

POLITECNICO DI TORINO

MASTER's Degree in QUANTUM ENGINEERING



MASTER's Degree Thesis

Time-series prediction using Quantum Reservoir Computing on IQM machines

Supervisors

Prof. Bartolomeo MONTRUCCHIO

Dr. Chiara VERCELLINO

Dr. Giacomo VITALI

Candidate

Marco VENDITTI

JULY 2026

Time-series prediction using Quantum Reservoir Computing on IQM machines

Marco Venditti

Abstract

This work investigates the application of Quantum Machine Learning (QML) techniques - with particular emphasis on the Quantum Reservoir Computing (QRC) paradigm - to time-series forecasting in the financial domain. Quantum reservoirs exploit the high-dimensional nature of Hilbert spaces and the intrinsic dynamics of quantum systems to process temporal data, offering a potentially powerful alternative to classical approaches.

The project involves the design and implementation of QML models based on quantum reservoir architectures, the execution of simulations on different quantum computing backends, and a systematic comparison with classical forecasting methods and benchmarks. Both idealized and realistic scenarios are considered in order to assess the impact of noise and hardware constraints on model performance.

The primary objective is to evaluate the effectiveness of quantum-based methods in capturing complex temporal dependencies and improving predictive accuracy in real-world financial applications. The results provide insights into the advantages and current limitations of quantum reservoir computing, highlighting its potential as a promising tool for next-generation data-driven modeling and forecasting tasks.

ACKNOWLEDGMENTS

*“I owe all my glory to so many races, to every victory,
to the missed opportunities, to the best time in practice,
to every tough track, to my skill...”*

Table of Contents

Acronyms	IX
1 Introduction	1
1.1 Quantum bit	2
1.2 Quantum Machine Learning	5
1.3 Quantum Circuit Ansatz	7
1.3.1 Multiscale Entanglement Renormalization Ansatz (MERA) .	8
1.3.2 Repeated Measurment (RM)	8
1.4 Reservoir Computing	10
1.5 Quantum Reservoir Computing	12
2 Method	15
2.1 Description of the dataset	15
2.2 Classical algorithm	17
2.3 Quantum algorithms	20
2.3.1 QuantumReservoirPy	22
2.3.2 Construction of the MERA and RM ansatz	25
2.3.3 Backends used for simulation	27
2.3.4 Models for fitting data	28
2.3.5 Post-processing methods construction	30
2.4 Metrics for the estimation of errors	33
3 Results	38
3.1 Memory capacity effects in Incremental reservoirs	38
3.2 Performance of Static and Incremental reservoirs	40
3.3 Influence of noise on reservoir performance	41
3.4 Performance on IQM Spark machine	42
3.5 Complete time-series prediction analysis	45
4 Discussion	48
4.1 Classical and Quantum time-series predictions	48
4.2 Performance of MERA and RM ansatz	49
4.3 Behavior and analysis of the multi-step forecasting	51
4.4 Real quantum hardware and simulator performance	51

5 Conclusion and outlook	52
5.1 Overview of the analysis and key findings	52
5.2 Improvements and future works	53
A Appendix A	56
A.1 Machine Learning	56
B Appendix B	58
B.1 Recurrent Neural Networks	58
C Appendix C	60
C.1 NARMA task	60
D Appendix D	62
D.1 Quantum Gates	62
E Appendix E	64
E.1 Quantum Measurement and State Collapse	64
F Appendix F	66
F.1 Correlation and entanglement	66
G Appendix G	68
G.1 Noise, decoherence and computational fidelity	68
H Appendix H	70
H.1 IQM SPARK	70
I Appendix I	71
I.1 Additional considerations on Prophet model	71
J Appendix J	73
J.1 Additional comparisons using different error metrics	73
K Appendix K	77
K.1 Computational tools and development environments	77
L Appendix L	79
L.1 Project repository structure	79
M Appendix M	81
M.1 Project contributions and supporting resources	81
Bibliography	82
Dedications	86

List of Figures

1.1	Classical and Quantum Information Representation.	3
1.2	Illustrative Quantum Circuit with encoding and variational ansatz layers.	6
1.3	Example of a quantum circuit illustrating both the encoding and ansatz stages.	6
1.4	Timeline highlighting major advancements in quantum machine learning. It is possible to divide the history of quantum machine learning into two phases: the first one started during the mid-90s until 2007, and was mainly focused on models' formulation, the second one, in which we currently are, is focused more on implementation [12]. Early works, dealing with quantum neural networks, were biologically inspired [13] [14] [15], with the hope of finding quantum-rooted explanations in the working principles of the brain. In the early 2000s, the question of statistical learning in a quantum framework was discovered [16] [17] [18]. The first monograph on the subject was published in 2014 [19]. In the following years, interest in the topic increased, producing a vast body of literature. In 2020, the first version of Tensorflow Quantum was released by Google [20], as a part of the Tensorflow platform. Furthermore, cloud-based models began to allow for the implementation of QML models on actual quantum computers. . . .	7
1.5	Multiscale Entanglement Renormalization Ansatz scheme [24]. . . .	9
1.6	Repeated Measurement scheme [26].	10
1.7	Example of a graphical diagram of a Reservoir Computing.	11
1.8	Example of a graphical diagram of a Quantum Reservoir Computing.	13
2.1	Graphical diagram of an Iterative Multi-Step Forecasting (Sliding Window Approach).	17
2.2	Prophet previsions.	18
2.3	Prophet trends.	19
2.4	Prophet final time-series prediction.	20
2.5	The five functions of QReservoir.	23
2.6	Static and Incremental three layered circuit structure.	24
2.7	Model performance comparison in term of absolute error for MERA analysis.	31

2.8	Model performance comparison in term of absolute error for RM analysis.	31
2.9	Comparison of different feature sets in terms of absolute error (AE) obtained after training and applying a linear regression model for MERA analysis	32
2.10	Features performance comparison in term of AE for RM analysis. . .	33
2.11	Comparison in term of MAE for PROPHET, RM and MERA cases.	33
2.12	Comparison in term of MAPE for PROPHET, RM and MERA cases.	34
2.13	Comparison in term of MSE for PROPHET, RM and MERA cases.	34
2.14	Comparison in term of RMSE for PROPHET, RM and MERA cases.	35
2.15	Comparison in term of MedAE for PROPHET, RM and MERA cases.	35
2.16	Comparison in term of NMSE for PROPHET, RM and MERA cases.	36
2.17	Dynamic Time Warping (DTW) Analysis for ATM 350002.	37
3.1	Memory-related behavior specifically associated with the Incremental reservoir in terms of absolute error for MERA approach.	39
3.2	Memory-related behavior specifically associated with the Incremental reservoir in terms of absolute error for RM approach.	39
3.3	Overall comparison memory-related behavior for MERA, RM, Prophet approaches in terms of MAE.	40
3.4	Performance comparison between the Static and Incremental reservoir approaches in terms of absolute error for MERA approach.	40
3.5	Performance comparison between the Static and Incremental reservoir in terms of absolute error for RM approach.	41
3.6	Comparison between AerSimulator and IQM_FakeAdonis in terms of absolute error for MERA approach (Static reservoir).	42
3.7	Comparison between AerSimulator and IQM_FakeAdonis in terms of absolute error for RM approach (Incremental reservoir).	42
3.8	Absolute Error comparison between the performance of RM and MERA approaches after simulation on IQM Spark machine (Incremental reservoir).	43
3.9	Absolute Error comparison between the AerSimulator, Adonis and Lagrange (IQM Spark) backends.	44
3.10	Absolute Error comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends; view the Appendix section for others results in term of error comparison.	44
3.11	Prophet final time-series prediction.	45
3.12	MERA_Incremental_AerSimulator final time-series prediction. . . .	46
3.13	MERA_Incremental_Fake_Adonis final time-series prediction. . . .	46
3.14	MERA_IQM_SPARK final time-series prediction.	46
3.15	RM_Incremental_AerSimulator final time-series prediction.	47
3.16	RM_Incremental_Fake_Adonis final time-series prediction.	47
3.17	RM_IQM_SPARK final time-series prediction.	47
D.1	List of single-qubit, two-qubit, and rotational quantum gates.	63

E.1	Quantum Measurement and State Collapse.	65
F.1	Quantum circuit for the preparation of an entangled two-qubit state. The first qubit is initialized in the $ 0\rangle$ state and transformed into a superposition by applying a Hadamard gate, resulting in $\frac{1}{\sqrt{2}}(0\rangle + 1\rangle)$. Subsequently, a controlled-NOT (CNOT) gate is applied, using the first qubit as control and the second as target. This operation entangles the two qubits, producing the Bell state $\frac{1}{\sqrt{2}}(00\rangle + 11\rangle)$, in which the measurement outcomes of the two qubits are perfectly correlated. . .	67
H.1	The topologic map of the device IQM Spark.	70
J.1	Relative error (RE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.	73
J.2	Mean absolute error (MAE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.	74
J.3	Mean absolute percentage error (MAPE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.	74
J.4	Mean squared error (MSE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.	75
J.5	Median absolute error (MdAE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.	75
J.6	Normalized mean squared error (NMSE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.	76
J.7	Root mean squared error (RMSE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.	76

List of Tables

2.1	Summary of Forecast Evaluation Metrics with Descriptions	37
4.1	Summary of Performance Comparison between MERA and RM . . .	50

Acronyms

AE	Absolute Error
AI	Artificial Intelligence
ATM	Automated Teller Machine
CZ	CNOT gate
DTW	Dynamic Time Warping
GRU	Gated Recurrent Units
H	Hadamard gate
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MedAE	Median Absolute Error
MERA	Multiscale Entanglement Renormalization Ansatz
MSE	Mean Squared Error
NISQ	Noisy Intermediate-Scale Quantum
NMSE	Normalized Mean Squared Error
QML	Quantum Machine Learning
QRC	Quantum Reservoir Computing
RC	Reservoir Computing
RE	Relative Error
RM	Repeated Measurement
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent

Chapter 1

Introduction

Quantum computing represents one of the most promising frontiers of modern technology, with the potential to revolutionize the way we process information. Unlike classical computers, which rely on bits that can be either 0 or 1, quantum computers use quantum bits — or qubits — that can exist in superposition states. This property allows them to perform certain types of calculations much faster and more efficiently, opening the door to new possibilities in fields such as optimization, cryptography, and artificial intelligence.

At present, the available quantum devices are still limited in scale and stability; they belong to what is known as the Noisy Intermediate-Scale Quantum (NISQ) era. Because of these limitations, researchers are focusing on developing quantum algorithms capable of functioning effectively even with imperfect hardware .

Among the various research directions, quantum machine learning (QML) is gaining increasing attention. It combines the principles of quantum computing with the methods of classical machine learning, aiming to process and analyze data in new, more powerful ways. One particularly important challenge in this area involves temporal information processing, that is, analyzing data that change over time such as stock market trends, speech signals, or robotic sensor data.

In classical computing, these problems are typically addressed using recurrent neural networks (RNNs), which can “remember” past information by creating feedback connections within the network. This allows the model to learn patterns and dependencies that evolve over time. However, training these networks requires considerable computational resources and can be a very demanding process. Quantum approaches could, in the future, provide more efficient alternatives, reducing complexity while improving learning performance.

One possible approach to efficiently processing time-series data is Reservoir Computing (RC) [1], a machine learning framework that exploits the nonlinear dynamics and memory properties of complex systems. In this paradigm, an input signal is injected into a high-dimensional dynamical system — the reservoir — whose transient responses are then used to make predictions or classifications. Reservoir computing can be implemented using either artificial neural networks or physical systems that naturally exhibit dissipative dynamics.

Within this context, Quantum Reservoir Computing (QRC) [2] has emerged as a particularly promising direction. It leverages quantum systems as computational reservoirs, taking advantage of their intrinsic properties such as superposition and entanglement to enhance nonlinear information processing. However, traditional QRC models face significant challenges, especially regarding execution time and the loss of temporal memory due to the collapse of the quantum state during measurement. This phenomenon effectively erases the information encoded in the system's evolution, limiting the performance of these models in tasks requiring long-term dependencies. The project presented in this work addresses these challenges by developing, implementing, and evaluating Quantum Machine Learning algorithms within the conceptual framework of Quantum Reservoir Computing. In this approach, the quantum system acts as a dynamical reservoir that processes the input data through its intrinsic quantum evolution, enabling the extraction of complex temporal features; in this case, the study focuses on applying this methodology to a financial time-series forecasting case study. So, the research involves designing and implementing QML models, performing simulations within quantum computing frameworks, and benchmarking their performance against classical machine learning approaches.

The primary objective is to assess the potential advantages of quantum-based methods for addressing real-world financial problems, particularly in tasks related to data analysis and predictive modeling. Through this investigation, the work aims to provide insights into the practical feasibility of quantum computing as a tool for next-generation intelligent systems.

Within this framework, the different computational backends employed in the simulations can be interpreted as implementations of the reservoir system itself. In other words, each backend represents the physical or simulated quantum platform responsible for driving the system dynamics, processing the encoded input data, and ultimately generating the predictions through the reservoir computing mechanism. Before delving into the analysis of the project, it is essential to provide a concise overview of the underlying theoretical concepts to ensure a clearer understanding of the work carried out.

1.1 Quantum bit

The quantum bit, commonly known as the qubit, represents the fundamental unit of information in quantum computing and underlies the operation of all quantum devices. Unlike the classical bit, which can only assume one of two discrete values — ‘0’ or ‘1’ — a qubit can exist in a superposition of both states simultaneously [3]. This means that its value is not limited to these two extremes but can take on any combination within the complex plane, including imaginary components. Mathematically, the equation below represents a qubit in a continuum state [4]:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle, \quad \text{where} \quad |\alpha|^2 + |\beta|^2 = 1 \quad (1.1)$$

Amplitudes are generally complex numbers; however, to gain an intuitive understanding of their role, they can initially be considered as ordinary real numbers. In quantum mechanics, amplitudes provide the mathematical coefficients that describe the contribution of each basis state within a superposition, allowing all possible superposed states of a quantum system to be represented formally. So, we cannot definitively determine whether the qubit is in state $|0\rangle$ or $|1\rangle$. Here, $|0\rangle$ and $|1\rangle$ are the ground state represented by column vectors, α and β denote the amplitudes of the qubit, with measurement probabilities corresponding to $|\alpha|^2$ and $|\beta|^2$. Upon measurement, the probability of observing $|0\rangle$ is $|\alpha|^2$, while the probability of observing $|1\rangle$ is $|\beta|^2$.

Squaring α and β to find the probability is similar to squaring a wave's amplitude to find the energy of the wave. Since the total probability of observing all the states of the quantum system must add up to 100%, the amplitudes must obey to the normalization rule (i.e. the coefficients α and β can always be rescaled by some factor to normalize the quantum state).

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (1.2)$$

To visualize these properties, qubit states are typically represented on the Bloch sphere (Figure 1.1), where any point on the sphere's surface corresponds to a possible quantum state.

In essence, while classical computing relies on deterministic and finite values, quantum computing operates on states and operators — that is, vectors and matrices — reflecting the probabilistic and linear-algebraic nature of quantum mechanics.

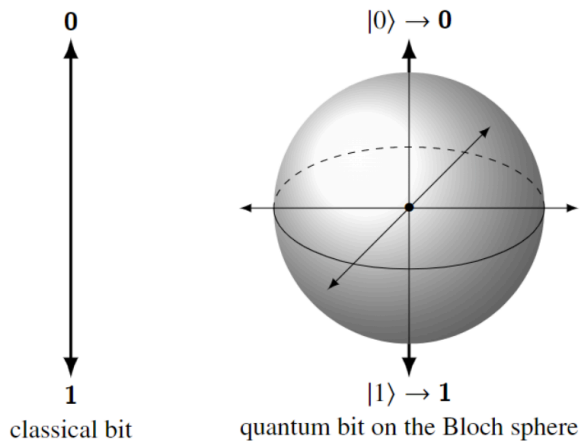


Figure 1.1: Classical and Quantum Information Representation.

To better understand the fundamental principles of quantum computing, it is essential to consider the following key points:

- **Superposition:** unlike classical bits, qubits can exist in a superposition of states, meaning that they can simultaneously represent both '0' and '1' with certain probabilities. This property enables quantum computers to explore

multiple possibilities at the same time, providing a form of parallelism that is unattainable in classical computing [5].

- **Quantum Logic Gates:** quantum computation also relies on logic gates, but unlike classical gates built from transistors, quantum gates are implemented using precise electromagnetic pulses that manipulate qubit states over time (for more details see Appendix D). Consequently, a quantum circuit evolves dynamically in time, rather than in space as in classical circuits. These gates are reversible, a feature that distinguishes quantum computation and allows for the preservation of information during intermediate computational steps [6].
- **Measurement and State Collapse:** upon measurement, a qubit's superposition collapses to a definite state — either 0 or 1 — eliminating the superposition and preventing direct restoration of the original quantum state (for more details see Appendix E). This inherent probabilistic nature of measurement is central to quantum mechanics and underlies the need for repeated experiments or sophisticated algorithms to extract meaningful information from quantum systems [7].
- **Entanglement and Correlation:** it is a uniquely quantum form of correlation that occurs when two or more quantum systems share a joint state that cannot be described independently. In an entangled system, the measurement of one particle instantaneously determines the state of the other, regardless of the distance between them. Unlike classical correlations, entanglement cannot be reproduced by any classical probabilistic model and represents one of the key resources of quantum information processing (for more details see Appendix F).
- **Additional Considerations:** it is important to underline that entanglement, combined with superposition, is what gives quantum computing its unique computational power, enabling complex tasks such as optimization, simulation of quantum systems, and certain types of machine learning that are beyond the reach of classical computers [8]. However, the design of quantum algorithms must account for decoherence and noise, which can disrupt quantum states and affect computational fidelity (for more details see Appendix G).

Noise refers to unwanted disturbances that affect the evolution of a quantum system during a computation. These disturbances can arise from imperfections in quantum gates, measurement errors, or interactions with the surrounding environment, leading to deviations from the ideal quantum state evolution.

Decoherence is a specific physical process in which a quantum system loses its coherence due to interactions with the environment. This process destroys quantum properties such as superposition and entanglement, gradually causing the system to behave more like a classical system.

Computational fidelity measures how accurately a quantum operation or computation reproduces the expected ideal result. A high fidelity indicates that the implemented quantum circuit closely matches the theoretical operation, while lower fidelity reflects the presence of noise and other hardware imperfections [9].

1.2 Quantum Machine Learning

Quantum Machine Learning (QML) is an emerging interdisciplinary field that combines quantum computing with machine learning (ML). The goal is to leverage the principles of quantum mechanics — such as superposition, entanglement, and quantum interference — to improve the performance of classical machine learning algorithms or to enable entirely new approaches that are infeasible for classical computers (for more details see Appendix A). In essence, the central question driving is how quantum computing can enhance and extend the capabilities of intelligent data mining [10]. Quantum systems can naturally encode data in high-dimensional Hilbert spaces, allowing more expressive representations of complex datasets. Additionally, superposition enables qubits to exist in multiple states simultaneously, allowing quantum algorithms to process many possibilities at once. Finally, entanglement generates correlations between qubits that have no equivalent in classical systems, giving QML models the ability to capture complex relationships and dependencies in data more effectively than classical.

So, the quantum approach serves to represent richer data correlations and potentially achieve improved learning performance in tasks such as pattern recognition, classification, and time-series prediction.

Quantum machine learning can be categorized into several approaches:

- Quantum-enhanced classical ML: classical algorithms are improved using quantum subroutines, for example, using quantum linear algebra routines to accelerate data processing in support vector machines or principal component analysis.
- Quantum-native ML: entirely quantum algorithms are designed, such as variational quantum classifiers or quantum neural networks, which process quantum-encoded data directly.
- Hybrid quantum-classical ML: these approaches use quantum circuits as components in classical ML pipelines. For instance, a variational quantum circuit acts as a feature map or function approximator, while the optimization and decision-making are handled classically.

In general, a QML circuit consists of two main components: a data encoding stage, responsible for mapping classical information into quantum states, and a variational ansatz, which performs parameterized quantum operations to process the encoded data (Figure 1.2 and Figure 1.3).

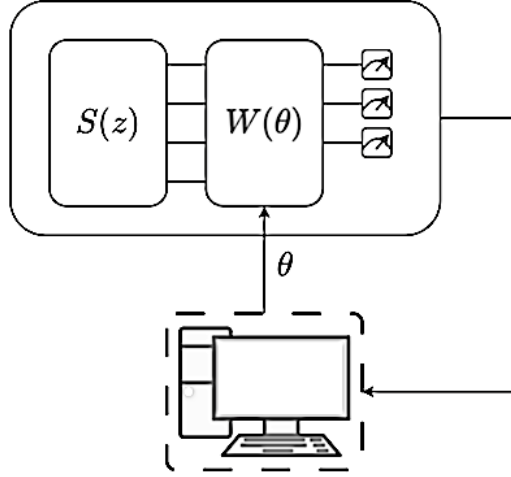


Figure 1.2: Illustrative Quantum Circuit with encoding and variational ansatz layers.

