit vector b is applied to the state, and the resulting state is used as input for the Hadamard test as in Figure 4. The Hadamard test allows us to evaluate each term of $\langle b|\psi\rangle$. Finally, the result of the Hadamard test is multiplied by the corresponding α_i, and the final cost function is obtained by summing over all terms.

As an alternative to the Hadamard test, the Swap test can be used. The approach is similar to the Hadamard test, and our experiments with the Swap test have shown the same outcomes as with the Hadamard test, so we omit the details of the Swap test implementation.

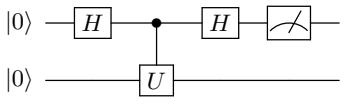


Figure 4. Hadamard Test circuit

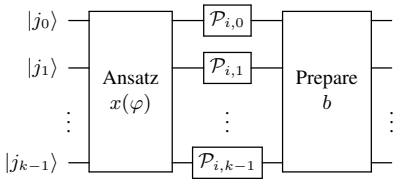


Figure 5. Quantum Circuit U for the Hadamard Test, shown in Figure 4.

The circuit in Figure 5 illustrates the estimation of $\langle b|M_i|x(\varphi)\rangle$ for a matrix M_i, which represents the i-th term in the decomposition of M of Pauli gates $\mathcal{P}_0, \ldots, \mathcal{P}_{k-1}$. Initially, all $k = \log n$ qubits are prepared in the state $|0\rangle$. A parameterized ansatz $x(\varphi)$ is then applied, followed by the application of the M_i decomposition into Pauli gates $\mathcal{P}$. Finally, a transformation is applied to prepare the vector b.

3 Experimental results

In this section, we present the results of our experiments, including training and test performance of quantum and classical models, learning dynamics, and error comparisons.

3.1 First comparison of performance of PQC and classical regression

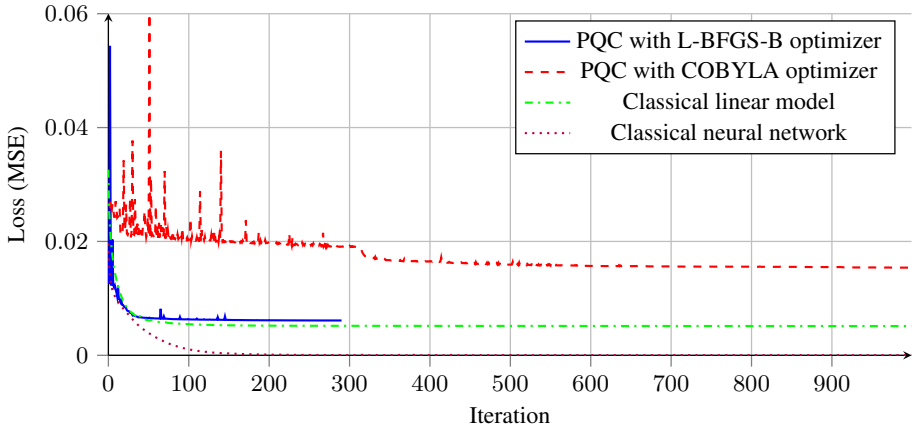


Figure 6. Loss function (MSE) comparison of different optimization methods and models over iterations. PQC with L-BFGS-B optimizer converged after 291 iterations.

Figure 6 demonstrates the learning curve of different models, showing how the training dataset MSE evolved over iterations. The PQC model using the L-BFGS-B optimizer showed the fastest convergence, reaching its minimum error at 291 iterations. In contrast, the COBYLA optimizer required more iterations and converged to a slightly higher error level. Among the classical models, the linear model achieved rapid convergence but displayed a higher final error compared to the quantum model with L-BFGS-B. The neural network showed fast improvement over iterations, leading to overfitting, which is clearly visible in Figure 7. However, the PQC with L-BFGS-B outperformed all the classical models in terms of test set MSE, as summarized in Table 1, indicating a potential advantage in using quantum-enhanced optimizers for this problem.