$S(z)$ represents the encoding circuit, which allows classical data to be embedded into a quantum system, enabling the quantum algorithm to process it. Proper data encoding is essential, as it directly affects the expressive power and efficiency of the quantum model. There are several commonly used encoding strategies, including:

- Basis Encoding: it associates each input with a computational basis state.
- Amplitude Encoding: it associates each feature as the amplitude probability of the state vector (if the input vector norm $\neq 1$, it must be normalized).
- Angle Encoding: it associates each input data as angles for rotational gates.
- Quantum Feature Maps: data is mapped into a high-dimensional Hilbert space using entangling gates, creating complex correlations that can enhance the model's ability to capture patterns in the data.

$W(\theta)$ represents the ansatz, a parameterized quantum circuit specifically designed to encode, process, and transform input data. The ansatz acts as a flexible template, whose parameters θ are adjusted during the training or optimization process to approximate the optimal solution to a given problem.

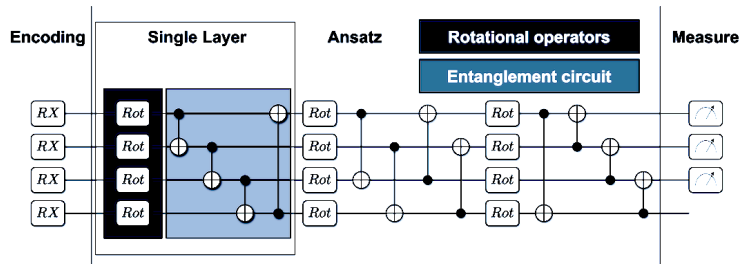


Figure 1.3: Example of a quantum circuit illustrating both the encoding and ansatz stages.

The choice of ansatz is critical, as it determines the expressive power of the quantum circuit, its ability to capture relevant features of the data, and ultimately the efficiency and accuracy of the algorithm. A well-designed ansatz balances an adequate level of expressivity while maintaining a reasonable circuit depth, allowing the reservoir to generate informative feature representations without exceeding the limitations of current quantum hardware.

For a sense of completeness, in the following (Figure 1.8), we provide an overview of the evolution of the quantum machine learning, highlighting the most influential contributions that have shaped its theoretical foundations and practical development [11].

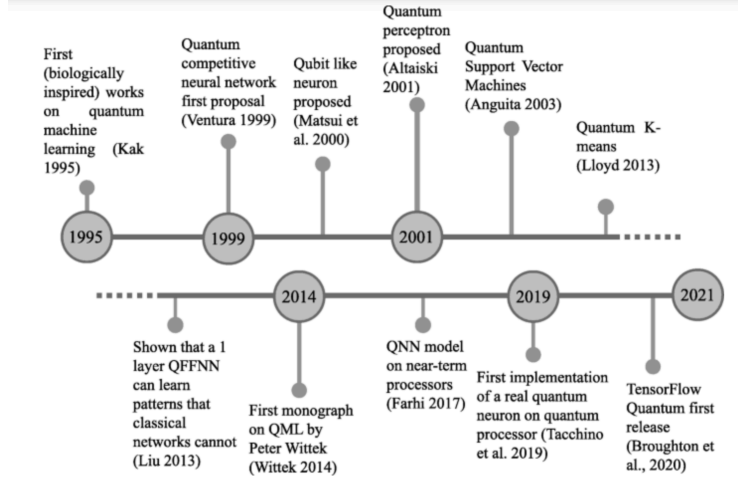


Figure 1.4: Timeline highlighting major advancements in quantum machine learning. It is possible to divide the history of quantum machine learning into two phases: the first one started during the mid-90s until 2007, and was mainly focused on models’ formulation, the second one, in which we currently are, is focused more on implementation [12]. Early works, dealing with quantum neural networks, were biologically inspired [13] [14] [15], with the hope of finding quantum-rooted explanations in the working principles of the brain. In the early 2000s, the question of statistical learning in a quantum framework was discovered [16] [17] [18]. The first monograph on the subject was published in 2014 [19]. In the following years, interest in the topic increased, producing a vast body of literature. In 2020, the first version of Tensorflow Quantum was released by Google [20], as a part of the Tensorflow platform. Furthermore, cloud-based models began to allow for the implementation of QML models on actual quantum computers.

1.3 Quantum Circuit Ansatz

Quantum circuit ansatzes refer to specific choices of quantum circuits with adjustable parameters. These circuits are designed to prepare or approximate a target quantum state suitable for a particular computational problem. It is implemented as a parameterized quantum state designed to provide an informed starting point for tackling a specific problem, such as determining the ground state of a quantum system.

Practically, an ansatz takes the form of a quantum circuit whose structure — comprising the number of qubits, sequence of operations, and types of gates — is carefully designed to align with the characteristics of the task. Its architecture may be guided by prior knowledge of the system or by physical intuition. In Quantum Reservoir Computing, the circuit parameters are typically fixed and serve to generate a rich and nonlinear quantum dynamics, from which informative features can be extracted and subsequently processed by a classical learning algorithm.

As the algorithm progresses, the ansatz can be iteratively refined to incorporate new information and improve convergence. Below, the Multiscale Entanglement Renormalization Ansatz (MERA) and the Repeated Measurement (RM) technique are examined and adopted, as they represent particularly suitable approaches for time-series prediction tasks.

1.3.1 Multiscale Entanglement Renormalization Ansatz (MERA)

The development of efficient algorithms [21] for simulating quantum many-body systems relies heavily on the use of tensor networks as compact representations of complex quantum states. A central requirement in this context is the ability to compute the expectation value of local observables efficiently from these tensor network structures. To address this challenge, the Multi-scale Entanglement Renormalization Ansatz (MERA) (Figure 1.5) was introduced as a framework capable of efficiently encoding the quantum many-body state of a D-dimensional lattice and accurately evaluating local expectation values [22].

MERA provides a structured way to describe quantum states by exploiting the underlying entanglement patterns present in many-body systems. Its architecture consists of multiple layers of tensors, each capturing entanglement at a different length scale. As a result, MERA organizes quantum correlations hierarchically — ranging from microscopic to macroscopic scales — enabling a detailed and scalable representation of the system’s entanglement structure. This multi-layer design makes MERA particularly effective for studying the intrinsic properties, correlations, and emergent behaviors of quantum many-body states [23].

In the context of D-dimensional lattice models, MERA is especially valuable when the goal is to compute local observable expectation values both accurately and with reduced computational cost. Furthermore, it forms the conceptual foundation of entanglement renormalization, an advanced coarse-graining technique for lattice-based quantum systems that preserves essential entanglement features while simplifying the system’s description. This framework has therefore become a powerful tool for analyzing strongly correlated quantum systems across different scales.

1.3.2 Repeated Measurement (RM)

The Repeated Measurement approach involves injecting input data into a quantum system, allowing it to evolve for a short period, and then performing a measurement at every time step. Although each measurement collapses the state, the repeated

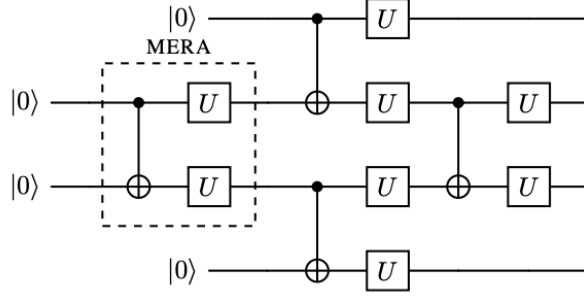


Figure 1.5: Multiscale Entanglement Renormalization Ansatz scheme [24].

interaction between input encoding, system dynamics, and measurement outcomes allows the system to generate a rich time series of nonlinear responses.

Instead of relying on the quantum state to retain information, the technique relies on the intrinsic dynamics of the quantum system, the statistical properties of measurement outcomes and the sequence of consecutive measurements to encode temporal relationships in the data [25].

The output time series is constructed by computing the average measurement outcome of all qubits at each timestep. To efficiently access intermediate states of the reservoir, mid-circuit measurements and reset operations are employed. This approach eliminates the need to repeatedly initialize and evolve the quantum system from its initial state for every timestep, enabling the extraction of temporal information while maintaining the ongoing reservoir dynamics (Figure 1.6).

This technique fits very well in the context of the Quantum Reservoir Computing, since the reservoir does not need to store explicit internal memory; instead, the dynamics themselves should produce features that implicitly reflect the influence of past inputs [26].

Repeated Measurement satisfies this requirement because:

- Quantum dynamics evolve continuously, so each input perturbs the system in a way influenced by previous states.
- Even though each measurement collapses the state, the sequence of collapse outcomes forms a high-dimensional, temporally-structured signal.
- The measurement noise and quantum randomness introduce useful nonlinearities, letting the readout layer learn complex temporal patterns.
- The process is relatively simple to implement, as it does not require deep quantum circuits or highly complex entangling operations. Consequently, it is generally compatible with current NISQ hardware. However, for long temporal sequences, the repeated application of quantum operations and measurements may lead to the accumulation of noise and decoherence effects, which can ultimately limit the reliability of the computation.

In summary, the Repeated Measurement technique allows Quantum Reservoir Computing to make effective use of noisy, short-coherence quantum systems by turning measurement outcomes over time into the reservoir’s feature space. Instead of avoiding measurement-induced collapse, the technique embraces it as a computational resource, enabling temporal processing even with very limited quantum hardware.

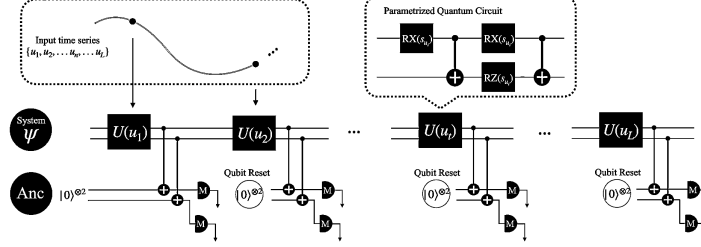


Figure 1.6: Repeated Measurement scheme [26].

1.4 Reservoir Computing

Reservoir computing (RC) is a computational paradigm specifically designed for processing temporal or sequential data. It originates from recurrent neural network (RNN) models such as Echo State Networks (ESNs) [27] and Liquid State Machines (LSMs) [28], and its core idea is to exploit the complex dynamics of a nonlinear system — called the reservoir — to transform input signals into a high-dimensional representation (Figure 1.7).

The system is composed of two main parts: the reservoir, which projects input data into a rich dynamic space, and the readout layer, which extracts relevant information and performs pattern recognition or prediction tasks based on the reservoir’s internal states. A key feature of reservoir computing is that only the readout layer is trained, typically using simple linear methods such as regression or classification [29]. The internal reservoir dynamics remain fixed, drastically reducing computational costs and training time compared to traditional RNNs (for more details see Appendix B). This property also makes RC particularly suitable for hardware implementations, since the reservoir can be realized using a variety of physical systems — optical, electronic, mechanical, or even biological — without the need for continuous parameter optimization.

The simplest implementation of a reservoir computer consists of a network of N sparsely connected nonlinear nodes, into which data are sequentially fed in the form of a time series [30].

Let $\mathbf{u}(t)$ denote the $N \times 1$ input vector at time t . The signal injected into the reservoir nodes is then given by $\mathbf{W}_{\text{in}}\mathbf{u}(t)$, where $\mathbf{W}_{\text{in}}$ represents the $N \times N$ input weight matrix. In practical implementations, $\mathbf{W}_{\text{in}}$ is fixed for a given task. Moreover, if the input data are connected only to a subset of reservoir nodes, the dimension N can be replaced by N_{in} , representing the number of nodes that directly receive the input.

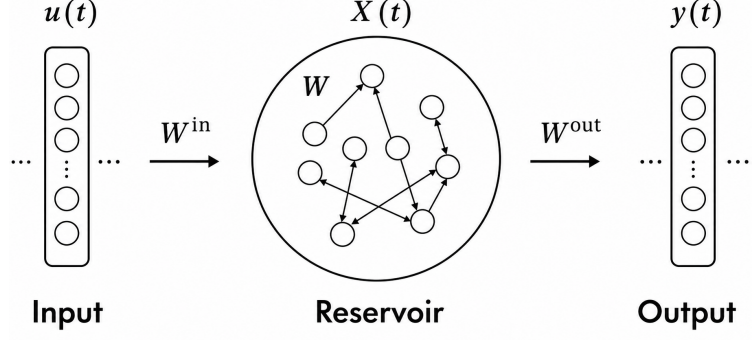


Figure 1.7: Example of a graphical diagram of a Reservoir Computing.

Without loss of generality, each reservoir node can be assumed to possess a single degree of freedom. Under this assumption, the complete internal state of the reservoir at time t is described by the vector $\mathbf{x}(t)$. The temporal evolution of the reservoir state is governed by a nonlinear dynamical equation of the form:

$$\frac{\partial \mathbf{x}(t)}{\partial t} = f(\mathbf{W}_{\text{res}} \mathbf{x}(t), \mathbf{W}_{\text{in}} \mathbf{u}(t), \mathbf{W}_{\text{fb}} \mathbf{x}(t - \tau)) \quad (1.3)$$

The function $f(\mathbf{a}, \mathbf{b}, \mathbf{c})$ represents a nonlinear transformation governing the reservoir dynamics [30]. The first argument models the internal interactions among the reservoir nodes, described by the internal weight matrix $\mathbf{W}_{\text{res}}$, which also accounts for the intrinsic decay of each node's state. The second argument represents the contribution of the external input signal entering the reservoir, as discussed previously. The third argument corresponds to the time-delay feedback term, characterized by the feedback weight matrix $\mathbf{W}_{\text{fb}}$, which introduces delayed interactions within the reservoir.

We denote the reservoir output signal by $\mathbf{s}(t)$, which in many implementations coincides with the internal state vector $\mathbf{x}(t)$. To perform computation with the reservoir, the time-dependent outputs of all N reservoir nodes are discretized over P time steps and organized into an $N \times P$ matrix $\mathbf{s}_t$. For convenience, this matrix is then column-stacked into a vector of dimension $NP \times 1$, denoted as $\vec{\mathbf{s}}_t$.

The final stage of the reservoir computing framework consists of applying the output weight matrix $\mathbf{W}_{\text{out}}$ of size $L \times NP$ to this vector in order to produce the computed output:

$$\mathbf{y} = \mathbf{W}_{\text{out}} \vec{\mathbf{s}}_t \quad (1.4)$$

Here, $\mathbf{y}$ is an $L \times 1$ vector representing the task output and ideally corresponds to the desired solution of the problem addressed by the reservoir computer. Similarly to the input layer, if the output signal is collected from only a subset of reservoir nodes N_{out} , the output weight matrix can be reduced to a size $L \times (N_{\text{out}}P)$.

In reservoir computing, the only trainable parameters are the elements of the output weight matrix $\mathbf{W}_{\text{out}}$, which are optimized using a supervised learning procedure [31]. So, for example, the training can be performed using the standard ridge regression

approach, given by:

$$\mathbf{W}_{\text{out}} = \mathbf{Y}\mathbf{S}_t^\top (\mathbf{S}_t\mathbf{S}_t^\top + \gamma\mathbf{I})^{-1}, \quad (1.5)$$

where $\mathbf{Y}$ and $\mathbf{S}_t$ are matrices of size $L \times M$ and $(N_{\text{out}}P) \times M$, respectively, containing the data corresponding to M training instances of the task. The matrix $\mathbf{Y}$ stores the target outputs associated with each training example, while $\mathbf{S}_t$ contains the reservoir output signals $\mathbf{s}_t$ generated for the corresponding inputs. The parameter γ represents the ridge regression regularization coefficient, which is introduced to mitigate overfitting.

One of the main advantages of reservoir computing is that the training phase requires only a single matrix inversion, resulting in a significant computational advantage compared to traditional neural network training procedures. Moreover, the adopted training strategy allows the model to exploit correlations present in the reservoir outputs both across different nodes at a given time step and across multiple time steps. This capability arises from the use of the full output vector $\mathbf{s}_t$ during training and has been observed to yield improved predictive performance.

If further reductions in computational cost are required, the structure of the output weight matrix $\mathbf{W}_{\text{out}}$ can be constrained to a block-diagonal form, preventing interactions between reservoir outputs corresponding to different time steps and thereby simplifying the training procedure.

1.5 Quantum Reservoir Computing

Quantum systems offer an exceptionally promising platform for implementing reservoir computing [26], giving rise to the concept of Quantum Reservoir Computing (QRC). Quantum systems naturally exhibit intrinsic properties — such as superposition, entanglement, and nonlinearity through measurement-induced dynamics — that can be leveraged to create highly expressive and high-dimensional state spaces. These characteristics enable the reservoir to encode temporal correlations and nonlinear dependencies in ways that classical systems cannot easily replicate [32]. In fact, the evolution of quantum states over time inherently provides a temporal processing capability, making quantum systems an ideal substrate for dynamic computation tasks (Figure 1.8).

Moreover, the interaction between qubits, governed by unitary transformations and decoherence effects [33], introduces a natural source of complexity and memory — two essential ingredients for an effective reservoir. By exploiting these phenomena, a quantum reservoir can potentially achieve superior performance in time-series prediction, pattern recognition, and optimization tasks while maintaining the fast learning efficiency typical of classical RC.

As quantum hardware continues to mature, quantum reservoir computing could serve as a bridge between classical machine learning and full-scale quantum computation, providing a practical and physically realizable model [32] for near-term quantum

devices (Noisy Intermediate-Scale Quantum systems, or NISQ). This makes QRC not only a fascinating research direction but also a viable strategy for harnessing the computational advantages of quantum mechanics in real-world machine learning applications.

Like classical reservoir, one of the main advantages of quantum reservoir computing is that only the output layer requires training. The reservoir itself remains fixed, while the output is obtained by learning a linear combination of the reservoir responses to the input signals. Given a set of reservoir observables $\mathbf{x}_k$ selected as outputs, the predicted output can be written as $y_k = \mathbf{w}^\top \mathbf{x}_k$, where $\mathbf{w}$ represents the trainable weight vector. These weights are optimized by minimizing the error with respect to a target output $\bar{y}_k$. In general, the training of the output weights is typically performed using linear regression techniques [34].

However, a fundamental question in classical and quantum reservoir computing concerns the characteristics that a physical system must possess in order to function effectively as a reservoir [35]. Identifying the conditions that enable a system to provide sufficient computational capability, memory, and nonlinear dynamics remains an open research problem.

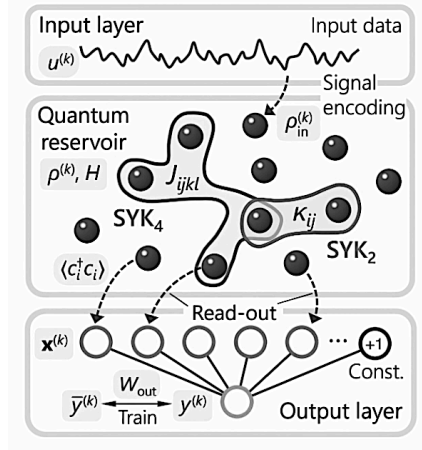


Figure 1.8: Example of a graphical diagram of a Quantum Reservoir Computing.

To evaluate the performance of the reservoir, typically two specific benchmark tasks are considered together with a more general indicator of the system’s information-processing capacity. As a first benchmark, the nonlinear autoregressive moving average (NARMA) model [36] is used (for more details see Appendix C), which is widely employed to assess the performance of recurrent neural networks [37].

The general NARMA_n task is defined by:

$$\bar{y}_k = 0.3 \bar{y}_{k-1} + 0.05 \bar{y}_{k-1} \left(\sum_{j=1}^n \bar{y}_{k-j} \right) + 1.5 s_{k-n} s_{k-1} + 0.1, \quad (1.6)$$

where n represents the maximum delay of the system, $\bar{y}_k$ is the target output at time step k , and s_k denotes a random input signal driving the system.

The reservoir computer must learn to reproduce the dynamics described by the

previous equation using the random input sequence s_k . In other words, the system is required to approximate a quadratic nonlinear function of the input sequence that depends on past inputs up to a fixed delay. For instance, the maximum delay can be set to $n = 10$.

To evaluate the reservoir’s capability to process both nonlinear and linear temporal dependencies, a linear memory task can also be considered, defined as $\bar{y}_k = s_{k-\tau}$, where the delay is fixed to $\tau = 10$.

In both tasks, the choice of the value 10 determines the required memory depth of the system. Specifically, the reservoir must retain information about at least the ten previous input values in order to successfully reconstruct the target output at each time step k [36].

So, quantum reservoir computing offers a promising framework for temporal information processing by exploiting the intrinsic dynamics of quantum systems and a key advantage of QRC lies in the exponentially large dimensionality of the Hilbert space, which enables complex transformations of input data and can lead to high computational performance. However, the effectiveness of a quantum system as a reservoir strongly depends on the operational regime in which it is driven. Proper tuning of the reservoir parameters is therefore essential to ensure a balance between dynamical richness and stability, allowing the system to generate diverse and informative responses to input signals.

Another important factor influencing performance is the topology of the network underlying the reservoir. While it is widely recognized that random connectivity can enhance performance by preventing redundancies among the degrees of freedom, randomness alone is not sufficient. In particular, network structures that introduce disorder while simultaneously generating an extensive number of conserved quantities may hinder the reservoir’s computational capability, as they restrict the effective exploration of the system’s state space.