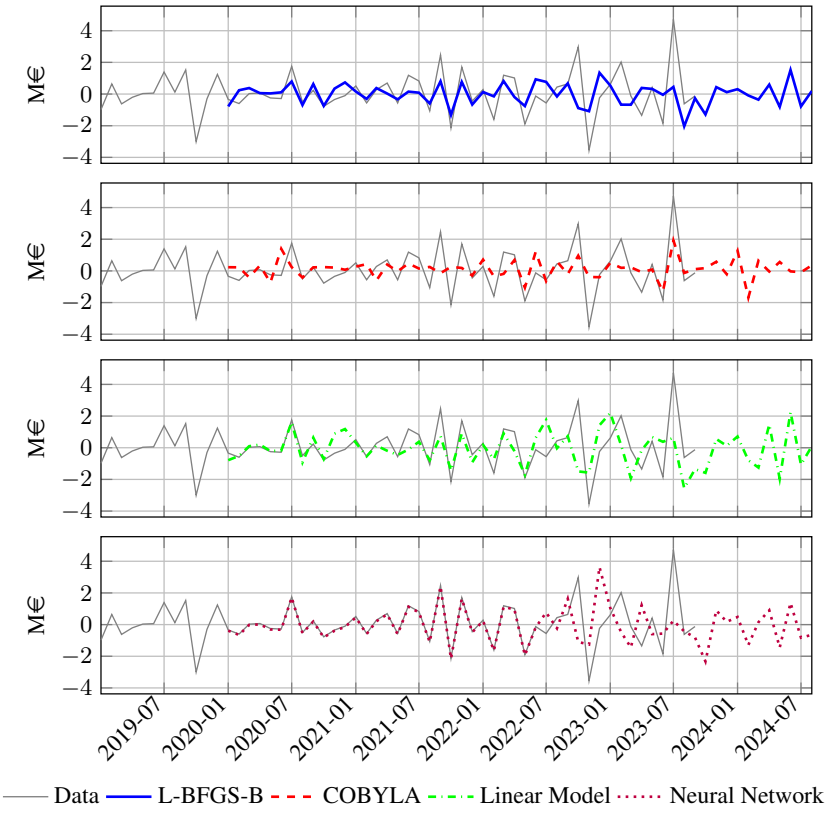


Figure 7. Comparison of different model predictions of the month-on-month sales growth against the actual data.

Table 1. MSE of models on training and test sets

Model	**Training Set MSE**	**Test Set MSE**
PQC with COBYLA	0.01257	0.02106
PQC with L-BFGS-B	0.00612	0.04418
Classical Linear Model	0.00514	0.05177
Classical Neural Network	0.00003	0.05767

The final MSE values on the test set, summarized in Table 1, show that the PQC with COBYLA achieved the best performance on the test set, followed by the PQC with L-BFGS-B. However, both classical models were able to fit the training data better, and with proper regularization would likely achieve improved test MSE. No explicit regularization was applied to any of the models. The neural network, in particular, exhibited signs of overfitting, suggesting that techniques such as early stopping or other forms of regularization could significantly improve its performance.

3.2 Further comparison of performance of PQC and classical regression

In this subsection, we show the results after performing experiments for a fairer and detailed model comparison, described in the last paragraph of Section 2.3. Results in Table 2 and Table 3 were achieved with the random seed set to 25. We repeated our experiments with different fixed seeds (96, 123, 817, 4321, 9999), and the results were similar to the ones that we obtained with the seed set to 25.

Table 2. Evaluation metrics for all models on training and test sets. The lowest Test MSE is shown in bold. MASE is reported once per model.

Model	Dataset	MSE	MAE	sMAPE	MASE	Epochs/Fits	Runtime (s)
PQC (COBYLA)	Test	**0.02065**	0.10191	0.475			
PQC (COBYLA)	Train	0.01276	0.08883	0.533	0.692	1000	807.7
PQC (L-BFGS-B)	Test	0.04363	0.15779	0.636			
PQC (L-BFGS-B)	Train	0.00614	0.06432	0.481	1.072	264	9511.5
Linear Model	Test	0.05111	0.19221	0.676			
Linear Model	Train	0.00516	0.05417	0.449	1.306	1000	4.4
Ridge Regression	Test	0.04211	0.15167	0.651			
Ridge Regression	Train	0.00717	0.06725	0.487	1.031	100	0.2
Neural Network	Test	0.05600	0.19686	0.707			
Neural Network	Train	0.00712	0.06336	0.445	1.338	66	22.4

Table 2 summarizes all evaluation metrics for training and testing datasets. The column *Epochs/Fits* for PQC shows the number of optimizer iterations before the algorithm has stopped, and for classical models, the column shows the number of epochs

before the algorithm has stopped. The *Runtime* column shows the total training time of a given model.

As we can see, PQC with COBYLA still achieved the best performance on the test dataset, among all error measures. We can also see that PQC with L-BFGS-B took the longest time, taking more than 30 seconds per optimization iteration. The performance of PQC with L-BFGS-B and ridge regression is similar across all evaluation metrics.

We can also see in Table 2 that results for neural network have slightly improved, but for some fixed random seeds, neural network showed better results than other classical counterparts, and sometimes even surpasses PQC with L-BFGS-B (e.g., for seed 817). In most cases, higher error rates on the training set resulted in lower error rates on the test set, but not without exceptions (e.g., for seed 4321).

Table 3. Evaluation metrics with 95% confidence intervals for all models on training and test sets.

Model	Dataset	MSE [95% CI]	MAE [95% CI]	sMAPE [95% CI]
PQC (COBYLA)	Test	0.020 (0.006–0.039)	0.101 (0.053–0.157)	0.475 (0.304–0.651)
PQC (COBYLA)	Train	0.013 (0.007–0.020)	0.088 (0.065–0.114)	0.530 (0.414–0.651)
PQC (L-BFGS-B)	Test	0.043 (0.017–0.076)	0.157 (0.092–0.226)	0.637 (0.440–0.833)
PQC (L-BFGS-B)	Train	0.006 (0.004–0.009)	0.064 (0.048–0.080)	0.482 (0.372–0.594)
Linear Model	Test	0.051 (0.025–0.083)	0.191 (0.133–0.252)	0.676 (0.496–0.845)
Linear Model	Train	0.005 (0.003–0.008)	0.054 (0.038–0.071)	0.449 (0.334–0.572)
Ridge Regression	Test	0.041 (0.014–0.074)	0.150 (0.085–0.225)	0.652 (0.454–0.836)
Ridge Regression	Train	0.007 (0.004–0.011)	0.067 (0.049–0.086)	0.486 (0.381–0.601)
Neural Network	Test	0.055 (0.026–0.091)	0.196 (0.133–0.264)	0.707 (0.537–0.850)
Neural Network	Train	0.007 (0.003–0.012)	0.063 (0.044–0.085)	0.445 (0.332–0.563)

Table 3 shows 95% confidence intervals (bootstrap), which are related to the evaluation metrics from Table 2.

Experiments of this subsection were performed on a laptop, equipped with 16 GB of RAM and an Intel Code i5-7300U CPU (no GPU), running Windows 10. We had the following versions of libraries during experiments: 'python_version': '3.8.8', 'qiskit_version': '1.1.1', 'qiskit_machine_learning': 'qiskit_machine_learning 0.7.2'. Results with other seeds are available in our code repository: `https://github.com/infenrio/pqc_and_vqls`. The code also contains seed setting, data preprocessing, optimizer settings, and circuits.

3.3 Performance of VQLS

Testing the VQLS algorithm with the provided matrix decomposition from (Reznik and Oppenheim, 2004) yielded unsatisfactory results. The time-series forecasting predictions were indistinguishable from random values, offering no reasonable forecast. This was also observed when we tested the approach on synthetic data from $Forse\ AI$. One key issue is the extremely high condition number of the corresponding arbitrary matrix.

Transforming a custom matrix with n rows and columns requires $O(n^2)$ Pauli matrices (Werner, 2001).

An alternative solution for addressing this issue in future work could involve applying the Gram-Schmidt orthogonalization procedure. Another potential factor contributing to the poor performance is the COBYLA optimizer, which demands a large number of iterations to converge.

On the other hand, results were satisfactory when we tried time series that strictly grow or strictly decline. In both cases, the output continued the growing/declining trend. This result may have potential uses in trading and investment areas in cases of growing and declining trends. Interestingly, we achieved ideal results when the data represented a geometric progression (both increasing and decreasing), but it is quite an artificial case to be considered useful practically.

Overall, we think that this approach deserves further investigation, as it is based on shallow quantum circuits on a small number of qubits, which is within the capacity of available quantum computers.

4 Conclusion

In this section, we conclude the paper by summarizing the key findings, reflecting on their potential impact, and mentioning limitations and future research directions.

Our findings can be situated within the broader landscape of quantum approaches to time-series forecasting. Several studies have explored shallow PQCs with data reuploading strategies, which enhance expressivity without deep architecture and have shown promising results for sequential learning tasks. Recent work has adapted it to forecasting tasks (e.g., for traffic data) with promising results (Pérez-Salinas et al., 2020). Another promising direction is quantum reservoir computing, where fixed random quantum dynamics are used as a nonlinear feature map. Such models have been shown effective for chaotic time-series, stock market prediction, and system memory tasks (Kutvonen et al., 2020). Alternatively, hybrid architectures such as quantum long short-term memory networks (QLSTM) (Khan et al., 2024), provide a natural way to combine recurrent inductive biases with quantum feature maps. Compared to these approaches, our PQC-based regressor emphasizes shallow, directly trainable circuits optimized for forecasting tasks. PQC as architecture represents a middle ground: more structured than fixed-reservoir QRC, but simpler than full hybrid RNNs.

The considered PQC and VQLS models may scale well. For the PQC model, the window lag is equal to the number of qubits; therefore, if PQCs with shallow depth can show consistent results for a large number of qubits, it can be practical on quantum computers with a high number of qubits and a moderate gate error rate. For the VQLS model, the window lag is proportional to the square of the number of qubits. VQLS can become practical on intermediate-scale quantum computers with low gate error rates.