Practical implementations of QRC raise additional challenges related to quantum measurements and the extraction of useful information from the system. Developing efficient strategies for online data processing that properly account for measurement back-action and limited observability remains an open and important research direction. Addressing these issues will be crucial for advancing the practical realization and scalability of quantum reservoir computing architectures.

Chapter 2

Method

This work focuses on the development of a Quantum Machine Learning model designed for time-series forecasting on financial data. The main objective is to analyze the behavior of the proposed model in detail and to evaluate the potential improvements it can offer compared to a corresponding classical forecasting approach. The following chapter is organized as follows. First, the dataset used in the study is described and, next, the implementation of the classical algorithm is presented. Subsequently, the development of the quantum models is explained and, finally, the evaluation metrics employed to assess the predictive performance of the models are introduced.

The simulation results, instead, will be analyzed and compared in the next chapter, along with a concluding discussion of the findings.

2.1 Description of the dataset

The tabulated data were analyzed starting from a public dataset available on the web [38].

This dataset has been created to predict the amount of money in the ATMs and it is composed of 12 features in total. Among these, five are categorical attributes stored as String values, six are numerical attributes represented as Integers, and one is a temporal attribute of type Timestamp. Importantly, the dataset contains no missing or null entries, ensuring data integrity and eliminating the need for preprocessing steps related to imputation or cleaning.

Overall, the dataset includes approximately 30,000 transaction records originating from a network of 359 Automated Teller Machines (ATMs). The data spans a continuous period of one week, with each ATM providing operational or transactional information every two hours. Given that one week comprises 168 hours, each ATM is represented by exactly 84 instances, resulting in a uniform temporal resolution across the entire dataset. This structure makes the dataset particularly suitable for time-series forecasting tasks, anomaly detection, and spatiotemporal analysis.

Furthermore, the initialized total amount of every ATM is 40000 and the amount of money changes over time in accordance with the transactions made.

To import the dataset, you can download the CSV file or use the following command lines in the code:

```
1 import kagglehub
2 path = kagglehub.dataset_download("yarenyilmaz/atm-transactions")
3 print("Path to dataset files:", path)
```

The project focuses on analyzing the temporal behavior of each individual ATM in the dataset, using the values contained in the 'totalBalance' column as the primary variable of interest; furthermore, as already mentioned, for each ATM 84 sequential data points are available.

Based on this structure, the time series is partitioned into two segments: 64 samples are allocated for model training, while the remaining 20 samples are reserved for testing and performance evaluation.

The forecasting procedure is iterative: at each step, the model predicts the ten subsequent timesteps following the last observed value; then, the window is shifted by one position along the time series (Figure 2.1), and the prediction is repeated. This sliding-window approach continues until the 74th data point is reached, beyond which all model predictions extend to the end of the available sequence. Such a strategy allows the model to produce multi-step forecasts while maintaining coherence with the temporal structure of the dataset. For a clearer understanding, below is a small recap.

- 1. Data setup:

Each ATM has a time series of 84 points:

- Training set: first 64 points
- Test set: last 20 points

- 2. Prediction steps:

- The model uses the first 64 points to predict the next 10 timesteps (points 65–74).
- After predicting 10 steps, the window moves forward by one timestep.
- The model now predicts the next 10 points, starting from point 65.
- This process repeats iteratively until predictions reach the end of the time series.

- 3. Advantages:

- Maintains temporal coherence
- Produces multi-step forecasts
- Ensures all points in the test set are eventually predicted

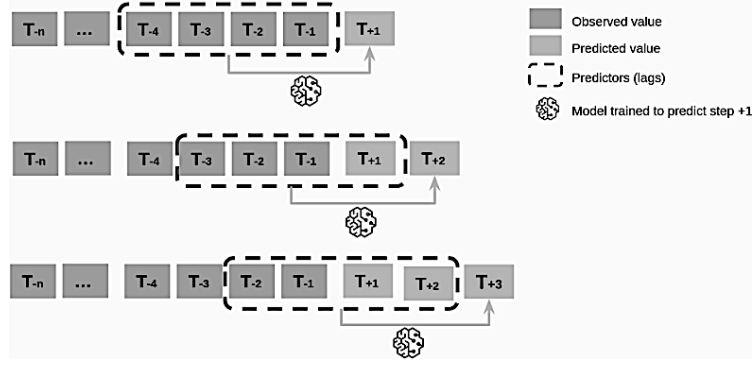


Figure 2.1: Graphical diagram of an Iterative Multi-Step Forecasting (Sliding Window Approach).

2.2 Classical algorithm

In this section, the analysis and design of the classical computational algorithm used for time-series forecasting are presented. In particular, the open-source software Prophet, developed by Facebook’s Core Data Science team, was adopted for this purpose [39].

Prophet is an additive model-based approach for forecasting time series data that matches non-linear patterns with yearly, weekly, and daily seasonality, as well as the holiday effect. It employs a decomposable model that consists of three primary components: trend, seasonality, and holiday effects [40]. Through its implementation, you get reasonable prediction on messy data without any manual intervention as it is robust against outliers, missing data and drastic changes in your time series. Ultimately, in most cases, it outperforms any other classical approach (for more details see Appendix I).

For the sake of completeness, an example of using the package in code has been reported below:

```

1 from prophet import Prophet
2 m = Prophet()
3 m.fit(df) # df is a pandas.DataFrame with 'y' and 'ds' columns
4 future = m.make_future_dataframe(periods=window_size, freq='h') # 'h'
   = 'hours'
5 forecast = m.predict(future)
6 m.plot(forecast)
7 m.plot_components(forecast)

```

The simulation generates predictive results for every ATM included in the dataset. However, given the large number of ATMs (359 in total), it is impractical to present the outcomes for each one individually. Therefore, the analysis will focus on a representative case: the ATM with ‘atmId’ = 350002, whose performance will be examined in detail from this point onward.

The graph of the time forecasts processed with Prophet is shown below (Figure 2.2).

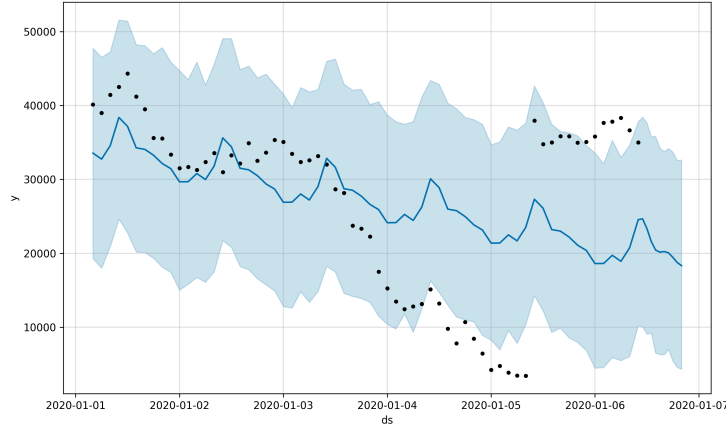


Figure 2.2: Prophet previsions.

It shows the time trend of totalBalance values, alongside the forecasts generated by the model and the related uncertainty. The horizontal axis (ds) represents the time scale, between January 1, 2020, and January 7, 2020. The vertical axis (y) indicates the numerical value of 'totalBalance', i.e., the amount of money in the ATM at each time step.

The graph contains several elements including black dots, a blue line and a light blue band.

The black dots represent the actual values recorded by the ATM over the period; they are arranged irregularly, showing significant variations from one hour to the next. Looking at it, we can notice that initially there is an initial phase with relatively high values (around 40000), followed by a progressive and marked decline to values close to 10000 and a subsequent rise towards the end of the interval, with values again close to 35000 – 38000. This dynamic suggests typical recharge and withdrawal cycles.

The blue line, on the other hand, represents the average estimate produced by the predictive model. It follows a smoother dynamic than the observed data points, capturing the overall trend but avoiding sudden fluctuations. In particular, the model captures the downward trend observed during the first few days and preserves the seasonal structure characterized by slow and regular oscillations. Although it does not reproduce the exact peaks of the real data, it successfully approximates their overall direction.

Finally, the colored band around the blue line represents the forecast confidence interval.

The width of the band indicates the degree of model uncertainty and, here, it appears quite wide, indicating that the model recognizes the high variability of the data. Moreover, it increases especially in areas without observed data, where uncertainty increases; this is typical of forecasting models when they must make forecasts on noisy and highly seasonal data.

To summarize, the model successfully captures the overall trend and seasonal structure, but fails to reproduce the most extreme peaks in real data; the wide uncertainty

band confirms that the problem is complex and characterized by strong volatility. After, with Prophet's "plot_components()" method, you can get a graph similar to the one below (Figure 2.3)

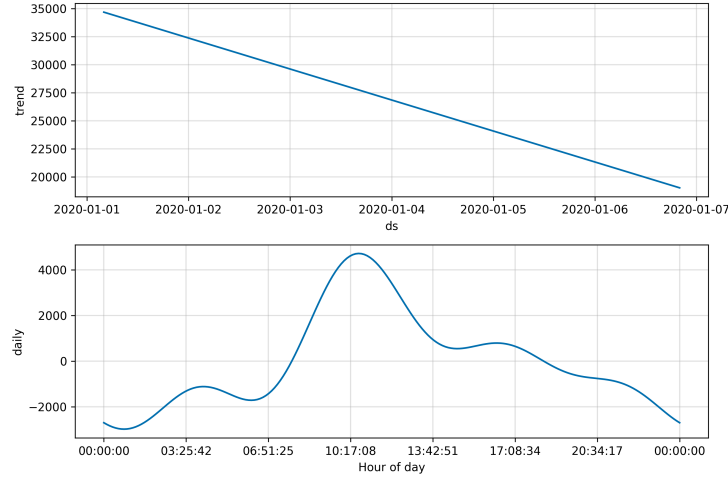


Figure 2.3: Prophet trends.

Prophet's - plot_components() - method is used to display the individual components that make up the forecast generated by the model. It typically provides two or more graphs, each showing a specific component of the model, derived from the decomposition:

$$y(t) = trend(t) + seasonality(t) + holidays(t) + error(t) \quad (2.1)$$

In the graph, the trend represents the overall increase or decrease in the ATM balance over time. This component corresponds to the non-seasonal part of the model and is typically modeled using either a piecewise linear or logistic function (Figure 2.3). In particular, we can see changes in slope (changepoints) that suggest that if the curve rises, the phenomenon is increasing over time; if it falls, it is decreasing. Finally, if it has "elbows," this means that Prophet has detected structural changes.

The 'daily' section, instead, plots the change in the ATM balance over the course of the day. This shows the periods during which the balance value increases or decreases on average.

For the sake of completeness, the final plot comparing the predicted and real values during the test period is presented below (Figure 2.4). It is included in this section to provide an initial visual reference for the model's predictive behavior. For consistency and comparative purposes, the same plot will also be reported later in the "Results" chapter, where it will be discussed alongside the outcomes produced by the quantum approaches.

The graph shows the overall trend generated by the predictions of the Prophet model. To construct this unique, structured predicted sequence, we proceeded by considering these steps:

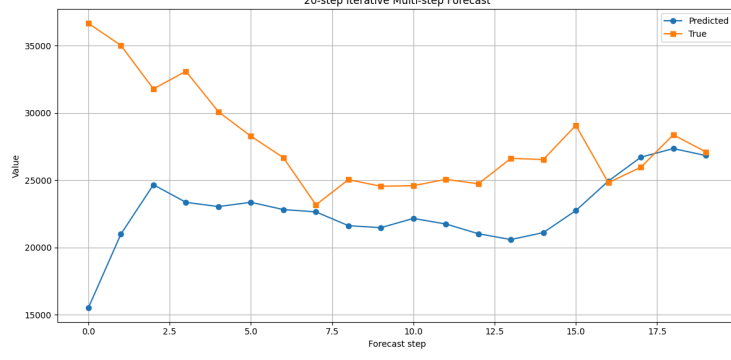


Figure 2.4: Prophet final time-series prediction.

- From each iteration (except the last), take only the first predicted value (`all_predictions[i][0]`).
- From the last iteration, take all the predicted values (`all_predictions[-1]`).
- Repeat the same steps for the real labels.

Prophet's model approximates the real values quite well, even though there are some flaws in the predicted values, especially in the initial phases of the plot. Subsequently, the curve gets closer and closer to the real case, with an appropriately low error rate.

2.3 Quantum algorithms

The transition from classical code to quantum computing is described in this section; the dataset division and other initial considerations remain unchanged. What changes is the way we compute the features to pass to the prediction model. As already explained, we will use the concept of quantum reservoir computing, made possible by the implementation of the two approaches MERA and Repeated Measurement (hereafter referred to as "RM"), which will provide different results precisely because of their different structure in terms of circuit gates.

In order to apply quantum machine learning techniques to the considered time-series data, a number of preprocessing steps are required to ensure that the data are compatible with the quantum computational framework.

The first and most important step consists of scaling the time-series values extracted from the dataset to the interval $[0,1]$. This normalization procedure is necessary because quantum circuits often require input parameters to lie within a bounded numerical range in order to ensure stable encoding within the quantum states. By mapping the original data into the $[0,1]$ interval, the time-series values can be more effectively embedded into the quantum circuit through the chosen encoding scheme. This preprocessing step therefore represents the starting point for the transition from the classical forecasting framework to the quantum reservoir computing models employed in the analysis:

```

1 from sklearn.preprocessing import MinMaxScaler
2 # take the first 64 values for the train.
3 y_train = history_df[t0_train:t][atm_id]
4 u_train = y_train.values.reshape(-1, 1)
5 # scale timeseries between 0 and 1
6 scaler = MinMaxScaler(feature_range=(0, 1))
7 scaler.fit(u_train)
8 u_train = scaler.transform(u_train)

```

Subsequently, the time series is fed into the quantum reservoir, where it is processed through the intrinsic dynamical evolution of the quantum system:

```

1 from reservoirs import QRC
2 # define the quantum reservoir
3 q_reservoir = QRC(n_qubits=4, scale_input=np.pi, backend=AerSimulator
4                   ())
5 # pass the data to the machine
6 x_train = q_reservoir.run(u_train, shots=shots)

```

At this stage, the last element of the x_{train} list and the first element of the y_{train} list are removed in order to correctly align the input sequences with their corresponding target values. This operation ensures that each input window is associated with the value to be predicted at the next time step, which is a required condition for applying the iterative multi-step forecasting approach.

The new lists will then be used to obtain a model score (you can select the model that you prefer):

```

1 from sklearn.linear_model import LinearRegression
2 # exclude the last sample since the prediction is the value at the
   following timestep
3 x_train_model = x_train[:-1]
4 # exclude the first sample, the target are at the following timestep
5 y_train_model = y_train[1:]
6 # choose the model that you prefer, fit and evaluate the score
7 model = LinearRegression()
8 model.fit(x_train_model, y_train)
9 score = model.score(x_train_model, y_train_model)

```

Finally, the outputs generated by the quantum reservoir are used to produce the predicted values, which are then employed in the testing phase of the model:

```

1 import numpy as np
2 # Predict on the test data
3 predictions = res_st.predict(
4     num_pred=window_size,
5     model=model,

```

```

6         shots=shots ,
7         from_series=u_train ,
8     )
9     # extract the predicted useful values
10    x_test = predictions[-window_size:]
11    # convert the compressed time series between 0 and 1 back to the
        required values
12    prediction = x_test.flatten()
13    # extract the real values for the test from the dataset
14    y_test = history_df[t:t + window_size][atm_id]
15    label = y_test.to_numpy()
16    # save into lists
17    all_predictions.append(prediction)
18    all_labels.append(label)

```

The elements described above represent the essential steps needed to generate predictions and evaluate their performance relative to the classical approach. It is important to note, however, that several implementation details have been intentionally omitted at this stage. These details are listed below and will be examined individually in the following sub-sections.

- There are several implementations of quantum reservoirs and, in the project, two classes are analyzed and tested: the 'Static' reservoir and the 'Incremental' one.
- The simulations were conducted by evaluating the performances of the RM and MERA systems separately.
- The simulations used three different backends: the AerSimulator, the IQM Adonis fake provider (similar to AerSimulator but including the noise model), and the IQM Spark [41].
- The way results are generated depends on the implementation of the 'predict' method in the Python code, which is responsible for producing the final outputs starting from the results of the Quantum Reservoir Computing model. These outputs are then used as input features for a subsequent regression model. In this project, three different implementations of the 'predict' method were tested.
- LinearRegression was ultimately chosen as the model for fitting the data, following a series of comparative tests conducted to evaluate and confirm its suitability as the primary approach.
- The performance of the results was analyzed by considering different metrics for the estimation of errors.

2.3.1 QuantumReservoirPy

The use of quantum circuits as reservoirs within the reservoir computing framework is a relatively recent development. Most examples currently found in the literature are

based on highly periodic quantum circuit architectures. The QuantumReservoirPy package [42] was developed to streamline the design and simulation of these periodic quantum circuits. It is distributed as a pip-installable Python package and can be obtained directly from PyPI [43].

```
1 pip install quantumreservoirpy
```

This line will install QuantumReservoirPy along with its dependent packages.

The design of the package’s interface is partially inspired by the ReservoirPy framework [44], and reviewing their tutorials can provide useful context for understanding its structure and usage.

Within QuantumReservoirPy, a reservoir is defined as a class derived from the abstract base class QReservoir. To build a fully custom reservoir, users are required to implement five core methods in their subclass implementation (Figure 2.5).

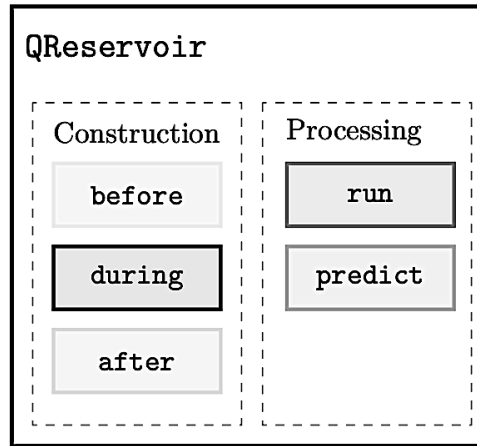


Figure 2.5: The five functions of QReservoir.

```
1 class CustomRes(QReservoir):
2     def before(self, circuit):
3         pass
4     def during(self, circuit, timestep):
5         pass
6     def after(self, circuit):
7         pass
8
9     def run(self, timeseries, **kwargs):
10        pass
11    def predict(self, num_pred, model, from_series, **kwargs):
12        pass
```

QuantumReservoirPy already includes several partially implemented reservoir models with simplified interfaces. In particular, the Static and Incremental reservoir classes provide ready-made implementations of the run and predict methods. As a result,

the user only needs to define the behavior of three components: before, during, and after.

All reservoirs derived from the Static and Incremental classes share the same three-layer quantum circuit structure:

- Initialization layer, defined in the 'before' method, responsible for preparing the reservoir state.
- Dynamic layer, a small quantum circuit for each timestep in the time series, defined in the 'during' method.
- Readout layer, applied after processing the full sequence, defined in the 'after' method.

This modular design (Figure 2.6) allows users to focus on the core logic of their reservoir while relying on a consistent and reusable execution flow:

```

1 from quantumreservoirpy.reservoirs import Static
2 class CustomRes(Static):
3     def before(self, circuit):
4         circuit.h(circuit.qubits)
5         circuit.barrier()
6
7     def during(self, circuit, timestep):
8         circuit.initialize(str(timestep), [0])
9         circuit.h(0)
10        circuit.cx(0, 1)
11
12    def after(self, circuit):
13        circuit.barrier()
14        circuit.measure_all()
15
16 res = CustomRes(n_qubits=2)
17 res.circuit([0, 1]).draw('mpl')

```

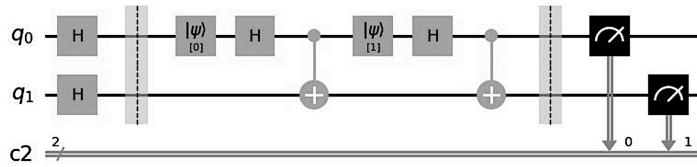


Figure 2.6: Static and Incremental three layered circuit structure.

The before, during, and after methods serve the same purpose in both Static and Incremental reservoir implementations; the key distinction lies in how each reservoir type executes the computation.

Once a reservoir instance is created, the time-series data can be processed simply by calling `reservoir.run()`, which returns a NumPy array containing the reservoir states at each timestep:

```
1 states = res.run(timeseries)
```

In a Static reservoir, the full quantum circuit is executed once over the entire time series. All measurements are then reshaped into an output array of dimensions $(\text{len}(\text{timeseries}), -1)$, where each row represents the reservoir state at a given timestep. In an Incremental reservoir, the circuit is built and executed progressively. At each timestep, only the last M samples of the time series are encoded and processed (with M defined in the Incremental.__init__ parameters, under the name 'memory'). This strategy reduces computational overhead for long sequences and enables efficient streaming-style processing. For additional implementation details and usage examples, please refer to the official documentation [45].

2.3.2 Construction of the MERA and RM ansatz

In the "Introduction" chapter, we provided a conceptual comparison between MERA and Repeated Measurement (RM) circuits, highlighting their structural and operational differences.

We now examine how these architectures are translated into Python code, with a focus on design choices and parameter settings.

For the MERA ansatz, the implementation shown below reflects its hierarchical tensor-network structure. The number of qubits, the depth of the network and the choice of entangling gates are exposed as configurable parameters, allowing the circuit to be adapted to different input dimensions and degrees of expressivity:

```
1 class QRC_MERA(Incremental):
2     def __init__(self, n_qubits, scale_input, memory=np.inf, backend=None,
3         coupling_map=None, layout=None, num_features=-1) -> None:
4         super().__init__(n_qubits, memory, backend, coupling_map, layout,
5             num_features)
6         self.n_qubits = n_qubits
7         self.scale_input = scale_input
8         self.coupling_map = coupling_map
9         self.layout = layout
10
11     def before(self, circuit):
12         for qb in range(self.n_qubits):
13             circuit.initialize([1, 0], qb)
14
15     def during(self, circuit, timestep, reservoirnumber):
16         angle = self.scale_input * timestep[0]
17         sys_qr = [0, 1]
18         anc_qr = [2, 3]
19         circuit.cx(sys_qr[0], sys_qr[1])
20         for qb in sys_qr:
21             circuit.append(UGate(angle, angle, angle), [qb])
22         circuit.cx(anc_qr[0], sys_qr[0])
23         for qb in anc_qr:
```

```

22         circuit.append(UGate(angle, angle, angle), [qb])
23     for qb in sys_qr:
24         circuit.append(UGate(angle, angle, angle), [qb])
25     circuit.cx(sys_qr[0], sys_qr[1])
26     for qb in sys_qr:
27         circuit.append(UGate(angle, angle, angle), [qb])
28     circuit.measure([2,3])
29     circuit.reset(2)
30     circuit.reset(3)
31
32     def after(self, circuit):
33         circuit.measure([0,1])

```

Conversely, the implementation of the RM ansatz is driven by the need for fast state reset and repeated measurement cycles. The parameters controlling the measurement frequency, circuit depth between measurements and the choice of single-qubit rotations allow fine-tuning of the reservoir dynamics for time-series prediction tasks:

```

1 class QRC_RM(Incremental):
2     def __init__(self, n_qubits, scale_input, memory=np.inf, backend=None,
3         coupling_map=None, layout=None, num_features=-1) -> None:
4         super().__init__(n_qubits, memory, backend, coupling_map, layout,
5             num_features)
6         self.n_qubits = n_qubits
7         self.scale_input = scale_input
8         self.coupling_map = coupling_map
9         self.layout = layout
10
11     def before(self, circuit):
12         circuit.h(circuit.qubits)
13
14     def during(self, circuit, timestep, reservoirnumber):
15         angle = self.scale_input * timestep[0]
16         # 0,1 are sys qubits, 2,3 are ancillas
17         circuit.rx(angle, 0)
18         circuit.cx(0, 1)
19         circuit.rx(angle, 0)
20         circuit.rz(angle, 1)
21         circuit.cx(0, 1)
22         circuit.cx(0, 2)
23         circuit.cx(1, 3)
24         circuit.measure([2, 3])
25         circuit.reset(2)
26         circuit.reset(3)
27
28     def after(self, circuit):
29         circuit.measure([0, 1])

```

Since the MERA and RM implementations produce different predictive behaviors, their performance metrics will be analyzed separately and compared in the "Results" chapter.

For more details on computational tools and development environments please refer to Appendix K.

For more details on the project’s repository structure, contributions and supporting resources, please refer to Appendix L and Appendix M.

2.3.3 Backends used for simulation

The term backend refers to the computational environment — either simulated or physical — used to execute the quantum circuit simulations. The backend determines how the algorithm is compiled, executed and measured; moreover, it significantly influences the performance of the model, especially in the presence of real-device noise and hardware constraints.

For this work, three types of backends were used in a progressive workflow:

- Qiskit AerSimulator (ideal, noise-free simulation)
- IQM Adonis (fake provider) (simulated hardware with realistic noise)
- IQM Spark quantum computer (execution on physical quantum hardware)

This sequence allows evaluation of the algorithm under increasingly realistic conditions: from ideal behavior, to noisy simulations, up to real quantum execution.

The first executions were performed using the AerSimulator, the standard high-performance simulator included in Qiskit Aer [46]. AerSimulator enables running quantum circuits locally on classical hardware (CPU and GPU), and supports different simulation methods (statevector, stabilizer formalism, matrix product states (MPS), etc.). It also allows the construction and injection of custom noise models, which makes it a fundamental tool for prototyping quantum algorithms, debugging, and validating their expected behavior prior to deployment on real quantum processors. The simulations were then extended using IQM Adonis, a fake provider backend. In Qiskit, a fake provider emulates the characteristics of a real quantum device, including qubit connectivity, gate fidelity, decoherence parameters, and measurement noise [47]. Although the device is not physically accessed, the backend reproduces realistic hardware conditions and transpiler constraints, allowing performance evaluation under noisy environments and testing the robustness of the algorithm. Such fake providers are widely used to benchmark the transpilation process and simulate hardware behavior when access to the actual device is limited.

Finally, simulations were executed on a real hardware backend: the IQM Spark quantum computer.

This step is essential to assess the viability of the model in real-world conditions and identify challenges such as decoherence, shot noise, and gate imperfections.

IQM Spark is a 5-qubit superconducting quantum processor developed for research and educational purposes and currently installed at Politecnico di Torino (for more details see Appendix H). Its architecture provides native access to realistic experimental

noise, limited connectivity patterns, and hardware timing constraints, enabling a direct comparison between theoretical simulations and physical execution [41]. So, the use of three progressively realistic backends — AerSimulator, IQM Adonis, and IQM Spark — enables a systematic evaluation of the proposed quantum models across different levels of experimental complexity. On the ideal simulator, the algorithm’s intrinsic performance can be measured without noise interference, establishing a theoretical performance baseline and allowing rapid testing of circuit design choices (e.g., ansatz depth, encoding strategies, measurement readout). The noise-aware fake provider introduces realistic quantum hardware imperfections, such as decoherence, gate error rates, and qubit connectivity constraints. This intermediate step is essential to understand how sensitive the model is to real-world noise, to test the robustness of the reservoir design, and to experiment with strategies such as circuit compression, noise-adaptive ansatz selection, and error-resilient training. Finally, execution on the real quantum device provides empirical evidence of the model’s viability under operational conditions, capturing effects that cannot be fully reproduced in simulation (e.g., calibration drift, cross-talk, environmental interference). By comparing results obtained across these backends, it becomes possible to quantify the performance degradation introduced by noise, identify bottlenecks in time-series prediction tasks, and assess whether quantum reservoir methods — such as MERA and RM approaches — can retain their advantage under realistic hardware constraints. This experimental workflow is therefore essential to bridge the gap between theoretical quantum machine learning models and practical, deployable QML solutions for real-world applications.

2.3.4 Models for fitting data

Machine learning models are computational algorithms capable of identifying patterns within data and generating predictions on previously unseen datasets. Unlike traditional rule-based systems, these models are not explicitly programmed with predefined rules but instead learn directly from data, allowing them to improve their performance as additional information becomes available.

Among the various machine learning techniques, regression models represent a class of supervised learning methods specifically designed to predict continuous numerical variables, such as stock prices, energy consumption, or temperature measurements. In the context of this work, regression models are particularly suitable because the task consists of forecasting the future values of a time series, namely the ATM balance over time, which naturally takes the form of a continuous numerical variable.

For this reason, regression-based approaches provide an appropriate framework for modeling the relationship between past observations of the time series and the corresponding future values. By learning these temporal relationships from historical data, the model can generate predictions that approximate the expected evolution of the variable under study.

The general objective of a regression model is to learn a function $f(x)$ that maps input features x to continuous outputs y , by minimizing the prediction error [48].

The classical formulation is:

$$y = f(x) + \epsilon \quad (2.2)$$

where ϵ is the prediction error (noise).

However, in a special category of regression models which are called linear models, the output is expressed as a linear combination of the input features:

$$y = w_0 + w_1x_1 + \dots + w_nx_n \quad (2.3)$$

In Machine Learning, regression encompasses several model variants that mainly differ in how they estimate the parameters w and how they control overfitting. Common approaches in this family include LinearRegression, Ridge, Lasso, ElasticNet, and SGDRegressor, each introducing different optimization strategies or regularization techniques to improve generalization and model stability.

- LinearRegression is the simplest and most classical regression model [49].

It estimates the parameters w by minimizing the sum of squared error (Residual Sum of Squares):

$$\min_w \sum (y - \hat{y})^2 \quad (2.4)$$

It is very fast, analytically solvable and it is very easy to interpret coefficients; moreover, no regularization is required, so it captures the full linear relationship.

However, it does not give protection against overfitting, its performance degrades with multicollinearity (correlated features) and it is sensitive to outliers.

So, it is recommended to use it when dataset is small, clean and features are not strongly correlated.

- Ridge Regression [50] adds an L2 regularization term to the LinearRegression loss function:

$$\min_w \sum (y - \hat{y})^2 + \alpha \sum w_j^2 \quad (2.5)$$

where the penalty term discourages large weight coefficients.

It reduces variance and improves generalization, handles multicollinearity well and keeps all features in the model; however, it does not perform feature selection (weights shrink, but rarely become zero).

Therefore, it is suggested to use it when there are many correlated features (and you want a stable linear model) and when the feature selection is not mandatory.

- Lasso Regression [51] adds an L1 penalty:

$$\min_w \sum (y - \hat{y})^2 + \alpha \sum |w_j| \quad (2.6)$$

where the L1 norm pushes many coefficients exactly to zero, resulting in automatic feature selection.

It performs sparse modeling, it is useful for high-dimensional data and reduces risk of overfitting significantly; notwithstanding, it can be unstable when features are highly correlated and it tends to select only one feature out of a group of correlated features.

In general, it enhances model interpretability and feature selection.

- ElasticNet [52] combines both L1 and L2 penalties:

$$\min_w \sum (y - \hat{y})^2 + \alpha \left(\lambda \sum |w_j| + (1 - \lambda) \sum w_j^2 \right) \quad (2.7)$$

where λ controls the trade-off between Lasso and Ridge behavior.

It balances feature selection (Lasso) and stability (Ridge), handles correlated features better than Lasso alone and it is useful when there are more features than samples.

Nevertheless, it presents two hyperparameters to tune (α , λ) and so it is slightly more complex.

- SGDRegressor [53] approaches regression differently from traditional analytical methods. Rather than computing the optimal parameters in a single closed-form solution, it learns the model iteratively using stochastic gradient descent. At each update step, it adjusts the weights based on a small batch (or even a single sample), which makes it highly flexible and adaptable.

A key benefit of SGDRegressor is that it can be paired with different loss functions and regularization strategies, making it suitable for a wide range of optimization settings. For example, it may minimize a squared loss or a Huber loss, and it can apply L1, L2, or ElasticNet penalties to control overfitting.

It is particularly effective when the dataset is too large for analytical solvers or when continuous training is required, such as in streaming applications or real-time systems.

The project analyzed these models and evaluated their performance using the absolute error (AE) metric for the MERA and RM analyses. The results are shown below (Figure 2.7 and Figure 2.8):

2.3.5 Post-processing methods construction

To interpret the outputs produced by the quantum reservoir and transform them into meaningful features for classical post-processing, three distinct evaluation strategies

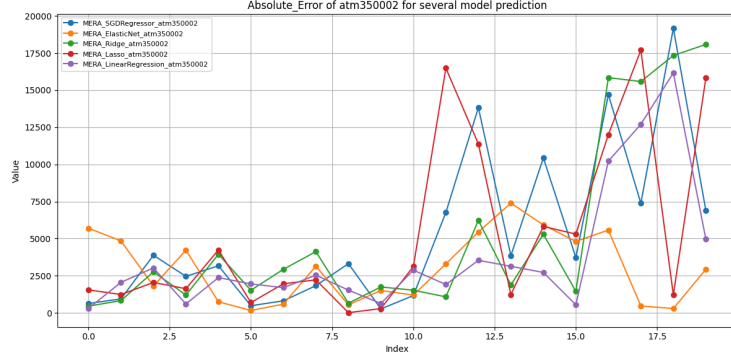


Figure 2.7: Model performance comparison in term of absolute error for MERA analysis.

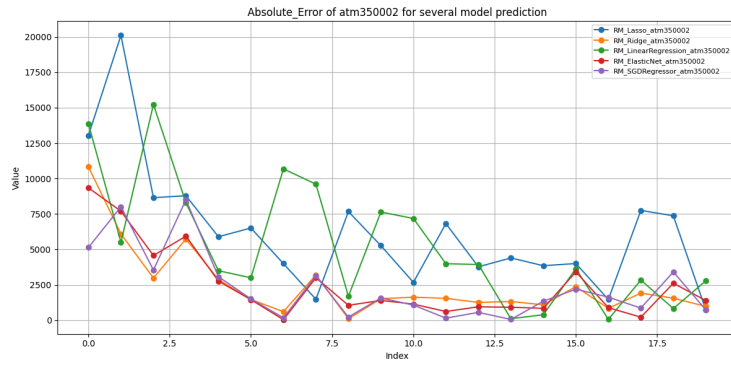


Figure 2.8: Model performance comparison in term of absolute error for RM analysis.

were implemented in Python and tested independently. Each method extracts different levels of statistical information from the reservoir’s measurement outcomes, and their performance was subsequently compared to identify the most effective representation.

Let $\mathbf{b} = \{b_1, b_2, \dots, b_N\}$, $b_i \in \{0, 1\}$ denote the bit string obtained from N repeated measurements of the quantum reservoir.

The first strategy analyzes the bit string generated by the quantum reservoir and computes the empirical probability of observing the outcome ‘0’, as well as the corresponding probability of observing ‘1’. Mathematically speaking:

$$P(0) = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(b_i = 0), \quad P(1) = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(b_i = 1), \quad (2.8)$$

where $\mathbb{I}(\cdot)$ is the indicator function.

This probabilistic representation provides a very compact, noise-robust, and informative feature description, capturing the aggregate behavior of the quantum system instead of relying on individual stochastic samples. The resulting probabilities are then supplied as input features to the classical learning model.

The second approach extends this idea by exploiting the presence of two ancilla qubits in the system. Instead of single-bit statistics, it evaluates the joint probabilities of

all possible bit pairs — ‘00’, ‘01’, ‘10’, and ‘11’.

Mathematically speaking, since the constructed system employs two ancilla qubits, each measurement produces a two-bit outcome: $\mathbf{c}_i \in \{00, 01, 10, 11\}$.

So, this method computes the joint probabilities of all possible bit pairs:

$$P(ab) = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(\mathbf{c}_i = ab), \quad ab \in \{00, 01, 10, 11\}. \quad (2.9)$$

This yields a four-dimensional feature vector that captures higher-order correlations between the ancilla qubits, providing a richer representation of the reservoir’s internal dynamics.

The third approach adopts a different perspective by computing the Pearson correlation coefficient from the same output bit string. The approach focuses on correlation structure rather than outcome frequencies. Given two binary sequences extracted from the bit string (e.g., corresponding to different qubits or time indices), X and Y , the Pearson correlation coefficient is computed as:

$$\rho_{X,Y} = \frac{\text{Cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}. \quad (2.10)$$

This metric quantifies linear dependencies in the measurement data and provides an alternative statistical descriptor of the quantum reservoir’s temporal and structural behavior.

Together, these three methods progressively increase the expressive power of the extracted features - from simple marginal statistics, to joint distributions, to correlation-based descriptors — allowing a systematic investigation of how different classical representations of quantum measurements affect downstream learning performance. Below, the graphs containing the performance in terms of absolute error (AE) using the three different methods for MERA (Figure 2.9) and RM (Figure 2.10) approaches.

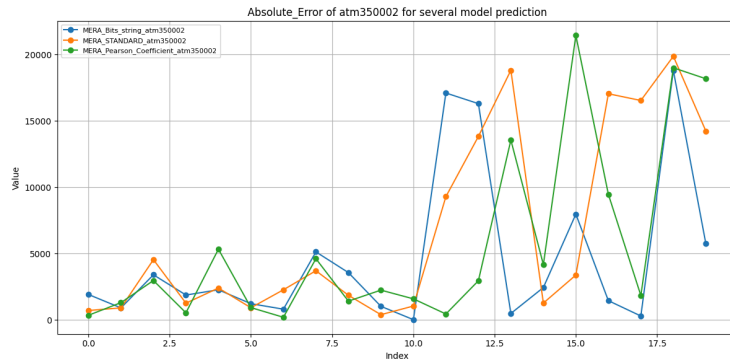


Figure 2.9: Comparison of different feature sets in terms of absolute error (AE) obtained after training and applying a linear regression model for MERA analysis

In the graphs the word "Standard" indicates the first method, the word "bit_string" indicates the second method and the word "Pearson_Coefficient" the third method.

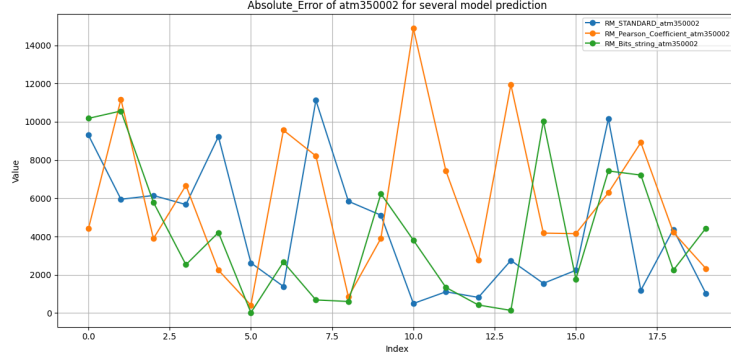


Figure 2.10: Features performance comparison in term of AE for RM analysis.

2.4 Metrics for the estimation of errors

To evaluate the performance of forecasting models, several metrics can be employed, each capturing different aspects of prediction accuracy. In this paragraph, we present examples derived from the simulation results for all the considered evaluation metrics. The Mean Absolute Error (MAE) measures the average magnitude of errors between predicted and true values, without considering their direction, providing a straightforward sense of overall prediction accuracy (Figure 2.11). Mathematically speaking:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (2.11)$$

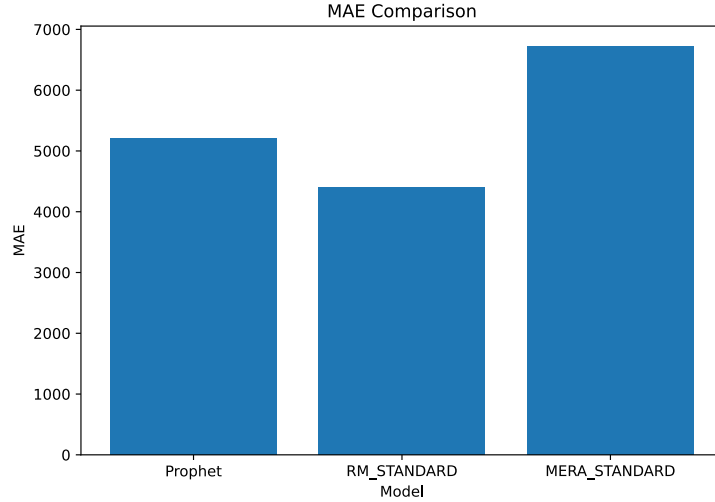


Figure 2.11: Comparison in term of MAE for PROPHET, RM and MERA cases.

The Mean Absolute Percentage Error (MAPE) expresses this error as a percentage of the true values, allowing for a scale-independent interpretation (Figure 2.12). In formula:

$$\text{MAPE} = \frac{100}{N} \sum_{i=1}^N \left| \frac{y_i - \hat{y}_i}{y_i} \right| \quad (2.12)$$

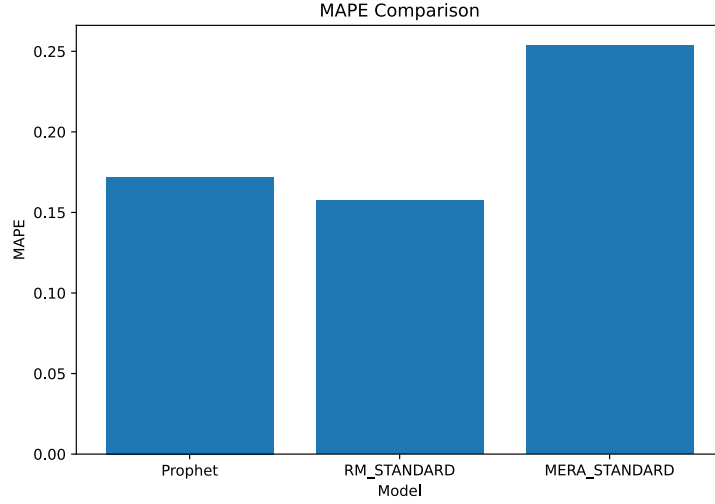


Figure 2.12: Comparison in term of MAPE for PROPHET, RM and MERA cases.