Our findings demonstrate that hybrid quantum-classical approaches can achieve forecasting accuracy comparable to simple classical models, highlighting their potential role in practical forecasting solutions when quantum hardware improves. Overall, our results suggest that the hybrid quantum-classical approach shows promise, but careful

tuning of hyperparameters, such as the choice of optimizer, plays a crucial role and requires extensive experimentation. Once effective quantum models are identified, they could be integrated into ensembles of models, potentially aiding with regularization and complementing classical models by offering unique strengths and weaknesses, thereby enriching the ensemble's overall performance.

We must acknowledge several limitations of our study. First, we conducted experiments on synthetic data, which simplifies many challenges of real-world forecasting; thus, the generalizability of our claims to diverse application domains remains untested. Second, we evaluated only relatively shallow circuits and small datasets, and the experiments were limited to noiseless simulators. Third, the comparison to classical baselines can be strengthened further, e.g., through systematic hyperparameter tuning and inclusion of more expressive models.

Future work should focus on extending experiments to real-world datasets (including diverse real benchmarks and multiple series), exploring noise-resilient and data reuploading ansätze, incorporating regularization for fairer comparisons, and evaluating scalability with respect to circuit depth and qubit count. An interesting future direction would be to integrate PQC-based forecasters with classical models – this may further clarify their unique contributions to time-series prediction.

Acknowledgements

The authors would like to express their gratitude to *Forse AI* for providing the synthetic dataset used in this study and for their financial support, which made this research possible.

The authors thank anonymous reviewers for their very helpful comments.

References

Benedetti, M. et al. (2019). Parameterized quantum circuits as machine learning models, *Quantum Science and Technology* **4**(4) (2019), p. 043001.

Bravo-Prieto, C. et al. (2023). Variational Quantum Linear Solver, *Quantum* **7** (Nov. 2023), p. 1188. DOI: 10.22331/q-2023-11-22-1188.

Fisher, A. R., Hodgdon, T. S., Lewis, M. G. (2024). Time-Series Forecasting Methods: A Review, ERDC/GRL TR-24-3. DOI: 10.21079/11681/49450.

Halevy, A. et. al. (2009) The Unreasonable Effectiveness of Data *IEEE Intelligent Systems* **24**(2) (2009), p. 8. DOI: 10.1109/MIS.2009.36.

Harrow, A. W., Hassidim, A, Lloyd, S. (2009). Quantum Algorithm for Linear Systems of Equations, *Phys. Rev. Lett.* **103** (15 Oct. 2009), p. 150502. DOI: 10.1103/PhysRevLett.103.150502.

Khan, S. Z. et al. (2024). Quantum long short-term memory (QLSTM) vs. classical LSTM in time series forecasting: a comparative study in solar power forecasting, *Front. Phys.* **12** (Oct. 2024). DOI: 10.3389/fphy.2024.1439180.

Kutvonen, A., Fujii, K., Sagawa, T (2020). Optimizing a quantum reservoir computer for time series prediction, *Sci Rep.* **10**(1) (Sep. 2020), p. 14687. DOI: 10.1038/s41598-020-71673-9.

Lara-Benítez, P., Carranza-García, M., Riquelme, J. C. (2021). An Experimental Review on Deep Learning Architectures for Time Series Forecasting, *International Journal of Neural Systems* **31**(03) (2021). DOI: 10.1142/S0129065721300011.

Masini, R. P., Medeiros, M. C., Mendes, E. F. (2020). Machine Learning Advances for Time Series Forecasting, *Journal of Economic Surveys* **37**(1) (Feb. 2023), pp. 76-111. DOI: 10.1111/joes.12429.

Pérez-Salinas, A., Cervera-Lierta, A., Gil-Fuster, E., Latorre, J. I. (2020). Data re-uploading for a universal quantum classifier, *Quantum* **4** (Feb. 2020), p. 226. DOI: 10.22331/q-2020-02-06-226.

Preskill, J. (2018). Quantum Computing in the NISQ era and beyond, *Quantum* **2** (Aug. 2018), p. 79. DOI: 10.22331/q-2018-08-06-79.

Reznik, B., Oppenheim, J. (2004). A probabilistic and information theoretic interpretation of quantum evolutions, *Quantum Physics* (Aug. 2004). DOI: 10.1103/PhysRevA.70.022312.

Werner, R. F. (2001). All teleportation and dense coding schemes, *Journal of Physics A: Mathematical and General* (2001). DOI: 10.1088/0305-4470/34/35/332.

Zhang, C. et. al. (2021) Understanding deep learning (still) requires rethinking generalization, *Communications of the ACM* **64**(3) (2021), p. 107. DOI: 10.1145/3446776.

Received June 19, 2025 , revised January 18, 2026, accepted February 18, 2026