The Mean Squared Error (MSE) computes the average of the squared differences between predictions and observations, penalizing larger errors more heavily and emphasizing outliers (Figure 2.13). His equation is:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (2.13)$$

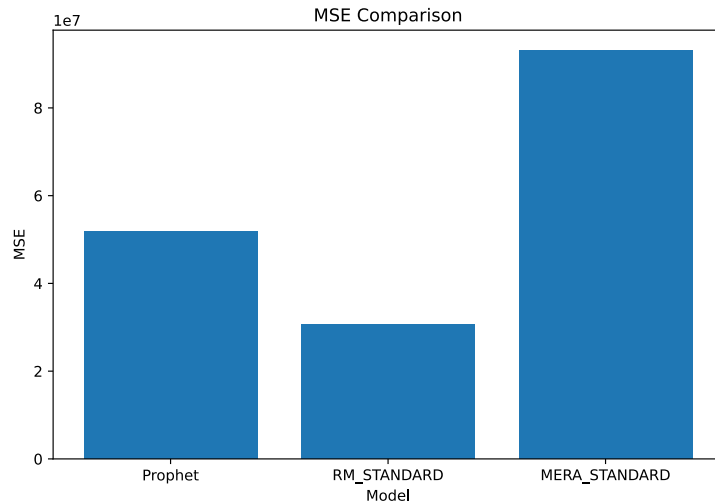


Figure 2.13: Comparison in term of MSE for PROPHET, RM and MERA cases.

Its square root, the Root Mean Squared Error (RMSE), brings the error back to the original scale of the data, making it easier to interpret (Figure 2.14). Its formulation is:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2} \quad (2.14)$$

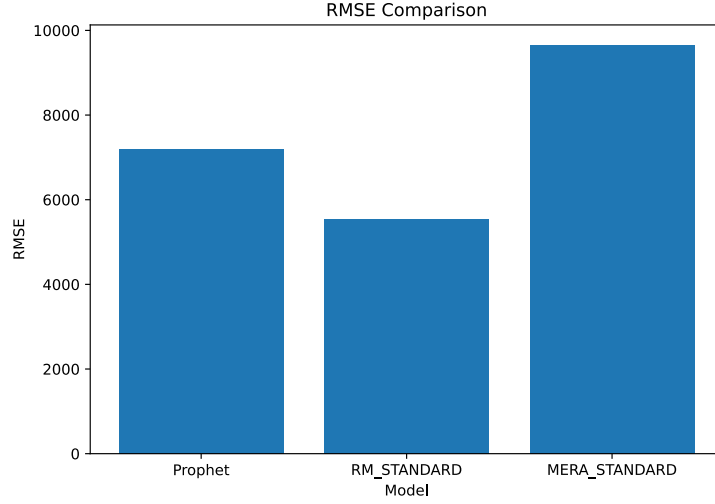


Figure 2.14: Comparison in term of RMSE for PROPHET, RM and MERA cases.

The Median Absolute Error (MedAE) represents the median of all absolute differences, offering a robust measure that is less sensitive to extreme outliers (Figure 2.15). Mathematically speaking:

$$\text{MedAE} = \text{median}(|y_1 - \hat{y}_1|, |y_2 - \hat{y}_2|, \dots, |y_N - \hat{y}_N|) \quad (2.15)$$

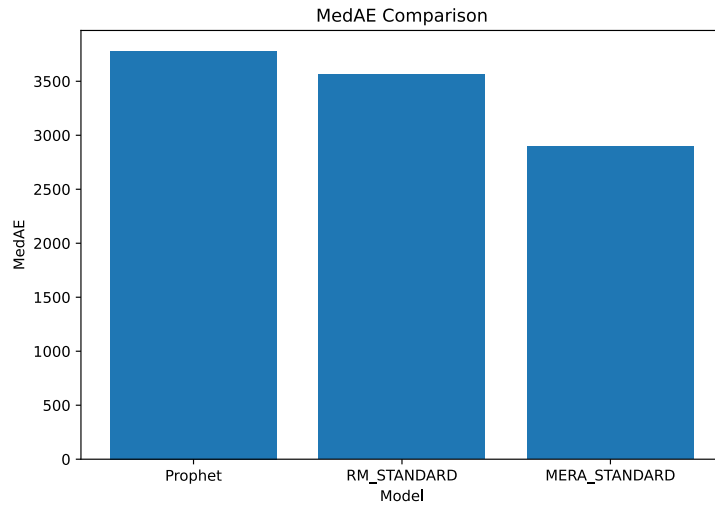


Figure 2.15: Comparison in term of MedAE for PROPHET, RM and MERA cases.

The Normalized Mean Squared Error (NMSE) scales the MSE by the variance of the true values, providing a dimensionless measure that facilitates comparisons across different datasets (Figure 2.16).

In the mathematical equation:

$$\text{NMSE} = \frac{\text{MSE}}{\sigma_y^2} = \frac{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2}{\frac{1}{N} \sum_{i=1}^N (y_i - \bar{y})^2} \quad (2.16)$$

where $\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i$ and σ_y^2 is the variance of the true values.

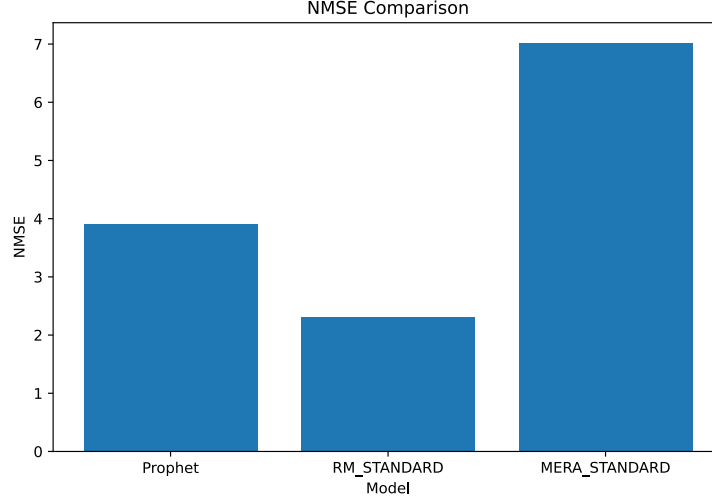


Figure 2.16: Comparison in term of NMSE for PROPHET, RM and MERA cases.

Finally, Dynamic Time Warping (DTW) evaluates the similarity between two sequences that may vary in time or speed, making it particularly useful for time-series data where shifts or temporal misalignments occur between predicted and actual values (Figure 2.17). In other terms, DTW measures the similarity between two sequences $y = (y_1, \dots, y_N)$ and $\hat{y} = (\hat{y}_1, \dots, \hat{y}_M)$ by finding an optimal warping path. Let $D(i,j)$ be the accumulated cost:

$$D(i, j) = |y_i - \hat{y}_j| + \min(D(i-1, j), D(i, j-1), D(i-1, j-1)) \quad (2.17)$$

with $D(0, 0) = 0$, then the DTW distance is:

$$\text{DTW}(y, \hat{y}) = D(N, M) \quad (2.18)$$

To conclude, the summary table below (Table 4.1) describes all the metrics previously described, using the mathematical formalism mentioned above. Consider that in the table y_i be the true value and $\hat{y}_i$ the predicted value for $i = 1, \dots, N$.

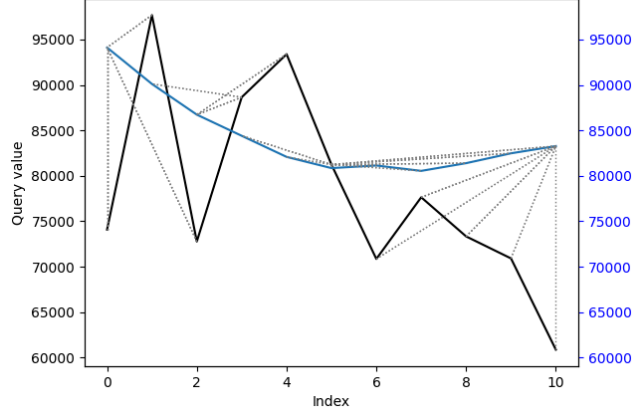


Figure 2.17: Dynamic Time Warping (DTW) Analysis for ATM 350002.

Table 2.1: Summary of Forecast Evaluation Metrics with Descriptions

Metric	Formula	Description
Mean Absolute Error (MAE)	$\frac{1}{N} \sum_{i=1}^N y_i - \hat{y}_i $	Average magnitude of errors between predictions and true values, ignoring direction.
Mean Absolute Percentage Error (MAPE)	$\frac{100}{N} \sum_{i=1}^N \left \frac{y_i - \hat{y}_i}{y_i} \right $	Average error expressed as a percentage of the true values; scale-independent.
Mean Squared Error (MSE)	$\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$	Average of squared errors; penalizes larger deviations more strongly.
Root Mean Squared Error (RMSE)	$\sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2}$	Square root of MSE; in the same units as the original data for easier interpretation.
Median Absolute Error (MedAE)	$\text{median}(y_1 - \hat{y}_1 , \dots, y_N - \hat{y}_N)$	Median of absolute errors; robust to outliers.
Normalized Mean Squared Error (NMSE)	$\frac{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2}{\frac{1}{N} \sum_{i=1}^N (y_i - \bar{y})^2}$, $\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i$	MSE normalized by the variance of true values; allows comparison across datasets.
Dynamic Time Warping (DTW)	$DTW(y, \hat{y}) = D(N, M)$	Measures similarity between two sequences that may vary in time or speed.

Chapter 3

Results

This chapter presents the final results obtained from the numerical simulations, aiming to assess the performance of the proposed approaches under different configurations and execution backends. In particular, the results are organized as follows: the memory behavior of the Incremental reservoir is first examined to determine the optimal memory parameter; this is followed by a performance comparison between the Static and Incremental reservoir configurations; finally, the effect of noise is discussed by contrasting noisy and noiseless implementations. For the sake of completeness, some results have already been introduced in the previous paragraph, providing preliminary insights that will be further elaborated and contextualized in the subsequent discussion.

Following these considerations, we present the complete time-series prediction simulations obtained using the best-performing configuration for each reservoir. Finally, the results achieved on the IQM Spark quantum processor are introduced and discussed, and their performance is analyzed in comparison with the other simulation backend.

3.1 Memory capacity effects in Incremental reservoirs

The behavior of the Incremental reservoir - for both MERA and RM frameworks - is strongly influenced by the selected memory size, which determines the temporal extent of past information retained during the forecasting process.

For smaller memory sizes (e.g., `memory_size = 2` or `5`), the reservoir retains only a limited history of past states, resulting in a relatively narrow temporal context to accurately model long-range dependencies. Consequently, the predictive performance in these configurations is typically lower, as the model primarily captures short-term dynamics. As the memory size increases (e.g., `memory_size = 7`, `10`, or `12`), the reservoir gains access to a broader temporal window, enhancing its ability to represent medium and long-term dependencies. This improvement generally leads to better forecasting accuracy, as the model can more effectively integrate historical information into its predictions. For very large memory sizes (e.g., `memory_size = 15`, `18`, `21`, or `24`), the reservoir has access to an extremely wide temporal window; however, performance gains may begin to saturate or even slightly degrade. In these

cases, the inclusion of excessive historical information can introduce redundancy and noise, increasing the complexity of the reservoir state and potentially reducing its responsiveness to recent inputs.

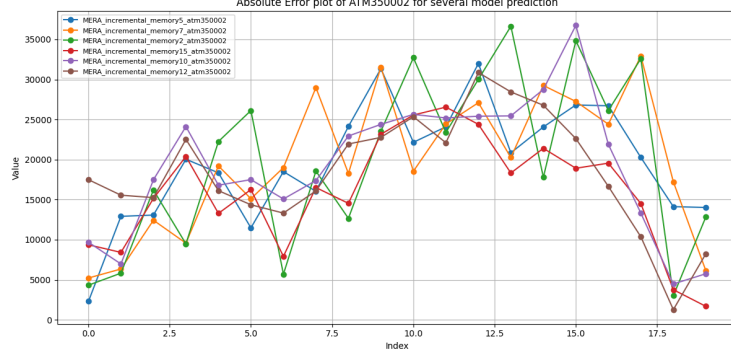


Figure 3.1: Memory-related behavior specifically associated with the Incremental reservoir in terms of absolute error for MERA approach.

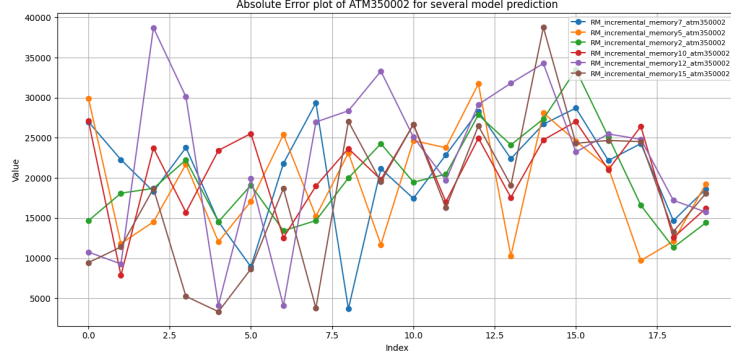


Figure 3.2: Memory-related behavior specifically associated with the Incremental reservoir in terms of absolute error for RM approach.

Overall, different behaviors can be observed depending on whether the MERA or the RM reservoir is employed. The RM approach relies strongly on explicit memory capacity to enhance performance, benefiting consistently from larger memory sizes, whereas MERA leverages its internal hierarchical structure to compress temporal information efficiently, making it less dependent on large memory windows. Consequently, MERA achieves competitive performance with smaller memory sizes (Figure 3.1), while RM requires larger memory allocations to reach comparable accuracy (Figure 3.2).

Above, the final comparison plot (Figure 3.3) summarizes the effect of the memory-size dimension on the predictive performance of the different approaches considered in this work, namely MERA, RM, and Prophet, evaluated in terms of Mean Absolute Error (MAE).

For MERA and RM architectures, the plot illustrates how the MAE varies as a function of the assigned memory size, highlighting the direct role played by temporal memory in these reservoir-based models. In contrast, the Prophet method is included for the sake of completeness and serves as a baseline reference: since Prophet does not

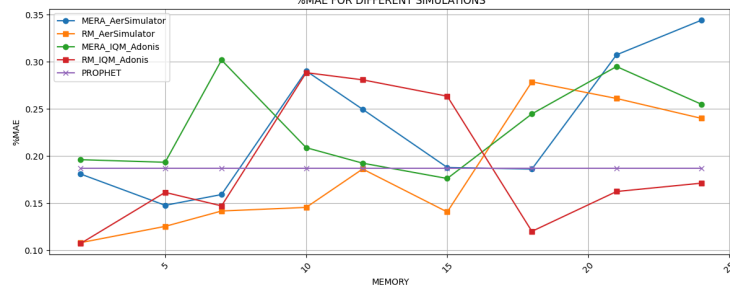


Figure 3.3: Overall comparison memory-related behavior for MERA, RM, Prophet approaches in terms of MAE.

explicitly rely on a memory mechanism, its MAE remains constant across all memory-size configurations. This representation allows for a direct visual comparison between memory-aware and memory-agnostic models, emphasizing how the availability and tuning of memory capacity influence prediction accuracy. Overall, the plot provides a comprehensive perspective on the trade-off between memory size and performance, while contextualizing the behavior of reservoir-based approaches with respect to a classical time-series forecasting method.

3.2 Performance of Static and Incremental reservoirs

The comparison between the Static and Incremental reservoir approaches highlights distinct behaviors depending on whether the MERA (Figure 3.4) or the RM (Figure 3.5) framework is adopted.

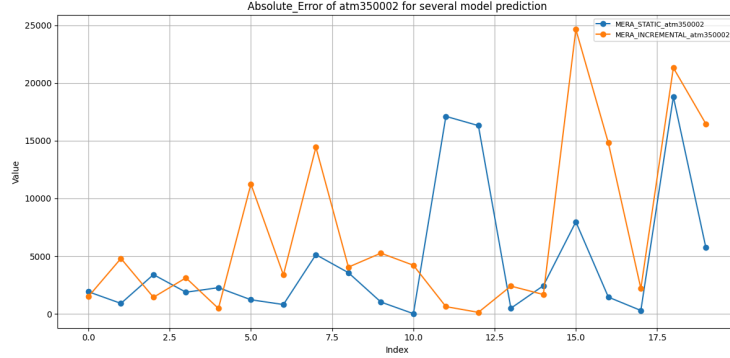


Figure 3.4: Performance comparison between the Static and Incremental reservoir approaches in terms of absolute error for MERA approach.

The observed superior performance of the MERA Static approach compared to the MERA Incremental configuration can be primarily attributed to the stability and global optimization of its representation. In the Static setting, the MERA circuit is trained to encode the system dynamics within a fixed and consistent structure, allowing it to capture long-range correlations and hierarchical features more effectively. This global optimization enables the model to maintain coherence across different scales, reducing the accumulation of errors during inference.

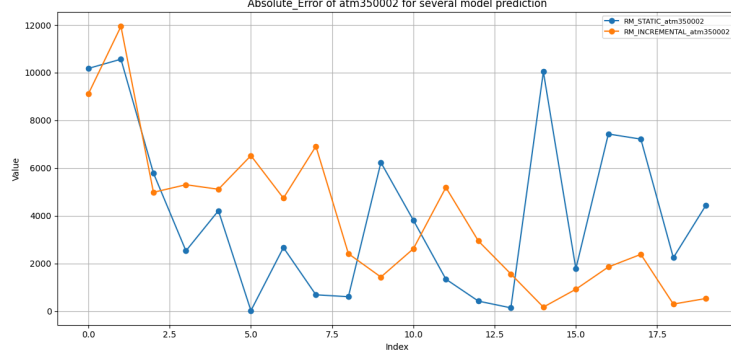


Figure 3.5: Performance comparison between the Static and Incremental reservoir in terms of absolute error for RM approach.

In contrast, the Incremental approach, which updates the reservoir progressively, is more susceptible to error propagation and temporal drift, as small inaccuracies introduced at each update step may compound over time. Moreover, the frequent adaptation of the reservoir in the Incremental scheme can disrupt the hierarchical balance inherent to the MERA architecture, ultimately limiting its ability to exploit multiscale correlations. As a result, the MERA Static configuration achieves greater robustness and predictive accuracy, particularly in forecasting scenarios where stable long-term representations of the system dynamics are crucial.

A different behavior is observed for the RM approach. In this case, the RM Incremental configuration outperforms the RM Static one, which can be attributed to its ability to dynamically adapt the reservoir to the evolving dynamics of the input data. In the Incremental setting, the reservoir is progressively updated as new data become available, allowing the model to continuously refine its internal representation and better track temporal variations in the underlying system. This adaptive behavior enhances the reservoir’s short-term memory and responsiveness to recent inputs, which are critical factors in time-series prediction tasks.

Conversely, the Static RM approach relies on a fixed reservoir configuration that may fail to adequately capture non-stationary patterns or temporal shifts in the data. As a result, the lack of adaptability in the Static configuration limits its predictive accuracy, particularly in scenarios where the system dynamics evolve over time. Overall, these findings suggest that the Incremental update mechanism enhances the adaptability of the RM architecture, allowing it to better capture evolving temporal dynamics and thereby improve forecasting performance.

3.3 Influence of noise on reservoir performance

From the analysis, it emerges that the model incorporating noise consistently outperforms the noiseless configuration. Across the considered evaluation metrics, the presence of noise leads to improved predictive accuracy, suggesting that the noisy model is better able to capture the underlying dynamics of the system. The comparison plots explained in terms of absolute error between the noise and noiseless models

for MERA (Figure 3.6) and RM (Figure 3.7) approaches, are reported below.

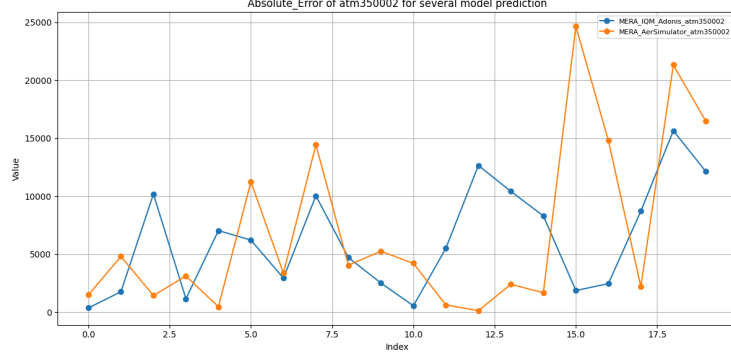


Figure 3.6: Comparison between AerSimulator and IQM_FakeAdonis in terms of absolute error for MERA approach (Static reservoir).

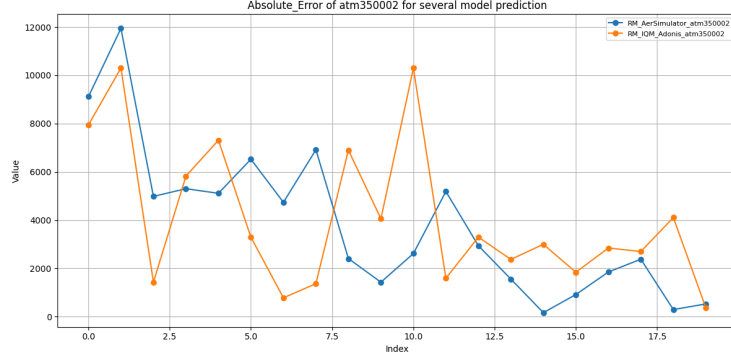


Figure 3.7: Comparison between AerSimulator and IQM_FakeAdonis in terms of absolute error for RM approach (Incremental reservoir).

It can be noticed that, in the MERA-based implementation, the presence of noise leads to a more pronounced performance improvement compared to the RM approach. This behavior can be attributed to the hierarchical structure of the MERA circuit, which inherently distributes correlations across multiple scales and may therefore benefit from noise acting as a regularizing mechanism. In this context, noise can mitigate over-sensitivity to local fluctuations, allowing the MERA model to capture more stable and global features of the underlying dynamics. Conversely, the RM approach, which relies on a flatter and more localized structure, appears less capable of exploiting the beneficial effects of noise, resulting in comparatively smaller performance gains. Overall, these observations suggest that the MERA architecture exhibits greater robustness to noise, making it more effective in realistic, noisy simulation environments.

3.4 Performance on IQM Spark machine

When also considering the performance of the time-series predictions evaluated on real quantum hardware (i.e., IQM Spark), additional observations can be made. For example, in this setting, the Repeated Measurements (RM) approach consistently

outperforms the MERA-based implementation (Figure 3.8). This behavior can be attributed to the stronger interaction between the RM architecture and the intrinsic properties of the quantum device. In particular, the RM framework directly exploits quantum measurement back-action, allowing the reservoir state to be dynamically reshaped at each iteration through the outcomes of real measurements. On actual quantum hardware, these effects become more pronounced than in simulated environments, leading to enhanced memory retention and richer state dynamics. In contrast, the MERA architecture relies on a more structured and static entanglement pattern, which may be more sensitive to hardware imperfections and less capable of leveraging measurement-induced dynamics. As a result, RM benefits more effectively from execution on IQM Spark, yielding superior time-series prediction performance compared to MERA.

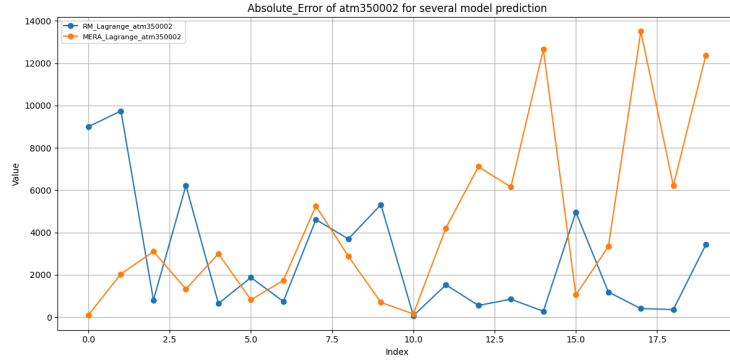


Figure 3.8: Absolute Error comparison between the performance of RM and MERA approaches after simulation on IQM Spark machine (Incremental reservoir).

Furthermore, the Repeated Measurements (RM) approach shows superior performance when executed on IQM Spark compared to both the AerSimulator and the Fake Adonis simulated backends (Figure 3.9). While these simulators aim to reproduce the expected quantum dynamics — either in ideal noise-free conditions or by incorporating approximate noise models — they cannot fully replicate the complex, hardware-specific effects present in an actual quantum device. On real hardware, the RM framework can more effectively exploit genuine quantum phenomena, device-specific noise correlations, and stochastic fluctuations, which collectively enhance the richness of the reservoir dynamics. As a result, the execution on the IQM Spark leads to improved memory retention and more accurate time-series predictions, outperforming both idealized and noise-simulated backends.

For a seek of completeness of the performance analysis, a final comparison plot is presented, summarizing the absolute error achieved by all considered backends. This plot provides a unified view of the predictive accuracy of the classical and quantum-based approaches, including Prophet, AerSimulator, Fake Adonis, and the real quantum hardware (Figure 3.10). By directly comparing the absolute error across these execution environments (for more details see Appendix J), the figure highlights the relative strengths and limitations of each backend and offers a concise overview of how simulation fidelity, noise modeling, and real hardware effects influence time-series

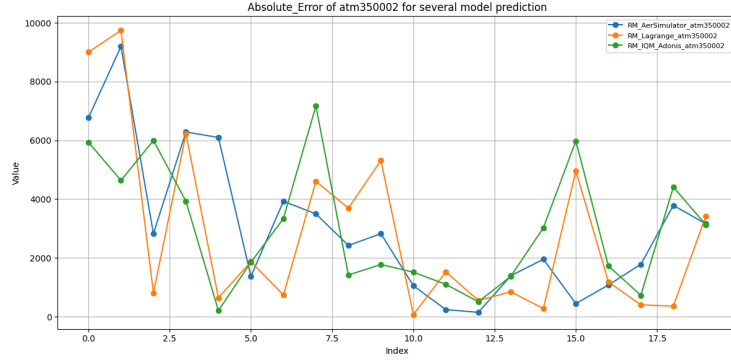


Figure 3.9: Absolute Error comparison between the AerSimulator, Adonis and Lagrange (IQM Spark) backends.

prediction performance.

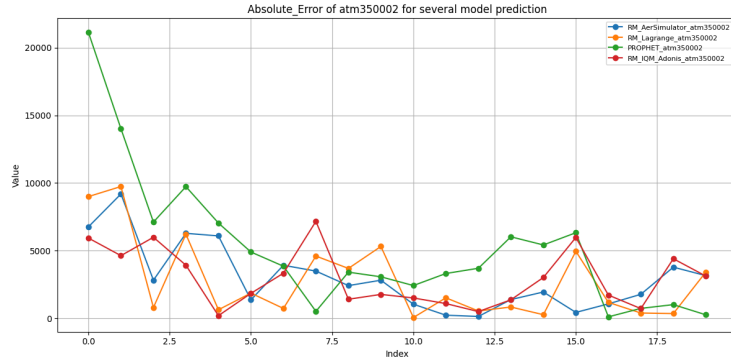


Figure 3.10: Absolute Error comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends; view the Appendix section for others results in term of error comparison.

3.5 Complete time-series prediction analysis

Finally, the time-series prediction plots obtained from the complete set of simulations are presented. These figures illustrate the forecasting behavior of the considered models over the entire prediction horizon, allowing a direct visual comparison between the predicted and the true values. The simulations shown correspond to the best-performing configurations previously identified for each reservoir architecture and backend. Both noise-free and noisy scenarios are included in order to provide a comprehensive overview of the obtained predictions.

At this stage, the purpose of these plots is primarily illustrative, providing a visual representation of the forecasting results. A detailed interpretation and comparative analysis of the observed behaviors will be carried out later in the "Discussion" chapter. The figures are presented following a progressive and structured order. The sequence begins with the results obtained using the Prophet model (Figure 3.11), which serves as the classical baseline. Next, the predictions produced by the MERA-based simulations on the AerSimulator backend (Figure 3.12) are reported, representing an idealized quantum scenario without noise effects. Subsequently, the predictions obtained with the Fake Adonis backend (Figure 3.13) are shown, introducing a realistic noise model while maintaining the advantages of a simulated environment. Finally, the results obtained on the IQM Spark quantum computer (Figure 3.14) are presented, allowing the behavior of the model to be observed under real quantum hardware conditions.

This presentation order provides a clear visual overview of the forecasting results across the different computational settings, allowing for an initial qualitative assessment of the models' behavior.

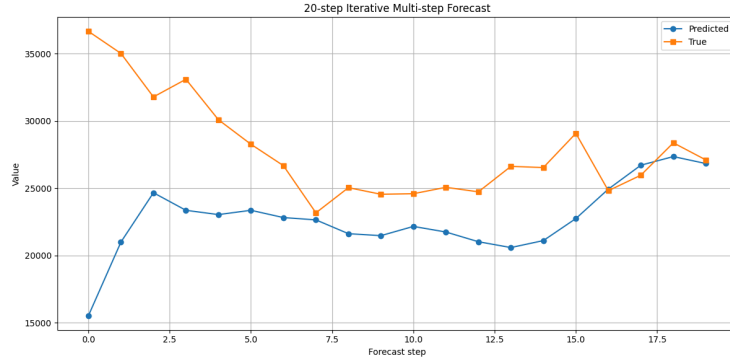


Figure 3.11: Prophet final time-series prediction.

The same presentation structure adopted for the MERA-based results is also followed for the Repeated Measurements (RM) simulations. In particular, the RM results will be examined following the same progression — from the classical baseline, through idealized noise-free simulations, to noisy and real-hardware executions — allowing for a coherent comparison between the different reservoir architectures and computational paradigms (Figures 3.15 - 3.16 - 3.17).

Overall, the predictions produced by the quantum approaches follow the behavior of

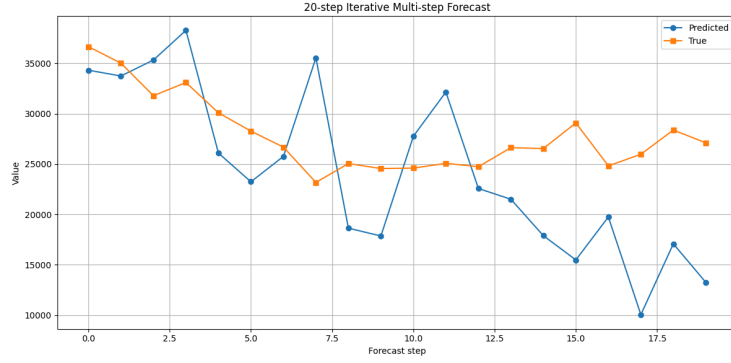


Figure 3.12: MERA_Incremental_AerSimulator final time-series prediction.

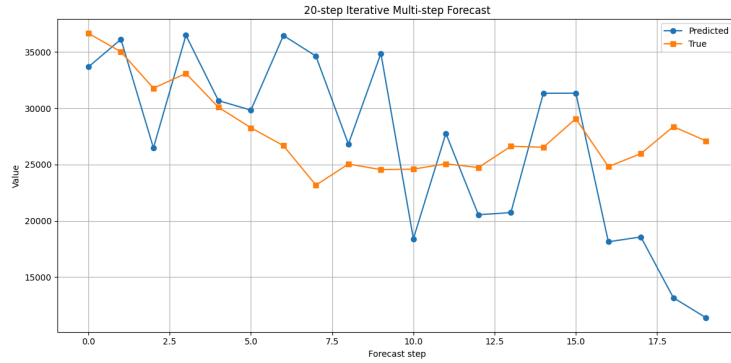


Figure 3.13: MERA_Incremental_Fake_Adonis final time-series prediction.

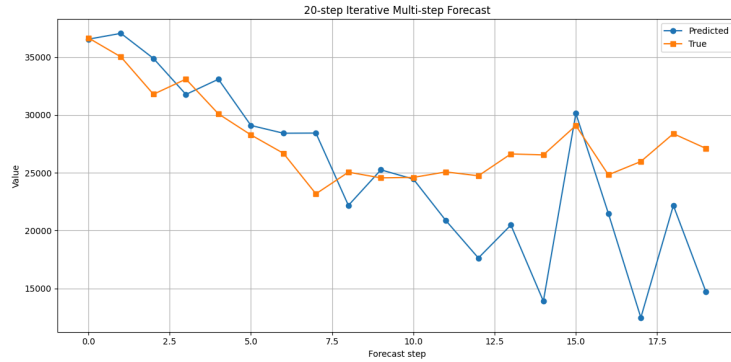


Figure 3.14: MERA_IQM_SPARK final time-series prediction.

the real time series more closely than the classical baseline provided by the Prophet model. Moreover, the results obtained on IQM Spark show a slightly better agreement with the real values compared to those generated by the simulated backends. Finally, a further distinction emerges between the two reservoir implementations: the MERA approach provides a more accurate approximation during the first ten prediction steps, whereas the Repeated Measurements (RM) approach performs better in the final part of the prediction horizon.

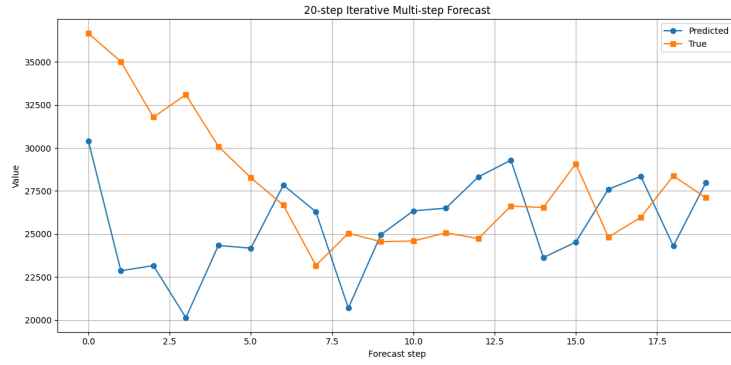


Figure 3.15: RM_Incremental_AerSimulator final time-series prediction.

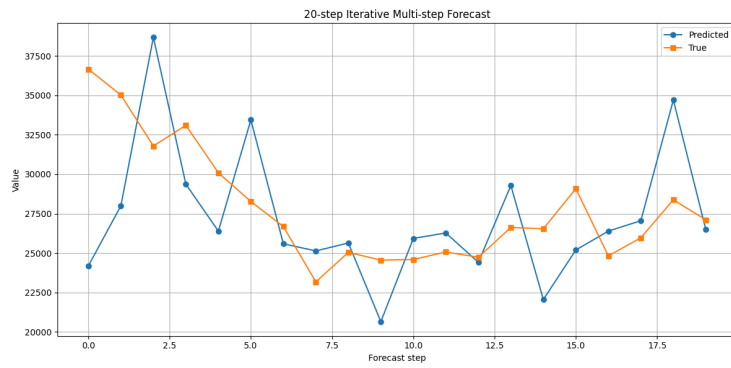


Figure 3.16: RM_Incremental_Fake_Adonis final time-series prediction.

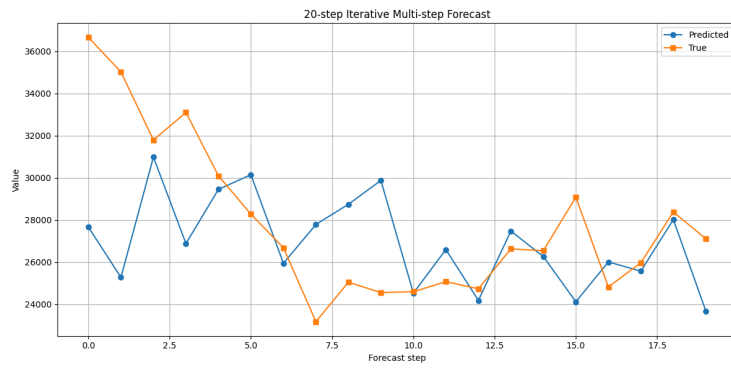


Figure 3.17: RM_IQM_SPARK final time-series prediction.

Chapter 4

Discussion

This section provides a comprehensive discussion of the results obtained from the time-series prediction simulations carried out throughout this study. The analysis encompasses both classical and quantum-inspired approaches, including Prophet as a classical baseline, and the Repeated Measurements (RM) and Multi-scale Entanglement Renormalization Ansatz (MERA) models implemented under different computational settings. In particular, the results are examined by progressively moving from ideal, noise-free simulations to noisy backends and, finally, to executions on real quantum hardware. This structured comparison enables an in-depth evaluation of the impact of noise, memory capacity, and reservoir architecture on predictive performance, while also highlighting the differences between classical and quantum paradigms. The discussion aims to synthesize these findings, identify consistent performance trends, and provide physical and algorithmic insights into the observed behaviors across the various models and execution environments. Remember that the results, presented and discussed in the previous paragraph, are obtained by considering the optimal set of parameters and configuration settings identified for each simulation scenario. In fact, for each model and backend, the hyperparameters — such as reservoir size, memory depth, and noise configuration — were carefully selected based on their best-performing configurations, as determined through preliminary analyses. This approach ensures that the reported results reflect the maximum achievable performance for each method under the corresponding computational conditions, enabling a fair and meaningful comparison across different architectures and execution environments.

4.1 Classical and Quantum time-series predictions

A first and immediately evident observation from the plots is that the quantum-inspired approaches consistently outperform the classical Prophet model, as evidenced by their superior time-series prediction accuracy. This performance gap highlights the advantage of exploiting quantum-inspired dynamics and high-dimensional state spaces over purely classical approaches, especially in the context of complex temporal dependencies. However, it is important to interpret this result with caution: the

relatively small size of the dataset may partially contribute to the observed performance gap. Classical models such as Prophet rely on trend and seasonal estimation, which can be limited when only a few data points are available, whereas quantum reservoirs leverage high-dimensional state spaces and complex interference patterns to encode temporal dependencies even from limited observations. Consequently, while the quantum approaches demonstrate superior predictive accuracy in the present simulations, the difference may diminish for larger datasets where classical methods can more effectively exploit available information.

4.2 Performance of MERA and RM ansatz

Another immediately evident observation emerging from the plots is that the predictive performance of the reservoir-based models improves in the presence of noise. In particular, both RM and MERA reservoirs exhibit lower prediction errors when noisy simulations are considered, suggesting that noise acts as a regularizing mechanism that enhances the expressive power and generalization capability of the reservoirs. Focusing on the MERA-based approach, the results also show that the Static reservoir configuration consistently outperforms the Incremental one. This behavior can be attributed to the hierarchical and multiscale structure of the MERA ansatz, which is inherently well suited to capture temporal correlations within a fixed and stable representation. In this context, maintaining a Static reservoir allows the model to preserve coherent multiscale features across time, leading to more accurate time-series predictions. Conversely, the Incremental update of the reservoir tends to disrupt this structured representation, limiting its ability to exploit the full expressive capacity of the MERA architecture.

In contrast, an opposite trend is observed for the Repeated Measurements (RM) approach, where the Incremental reservoir configuration yields better performance than the Static one. In this case, the Incremental strategy enhances the effective memory of the reservoir by progressively incorporating information from previous predictions, thereby improving the model’s ability to capture temporal dependencies. As a result, the Incremental RM reservoir exhibits superior predictive accuracy compared to its Static counterpart.

The contrasting behaviors observed between the MERA and RM reservoir configurations can be further interpreted by considering the interplay between memory capacity, noise sensitivity, and quantum measurement back-action. In both Incremental approaches, the best predictive performance is achieved for moderate memory sizes (e.g., 5, 7 or 10) rather than for very large ones. This effect can be attributed to the increasing number of parameters that must be processed as the memory size grows, which can lead to over parameterization and reduced generalization capability in the time-series prediction task. Consequently, excessively large memory depths introduce additional complexity without providing proportional informational gain, ultimately degrading performance.

Within the MERA Incremental framework, the reservoir maintains a multiscale

representation that captures temporal correlations over a structured memory horizon. In this case, the incremental update mechanism only increases sensitivity to noise and parameter fluctuations, which can disrupt the MERA approach’s consistent hierarchical structure. Therefore, this approach is most effective when using moderate memory sizes.

In the RM approach, the Incremental reservoir configuration is more effective due to its enhanced capacity to exploit quantum measurement back-action. Each repeated measurement and subsequent state update injects new information into the reservoir dynamics, effectively extending the usable memory of the system. When combined with a controlled level of noise, this mechanism promotes richer state-space exploration and improved temporal feature extraction. However, even in this case, overly large memory sizes increase noise accumulation and parameter complexity, ultimately offsetting the benefits of incremental updates.

Overall, these results highlight that optimal performance arises from a balance between memory capacity, noise-induced regularization, and the constructive role of quantum measurement back-action, rather than from maximizing memory depth or reservoir complexity. The table below summarizes all the most important aspects to remember for evaluating the performance of the two framework.

Table 4.1: Summary of Performance Comparison between MERA and RM

Comparison Terms	MERA	RM
Presence of the noise	Work better	Work better
Performance compared to Prophet	Work better	Work better
Static reservoir configuration	Work better	Work worse
Incremental reservoir configuration	Work worse	Work better
Increasing Memory Capacity (before saturation)	Work worse	Work better

4.3 Behavior and analysis of the multi-step forecasting

In general, the results show that the first ten predicted values exhibit higher accuracy than the last ten. This behavior is expected in an iterative multi-step forecasting setting, where predictions are generated step-by-step and each new forecast relies on previously predicted values. As the prediction horizon increases, small errors introduced at early steps progressively accumulate and propagate, leading to a degradation in accuracy for longer-term predictions. In contrast, the last ten predictions correspond to a more challenging scenario in which the model is required to forecast a longer sequence of future values with reduced reliance on ground-truth information, thereby amplifying the impact of error accumulation.

When comparing the RM and MERA approaches across the prediction horizon, a complementary behavior emerges. In the short-term regime (first ten predictions), the MERA-based models outperform the RM approach. This can be attributed to the hierarchical and multiscale structure of the MERA reservoir, which enables an efficient encoding of local temporal correlations and supports accurate near-term forecasts. However, in the long-term regime (last ten predictions), the RM approach demonstrates superior performance. This advantage arises from the incremental nature of the RM reservoir, which enhances memory retention through repeated measurements and state updates.

The associated quantum measurement back-action effectively reinforces long-range temporal dependencies, allowing the RM model to better preserve and propagate information over extended prediction horizons.

Overall, these results highlight a trade-off between short-term precision and long-term stability: MERA reservoirs excel at capturing immediate temporal structure, while RM reservoirs are more effective in sustaining predictive accuracy over longer horizons.

4.4 Real quantum hardware and simulator performance

It is also important to note that the results obtained from simulations executed on the real quantum hardware, specifically the IQM Spark processor, consistently demonstrate improved predictive performance compared to those obtained on the Fake Adonis backend. While Fake Adonis provides a realistic noise model and simulates the expected behavior of the quantum system, it cannot fully capture all aspects of the actual hardware dynamics. Executing the algorithm on the real quantum processor allows the reservoir to exploit genuine quantum effects, including physical coherence and measurement-induced correlations, which can enhance the representation of temporal dependencies in the time-series data. As a result, the IQM Spark simulations achieve lower prediction errors and demonstrate that the use of actual quantum hardware can provide tangible benefits over even highly realistic simulator-based approximations.

Chapter 5

Conclusion and outlook

This chapter concludes the work by summarizing the main findings and insights obtained throughout the study. The performance of classical and quantum-inspired reservoir computing approaches has been systematically analyzed across different architectures, noise configurations and execution backends, including both simulated environments and real quantum hardware. The conclusions highlight the key trends observed in the results, discuss their implications for time-series prediction and reflect on the strengths and limitations of the proposed methods. Finally, this chapter outlines the broader significance of the work and provides a perspective on potential directions for future research in quantum-enhanced reservoir computing.

5.1 Overview of the analysis and key findings

This work has presented a comprehensive study on time-series prediction using classical and quantum-inspired reservoir computing approaches. The analysis began with the introduction of the theoretical foundations of reservoir computing, followed by the description of the considered architectures, namely the Repeated Measurements (RM) and the Multi-scale Entanglement Renormalization Ansatz (MERA). Both Static and Incremental reservoir configurations were investigated in order to assess their impact on predictive performance and memory capacity.

Subsequently, the experimental framework was detailed, including the datasets, evaluation metrics, and forecasting strategies adopted for the simulations. A systematic comparison was then performed between noiseless and noisy configurations, highlighting the constructive role of noise in enhancing reservoir dynamics and improving generalization capabilities. The performance differences between Static and Incremental reservoirs were analyzed in depth, with particular attention to memory-related effects and their influence on short-term and long-term prediction accuracy.

The study further explored the behavior of the models under different execution environments, starting from classical baselines using Prophet, progressing through ideal quantum simulations with AerSimulator, noisy simulations with the Fake Adonis backend and, finally, real quantum hardware executions on the IQM Spark processor. This progression enabled a clear comparison between classical and quantum paradigms,

as well as between simulated and real quantum backends.

Special emphasis was placed on understanding how memory size, noise sensitivity, and quantum measurement back-action affect the predictive performance of RM and MERA approaches. The analysis revealed complementary behaviors between the two architectures, with MERA excelling in short-term predictions and RM demonstrating superior performance over longer horizons, particularly when executed on real quantum hardware.

Finally, the best-performing configurations for each reservoir and backend were identified and used to produce complete time-series prediction simulations. These results were synthesized through final comparison plots, providing a consolidated view of the performance across all methods and highlighting the potential advantages and current limitations of quantum-enhanced reservoir computing for time-series forecasting.

5.2 Improvements and future works

Several directions can be explored to further improve and extend the results presented in this work. First, the predictive performance could be enhanced by integrating alternative learning models for the readout layer, such as nonlinear regressors or deep learning-based approaches, which may better exploit the rich dynamics generated by the reservoir. In fact, in standard reservoir computing, the reservoir itself generates a high-dimensional dynamic representation of the input data, while the readout layer — typically a simple linear model — is responsible for mapping these rich dynamics to the predicted output. By integrating alternative learning models, such as nonlinear regressors or deep learning architectures, the readout layer can more effectively capture complex relationships embedded in the reservoir states. This is particularly beneficial when the reservoir dynamics are highly expressive or nonlinear, as a linear readout may not fully exploit the available information. While all of the above is applicable, it is partially in contradiction with the core principle of reservoir computing, which relies on a simple final predictor.

The suggestion is well-founded because it leverages the separation of responsibilities in reservoir computing: the reservoir encodes temporal features, while the readout interprets them. Using a more sophisticated readout can enhance predictive accuracy without significantly increasing the complexity of the reservoir itself. However, it is important to balance model complexity and overfitting, especially for small datasets, as more powerful readout models can also be more sensitive to noise and limited data.

In addition, the quantum ansatz structures — particularly for the MERA-based implementation — could be further optimized or redesigned to improve expressivity while reducing circuit depth and parameter complexity. The importance of optimizing or redesigning the quantum ansatz is so necessary to achieve two complementary goals: increased expressivity and reduced resource requirements. Expressivity refers to the ansatz’s ability to generate a diverse and informative set of quantum states

that can effectively encode temporal correlations in the input data. At the same time, minimizing circuit depth and parameter complexity is crucial for practical feasibility, as deeper or more complex circuits are more prone to noise, decoherence, and hardware limitations in real quantum devices.

Simplifying specific stages of the algorithmic pipeline may also lead to improved computational efficiency and scalability. For instance, state preparation and measurement post-processing are often computationally intensive steps that, if streamlined, can reduce runtime and improve scalability for larger datasets or longer time-series predictions. This reflects a practical consideration: even with a highly expressive ansatz, inefficiencies in implementation can limit overall performance, particularly when moving from simulation to real quantum hardware.

Moreover, extending the evaluation to longer and more diverse datasets would provide a more comprehensive assessment of the models' generalization capabilities and robustness over extended forecasting horizons. Indeed, the current results are limited by the dataset size and variety. Extending the evaluation to longer time-series and more heterogeneous data would allow for a more robust assessment of the models' generalization capabilities. In other words, it would test how well the models can maintain accurate predictions when faced with sequences that differ from the training data in length, temporal patterns, or statistical properties. This is particularly relevant for reservoir computing, where performance can be highly sensitive to both the richness of the training dataset and the complexity of temporal dependencies.

Furthermore, from an implementation perspective, refining the prediction procedure - especially the iterative forecasting strategy — could help mitigate error accumulation in long-term predictions. As a matter of fact, the iterative forecasting strategy is inherently prone to error accumulation: each predicted value is used as input for the next step, so small errors propagate and can amplify over long horizons. By refining this procedure — through techniques such as multi-step correction, hybrid prediction strategies, or more sophisticated update rules — the models could reduce long-term forecast errors and achieve more reliable predictions across extended horizons.

Additionally, the graphs can also be analyzed using error metrics other than the absolute error (AE) adopted in the main results. In particular, as discussed in more detail in the "Appendix" chapter, from both a visual and interpretative perspective, plots based on the Relative Error (RE) or the Normalized Mean Squared Error (NMSE) provide a clearer representation of the performance differences among the models. For this reason, RE and NMSE may represent convenient reference metrics for future analyses, as they offer a normalized scale for comparison and allow performance variations between different approaches to be highlighted more effectively.

Another promising direction for future work is the development of a hybrid strategy that combines the strengths of both the MERA and Repeated Measurements (RM) approaches within a multi-step iterative forecasting framework. Specifically, MERA could be employed to generate the initial prediction steps, where it demonstrates higher accuracy, while RM could be leveraged in the later stages, where it exhibits superior performance. Such a combination would allow the model to adapt its predic-

tive behavior across the forecasting horizon, potentially improving overall accuracy. This hybrid approach could better exploit the complementary characteristics of the two reservoirs, leading to more robust and reliable time-series predictions.

Finally, testing the proposed approaches on additional quantum hardware platforms and under different noise conditions would offer valuable insights into their portability, hardware dependence, and practical applicability in real-world time-series prediction tasks. In particular, testing on additional quantum hardware platforms is essential because different devices may have varying qubit connectivity, coherence times, gate fidelities, and native gate sets. These differences can significantly affect the performance of quantum reservoirs, making it crucial to assess the portability of the algorithms across devices. In other words, verifying that a given reservoir architecture performs reliably on multiple quantum platforms helps determine whether the approach is generally applicable or overly dependent on the specific characteristics of a single quantum computer.

To conclude, exploring different noise conditions — both intrinsic device noise and artificially injected noise — provides insight into the robustness of the proposed models. Understanding how sensitive the reservoir is to decoherence, gate errors, and measurement imperfections allows researchers to evaluate the practical applicability of these approaches in real-world time-series prediction tasks. This step is particularly important for near-term quantum devices, where noise is unavoidable, and the ability of an algorithm to tolerate or even exploit noise can determine its usefulness.

Appendix A

A.1 Machine Learning

Machine learning is a branch of Artificial Intelligence (AI) dedicated to creating algorithms and statistical models enabling computer system to learn from data. The models are a mathematical representation of the relationship within data; they are created by training an algorithm on a dataset, allowing the model to identify patterns and make predictions based on the input data. The models are defined by parameters (internal variables adjusted during training to minimize error in predictions) and by hyperparameters (configurable setting that influence how a model learns and generalizes). So, in general, models are not explicitly programmed; instead, they learn from historical data, enabling them to make predictions.

In the context of machine learning it is possible to recognize three different categories:

- Supervised Learning: the model learns from labeled data, where each example has a correct answer.
- Unsupervised Learning: the model aims to identify patterns in unlabeled data (clustering or association problems).
- Reinforcement Learning: the system learns through trial and error, receiving "rewards" for correct actions.

The information is provided thanks to the dataset: a collection of data, often organized in tabular form, where each row represents a unique record and each column corresponds to a variable.

Obviously, before passing the dataset to the machine, there are some pre-processing strategies:

- Data cleaning: serves to remove incomplete records, to eliminate duplicates and to impute missing values to prevent biased results.
- Data transformation: serves to normalize/standardize numerical features to a common range or distribution and to convert categorical data into numerical formats to allow models to process them.
- Feature engineering: serves to create meaningful features from existing data and to remove irrelevant features.

-
- Data splitting: serves to divide the data into two part, training and test, to evaluate the performance of the model (tipycally 80% for training and 20% for test).

To evaluate the discrepancy between the model's predictions and the actual values, a loss function is introduced. This function quantifies the degree of error in the predictions, effectively measuring how closely the predicted outcomes align with the true data.

Furthermore, to progressively reduce errors that arise during training, an optimizer can be used. It is an algorithm that iteratively updates a model's parameters to minimize the loss function, thereby improving the model's performance. It determines how the parameters should be adjusted based on the gradient (or estimated direction) of the loss function with respect to those parameters. By moving the parameters in the direction that reduces the loss, the optimizer guides the model toward better accuracy and generalization.

Among the most common optimization methods are Gradient Descent and its variants, such as Stochastic Gradient Descent (SGD), Momentum, RMSProp, and Adam. These algorithms differ in how they compute and apply parameter updates — for instance, Adam combines momentum and adaptive learning rate strategies to accelerate convergence and improve stability during training.

In essence, the optimizer serves as the mechanism that drives learning in machine learning models, allowing them to progressively refine their predictions through repeated exposure to data.

Appendix B

B.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed to process sequential data, where the order of the inputs and the relationships between past and present values are important [54]. They are widely used in fields such as time-series forecasting, natural language processing, speech recognition, and signal processing within the broader area of Machine Learning.

Unlike traditional feedforward neural networks, which treat each input independently, RNNs introduce recurrence, meaning that the network has internal connections that allow information from previous time steps to influence the current output. This gives the network a form of memory.

At each time step t , the network receives an input x_t and updates its internal hidden state h_t , which stores information about the past sequence. The hidden state is typically computed as:

$$h_t = f(W_x x_t + W_h h_{t-1} + b), \quad (\text{B.1})$$

where x_t is the input at time t , h_{t-1} is the hidden state from the previous time step, W_x and W_h are weight matrices, b is a bias term and $f(\cdot)$ is a nonlinear activation function (e.g., tanh or ReLU). Here, the output y_t can then be computed from the hidden state. Because the hidden state depends on previous states, the network effectively propagates information through time, allowing it to capture temporal dependencies in sequential data.

In particular, the defining feature of RNNs is their ability to retain information about previous inputs. This property makes them particularly suitable for tasks where the current prediction depends on past observations, such as time-series prediction, language modeling, speech processing and video analysis.

However, the memory of simple RNNs is often limited due to training difficulties such as the vanishing and exploding gradient problems. To address these issues, more sophisticated RNN architectures have been developed, including for example LSTM (Long Short-Term Memory) networks and GRU (Gated Recurrent Units).

These models introduce gating mechanisms that control how information is stored, forgotten, and passed through the network, enabling them to capture long-term dependencies more effectively.

In this context, Reservoir computing can be viewed as a special type of recurrent neural network. In traditional RNNs, all weights are trained during the learning process. In contrast, reservoir computing keeps the recurrent network (the reservoir) fixed and trains only the output layer, greatly simplifying the training procedure.

In summary, Recurrent Neural Networks are neural architectures specifically designed for sequential data processing. Their recurrent structure allows them to maintain an internal memory of past inputs, making them powerful tools for modeling temporal dependencies in a wide range of applications.

Appendix C

C.1 NARMA task

The NARMA task refers to a class of benchmark problems used to evaluate the temporal processing capability and memory capacity of dynamical systems such as recurrent neural networks and reservoir computing model [55].

NARMA stands for Nonlinear AutoRegressive Moving Average and it describes a nonlinear dynamical system in which the current output depends on:

- Previous outputs (autoregressive component),
- Past inputs (moving average component),
- Nonlinear interactions between these quantities.

In general, the system can be written as:

$$y_k = F(y_{k-1}, y_{k-2}, \dots, y_{k-n}, s_k, s_{k-1}, \dots, s_{k-n}), \quad (\text{C.1})$$

where y_k is the output at time k , s_k is the input signal and n is the order of the system, representing how many past time steps influence the output. Here, the presence of nonlinear terms makes the task challenging and suitable for testing the computational power of temporal models.

In the case of reservoir computing and quantum reservoir computing, several standard NARMA benchmarks are used, the most common being NARMA10, NARMA20, NARMA30. The number indicates the memory length required by the system and for example, the NARMA10 task is defined by:

$$y_k = 0.3y_{k-1} + 0.05y_{k-1} \left(\sum_{i=1}^{10} y_{k-i} \right) + 1.5s_{k-10}s_{k-1} + 0.1 \quad (\text{C.2})$$

where s_k is a random input sequence and the output depends on the previous 10 outputs and past inputs.

So, NARMA systems are widely used because they test several important properties of temporal models; in particular:

- Memory capacity - the system must remember inputs and outputs from many previous time steps.

-
- Nonlinear processing - the equation contains nonlinear interactions, such as $y_{k-1} \sum y_{k-i}$.
 - Long-term dependencies - to solve the task, the model must retain information across long time windows.

Reservoir Computing models are particularly well-suited to NARMA benchmarks because their intrinsic dynamical reservoir naturally retains information about past inputs. The high-dimensional state space enables rich nonlinear transformations, while only the output layer is trained. This combination allows the system to effectively approximate the complex nonlinear dependencies between past inputs and outputs.

In simple terms, a NARMA task asks a model to reproduce the behavior of a nonlinear dynamical system whose output depends on many past inputs and outputs. It is essentially a stress test for temporal learning systems, evaluating their ability to handle memory, nonlinear computation and time dependencies.

Appendix D

D.1 Quantum Gates

A quantum gate is the quantum equivalent of a classical logic gate, but it operates on quantum states (superpositions and entanglement). Mathematically, each gate is represented by a unitary matrix U such that:

$$U^\dagger U = I \quad (\text{D.1})$$

So, quantum gates act on qubits, changing their state according to:

$$|\psi'\rangle = U|\psi\rangle \quad (\text{D.2})$$

Unlike classical gates, they are intrinsically reversible and are the fundamental building blocks of quantum circuits that execute complex quantum algorithms.

The main features to remember to fully understand the topic are:

- Analogy with classic gates: quantum gates are the fundamental building blocks of quantum circuits, similar to how logic gates (AND, OR, NOT) are the building blocks of classical digital circuits.
- Operations on qubits: they perform manipulations on one or more qubits, exploiting quantum phenomena such as superposition and entanglement.
- Reversibility: quantum gates are always reversible, while most classical gates (with the exception of the NOT gate) are not.
- Transformations: the transformation performed by a quantum gate is represented by an operator, often in the form of a matrix.

Some examples of quantum gates include:

- Hadamard gate (H): one of the most important single-qubit gates, it creates a uniform superposition of both possible states of a qubit.
- Pauli gates (X, Y, Z): these are Pauli matrices that perform transformations such as negation (X, similar to a NOT) or rotations of quantum states.
- CNOT gate: it is a two-qubit gate and represents the analogue of the classical XOR.

Quantum gates are combined sequentially to build quantum circuits, which implement quantum algorithms. A circuit typically consists of state preparation, unitary transformations (quantum gates) and measurement; the sequence of gates determines how the quantum state evolves before measurement collapses it to classical information.

In quantum machine learning, quantum gates are used to construct encoding circuits that map classical data into quantum states or variational ansatz containing parameterized gates that can be trained. These parameterized gates allow the circuit to be optimized similarly to neural networks.

In summary, quantum gates are unitary operations that manipulate qubit states and control the evolution of quantum systems in a circuit. By combining single-qubit rotations and multi-qubit entangling gates, complex quantum algorithms and quantum machine learning models can be implemented. Their ability to create superposition and entanglement distinguishes them fundamentally from classical logic gates and enables the unique computational capabilities of quantum computers.

For a more detailed list of quantum gates, please refer to the summary table below.

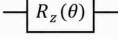
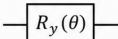
Gate	Symbol	Matrix
Pauli-X (NOT)		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli-Y		$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli-Z (Phase flip)		$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
S		$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
T		$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$
Hadamard		$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Rotation Z		$\begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$
Rotation Y		$\begin{pmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$
Rotation X		$\begin{pmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$
CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
Swap		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Figure D.1: List of single-qubit, two-qubit, and rotational quantum gates.

Appendix E

E.1 Quantum Measurement and State Collapse

In quantum mechanics, measurement is the process through which information is extracted from a quantum system. Unlike classical systems, where properties have definite values prior to observation, a quantum system is generally described by a superposition of multiple possible states. For instance, a qubit can exist in a linear combination of $|0\rangle$ and $|1\rangle$, with complex probability amplitudes.

When a measurement is performed, the system does not reveal all components of the superposition. Instead, it yields a single outcome, and the probability of obtaining each result is given by the squared magnitude of the corresponding amplitude (according to the Born rule). This probabilistic nature is a fundamental aspect of quantum theory.

The act of measurement leads to the so-called state collapse (or wavefunction collapse). This means that, immediately after measurement, the quantum state is projected onto the observed outcome. For example, measuring a qubit in superposition will force it into either $|0\rangle$ or $|1\rangle$, and all previous superposition information is lost.

The state collapse is not a gradual process — it's treated as instantaneous in standard quantum mechanics; in addition, it is irreversible: once measured, you cannot reconstruct the original superposition.

This phenomenon has important consequences in quantum computing. Measurement is typically performed at the end of a quantum circuit to extract classical information from qubits. However, intermediate measurements — such as those used in quantum reservoir computing or feedback-based models — can influence the system's evolution through measurement back-action, effectively altering future dynamics.

In quantum circuit diagrams, measurement operations are visually represented using specific symbols, as illustrated below (Figure E.1). The measurement maps a quantum state to a classical bit, which can then be used in the circuit for conditional operations. Here, the classical result is often drawn as a double line or single line labeled c , going from the measurement to other gates that depend on the measurement result.

Quantum measurement is not merely a passive observation but an active process that fundamentally changes the state of the system, distinguishing quantum behavior from classical intuition. Measurement is probabilistic - even if you prepare the same state many times, outcomes can vary according to the probabilities and the classical bit after measurement stores the collapsed outcome.

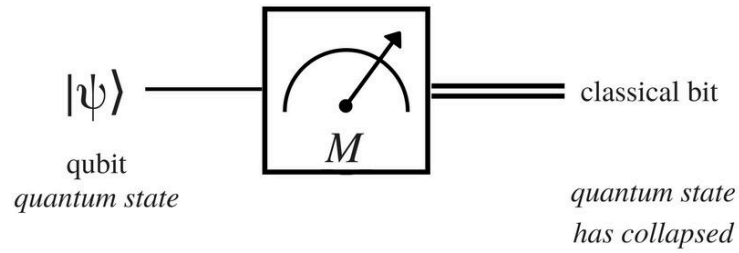


Figure E.1: Quantum Measurement and State Collapse.

In conclusion, this process is a fundamental aspect of quantum mechanics that bridges the quantum and classical worlds. A qubit in superposition represents multiple possibilities simultaneously, but once a measurement is performed, the system undergoes a wavefunction collapse, selecting a definite classical outcome according to the probabilistic rules dictated by the quantum state. This collapse is instantaneous and irreversible, and it is the mechanism through which quantum information becomes accessible to classical systems. In quantum circuits, measurement gates not only extract this classical information but also influence subsequent operations, enabling conditional logic based on the outcomes. Understanding the concept of collapse is therefore essential for designing, analyzing, and interpreting quantum algorithms and experiments.

Appendix F

F.1 Correlation and entanglement

Correlation refers to a statistical relationship between two or more variables or systems. In classical settings, correlations arise when variables share common information or are influenced by the same underlying factors. For example, knowing the value of one variable may provide partial information about another, but each still possesses well-defined properties independently of observation.

Entanglement, on the other hand, is a uniquely quantum form of correlation with no classical counterpart. When two or more qubits are entangled, their joint quantum state cannot be described as a simple product of individual states. Instead, the system must be treated as a whole, meaning that the measurement outcome of one qubit is intrinsically linked to the outcome of the other, regardless of the distance between them.

A key distinction is that classical correlations can always be explained by shared information or hidden variables, whereas entanglement produces non-classical correlations that violate classical bounds. For example, Bell inequalities - introduced by physicist John Bell in 1964 - are fundamental results to test whether the predictions of quantum mechanics could be explained by any local hidden variable theory—that is, a classical model in which measurement outcomes are predetermined and no information travels faster than light. In a typical Bell test, two distant observers measure correlated particles (e.g., entangled qubits) using different measurement settings. Classical theories impose constraints on the strength of the correlations that can be observed. These constraints are expressed mathematically as inequalities, such as the CHSH inequality: $|S| \leq 2$, where S is a combination of correlation functions between measurement outcomes.

Quantum mechanics, however, predicts that entangled states can violate this bound, reaching values up to: $|S| \leq 2\sqrt{2}$. This violation has been experimentally confirmed many times and demonstrates that quantum correlations cannot be reproduced by any classical, local theory [56].

The following diagram (Figure F.1) presents a quantum circuit used to generate an entangled pair of qubits.

It is worth noting that the entanglement generated by this circuit can be verified through measurement outcomes: repeated measurements in the computational basis yield perfectly correlated results, with only the states $|00\rangle$ and $|11\rangle$ observed. Following this construction, the resulting entangled state plays a fundamental role in quantum information processing,

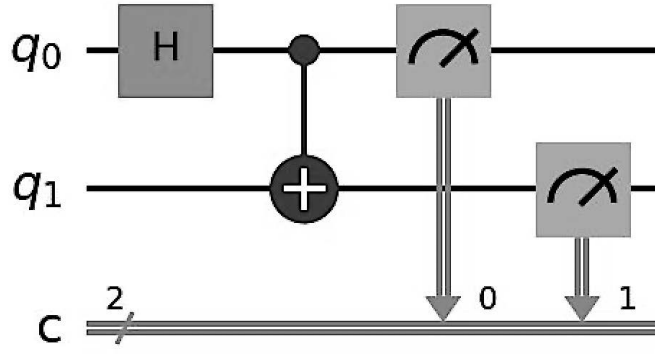


Figure F.1: Quantum circuit for the preparation of an entangled two-qubit state. The first qubit is initialized in the $|0\rangle$ state and transformed into a superposition by applying a Hadamard gate, resulting in $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Subsequently, a controlled-NOT (CNOT) gate is applied, using the first qubit as control and the second as target. This operation entangles the two qubits, producing the Bell state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, in which the measurement outcomes of the two qubits are perfectly correlated.

as it enables the establishment of strong correlations between qubits that cannot be explained classically. Such correlations are at the core of many quantum protocols, including quantum communication and quantum-enhanced computation. Moreover, the ability to generate and control entanglement through simple gate sequences, such as the Hadamard–CNOT combination, highlights one of the key advantages of quantum circuits over classical systems.

Appendix G

G.1 Noise, decoherence and computational fidelity

In quantum computing, noise refers to any unwanted interaction or imperfection that alters the state of a quantum system during its evolution. Unlike classical systems, quantum states are extremely sensitive to their environment, meaning that even small disturbances can significantly affect computation. These effects manifest through processes such as decoherence, where quantum superpositions degrade over time, and operational errors, arising from imperfect gate implementations and measurements.

Noise is particularly relevant in the current era of quantum technology, commonly referred to as NISQ (Noisy Intermediate-Scale Quantum) devices. These systems, which include today's quantum processors, are characterized by a limited number of qubits and the absence of full error correction. As a result, computations performed on NISQ hardware are inherently affected by noise, restricting circuit depth and overall reliability.

Several common types of quantum noise are typically considered:

- Decoherence (T_1 and T_2 processes) that is a loss of energy (relaxation) and loss of phase coherence (dephasing), which degrade quantum information over time.
- Gate errors that is inaccuracies in the implementation of quantum operations due to hardware limitations.
- Measurement errors that is incorrect readout of qubit states.
- Noise channels such as depolarizing, amplitude damping, and phase damping, which model different physical error mechanisms.

In practice, noise impacts the fidelity of quantum computations, reducing the agreement between the ideal (noise-free) outcome and the actual result obtained from hardware. This is why simulations are often performed both in ideal conditions (noise-free backends) and under realistic noise models, in order to better understand how algorithms behave when deployed on real devices.

In this context, the concept of computational fidelity becomes particularly important, as it provides a quantitative measure of how accurately a quantum system reproduces the expected outcomes of a given computation [57]. It measures the closeness between the ideal (theoretical) output of a quantum algorithm and the actual output obtained from a real or simulated (possibly noisy) quantum device.

In general terms, fidelity evaluates how similar two quantum states are. If ρ represents the ideal quantum state and σ the noisy state, the fidelity is defined as:

$$F(\rho, \sigma) = \left(\text{Tr} \sqrt{\sqrt{\rho} \sigma \sqrt{\rho}} \right)^2. \quad (\text{G.1})$$

For pure states (i.e. states that contain the maximum possible information about quantum systems and that represent systems that are in a well-defined quantum states [58]) $|\psi\rangle$ and $|\phi\rangle$, this simplifies to: $F = |\langle\psi|\phi\rangle|^2$.

So, a fidelity value of $F = 1$ indicates perfect agreement while $F = 0$ indicates completely different states. In general, the computational fidelity reflects how closely a quantum circuit reproduces the expected probability distribution, the accuracy of gate operations and state evolution or the impact of noise and hardware imperfections. For example, when running a circuit on real quantum hardware, the output distribution of measurement results is often compared to the ideal (noise-free) distribution. The closer the two distributions, the higher the computational fidelity.

Overall, addressing noise is one of the central challenges in quantum computing. Techniques such as error mitigation, noise-aware training, and eventually quantum error correction are essential for improving the performance and scalability of quantum algorithms in the NISQ era.

Appendix H

H.1 IQM SPARK

The IQM Spark is a 5-qubit quantum computer (Figure H.1) designed for universities and research laboratories, offering an accessible and scalable solution for learning and research. Its technical features include a typical gate fidelity of 99.9% for single qubits and 99.0% for dual qubits (CZ), a typical single-qubit operation time of 20 ns and a typical dual-qubit operation time of 60 ns [41].

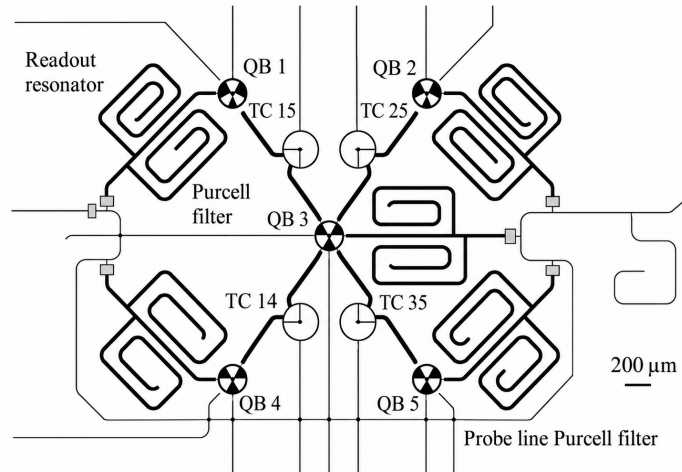


Figure H.1: The topologic map of the device IQM Spark.

It is part of the IQM production that deals with vertically integrated full-stack quantum computers that institutions install, operate and evolve with full-stack visibility into hardware, control and software. It is installed in Turin and is IQM's first commercial quantum computer in Italy; it is also called "Lagrange". Inaugurated on May 22, 2025, the system is the result of a strategic collaboration between Politecnico di Torino, Fondazione LINKS and Istituto Nazionale di Ricerca Metrologica (INRiM).

Appendix I

I.1 Additional considerations on Prophet model

The Prophet model is an open-source forecasting tool designed for time series prediction, particularly when the data exhibit strong seasonal patterns and missing observations. It was developed by Meta Platforms (formerly Facebook) and released as the software package Prophet. The model is widely used in business analytics, economics, and data science for forecasting quantities such as sales, web traffic, energy demand, and financial indicators.

Prophet is based on an additive time series decomposition, where the observed time series is modeled as the sum of several interpretable components:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t, \quad (\text{I.1})$$

where $g(t)$ represents the trend, describing long-term growth or decline in the time series; $s(t)$ represents seasonality, capturing periodic patterns such as daily, weekly, or yearly cycles; $h(t)$ represents holiday effects, allowing the model to incorporate known events that may affect the time series (e.g., holidays, promotions, or special events); ϵ_t is the error term, accounting for random fluctuations not explained by the model.

This structure makes Prophet highly interpretable because each component corresponds to a meaningful aspect of the data.

In general, the trend component is modeled as a piecewise function with automatic change points. These change points allow the model to adapt when the growth rate of the time series changes over time. Instead of assuming a single constant trend, Prophet identifies moments where the trend slope may shift. In particular, two common trend models are used: Piecewise linear trend and Logistic growth (useful when the time series has a natural upper limit, e.g. market saturation).

Instead, seasonal effects in Prophet are modeled using Fourier series, which allow flexible periodic patterns to be represented mathematically. This enables the model to capture multiple seasonalities simultaneously, such as weekly patterns, yearly cycles and daily patterns in high-frequency data.

A distinctive feature of Prophet is its ability to incorporate user-defined events. These events can influence the time series for specific dates or time windows, allowing the model to account for effects such as holidays, marketing campaigns, or system outages. This predictive-model is robust to missing data and outliers, has an automatic detection

of trend changes, handles multiple seasonalities, has interpretable components and it is easy to use with minimal parameter tuning.

Because of these properties, it is particularly useful for real-world forecasting problems where data are noisy and irregular like business and sales forecasting, website traffic prediction, demand forecasting in supply chains, energy consumption prediction, financial time series analysis etc.

Despite its strengths, Prophet also has some limitations:

- It assumes an additive structure, which may not capture complex nonlinear relationships.
- It may perform worse than advanced machine learning or deep learning models when strong nonlinear dependencies exist.
- It is primarily designed for univariate time series.

In summary, Prophet is a decomposable and interpretable time-series forecasting model that represents data as the combination of trend, seasonal patterns, event effects, and noise. Its main strengths lie in its flexibility, robustness to missing or noisy data, and minimal need for parameter tuning, making it particularly suitable for practical forecasting tasks involving structured temporal patterns. While it is not specifically designed to capture complex nonlinear dynamical behaviors, Prophet provides an efficient and transparent framework for modeling time series characterized by regular trends and seasonality.

Appendix J

J.1 Additional comparisons using different error metrics

In addition to the primary evaluation metric considered in the previous analyses (absolute error), it is useful to examine the model performance using alternative error metrics. Different metrics capture distinct aspects of the prediction accuracy and can therefore provide a more comprehensive assessment of the forecasting capabilities of the considered approaches.

The plots presented below provide a final additional comparative overview of the performance of all the considered backends across the different evaluation metrics. In particular, they summarize the results obtained for each implementation by reporting the prediction errors computed using multiple error measures. This comparison allows for a more comprehensive assessment of the forecasting performance, highlighting how the different backends behave when evaluated under various metrics. By analyzing these plots, it is possible to obtain a consolidated view of the relative performance of the classical and quantum approaches, as well as the differences between simulations and real quantum hardware implementations.

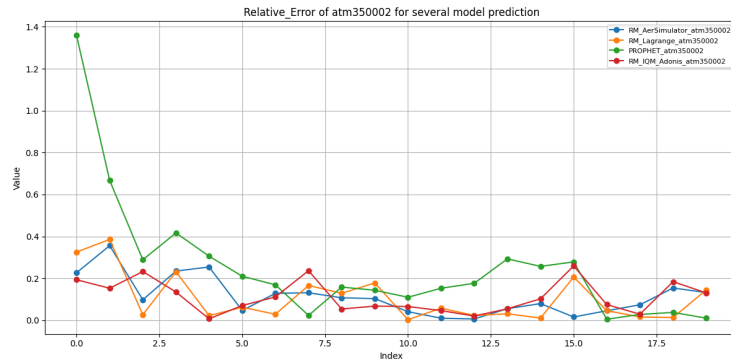


Figure J.1: Relative error (RE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.

Overall, the analysis performed using the different error metrics shows consistent results across the evaluated models and backends. Each metric captures a specific aspect of the prediction accuracy, and together they provide a comprehensive assessment of the forecasting performance. Despite their differences, the general trends observed in the results remain largely the same across all considered metrics, confirming the reliability of

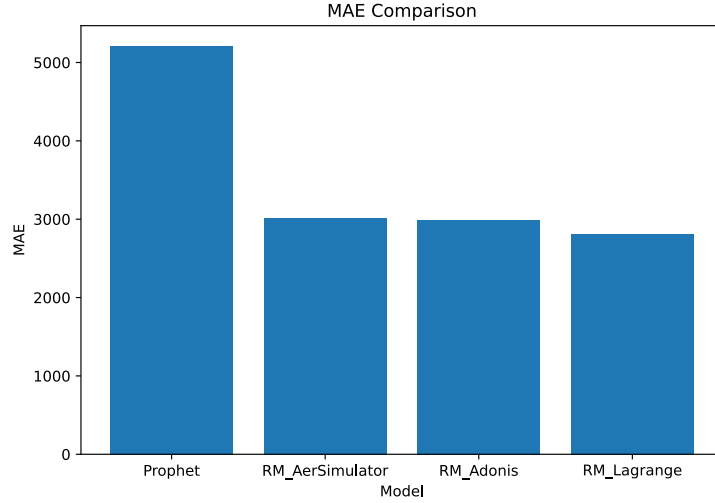


Figure J.2: Mean absolute error (MAE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.

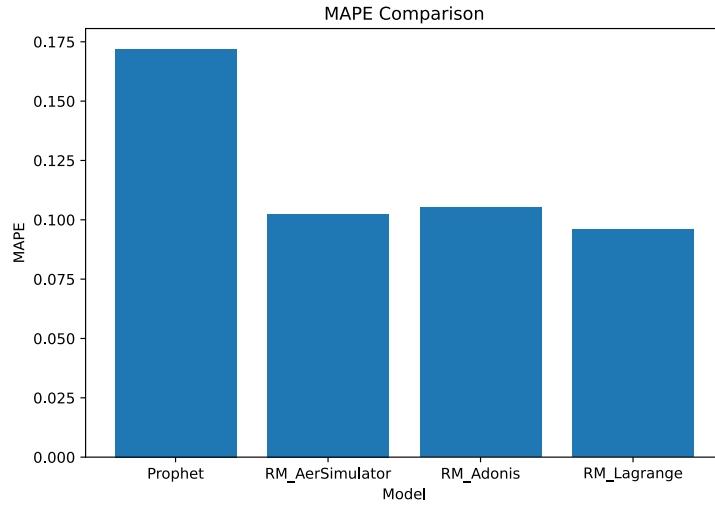


Figure J.3: Mean absolute percentage error (MAPE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.

the evaluation. From a visual and interpretative perspective, however, the plots based on the Relative Error (RE) or on Normalized Mean Squared Error (NMSE) appear to provide the clearest representation of the performance differences among the models. For this reason, RE or NMSE could serve as a convenient reference metric in future analyses, offering both a normalized scale for comparison and a visually effective way to highlight performance variations between different approaches.

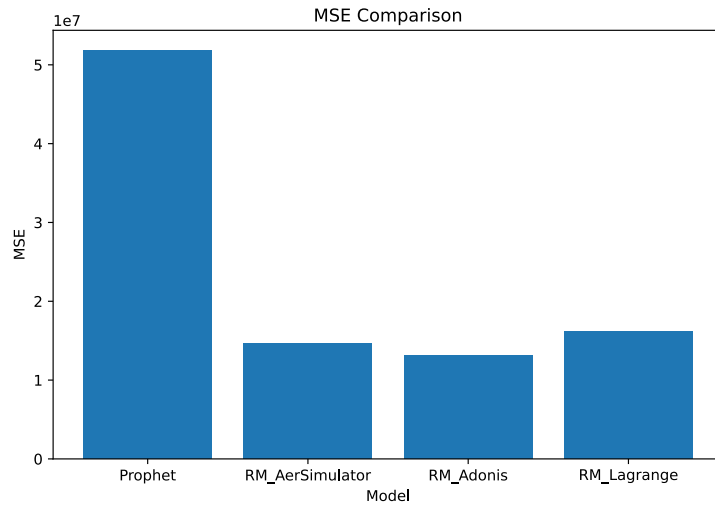


Figure J.4: Mean squared error (MSE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.

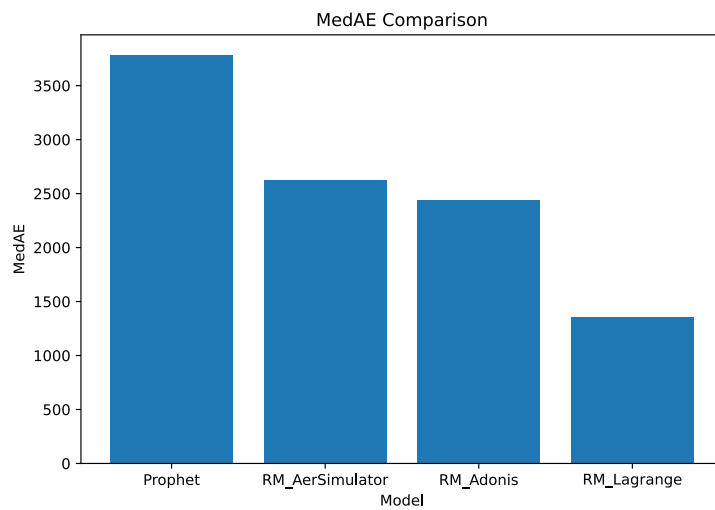


Figure J.5: Median absolute error (MdAE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.

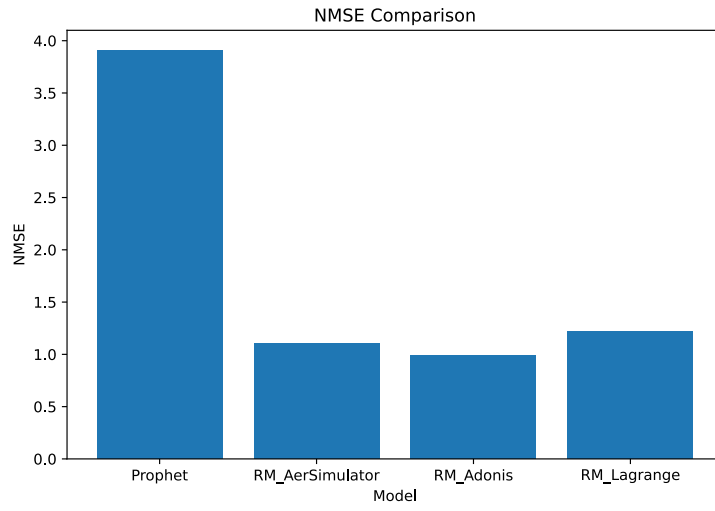


Figure J.6: Normalized mean squared error (NMSE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.

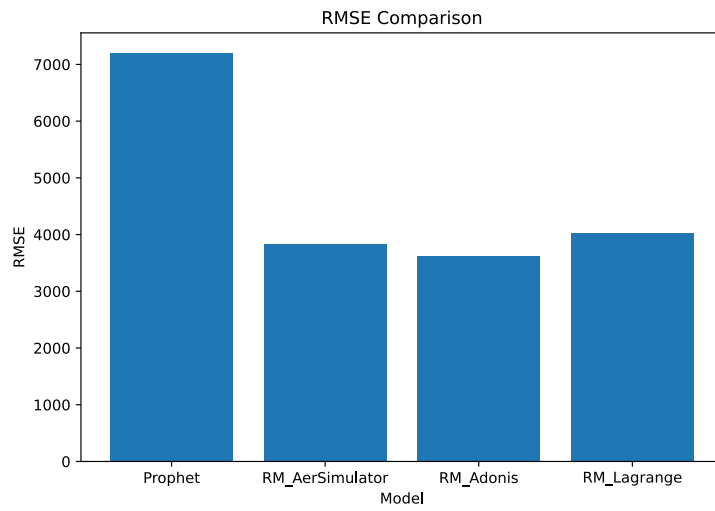


Figure J.7: Root mean squared error (RMSE) comparison between Prophet, AerSimulator, Adonis and Lagrange (IQM Spark) backends.

Appendix K

K.1 Computational tools and development environments

The development and evaluation of the proposed models were carried out using a combination of computational tools for both implementation and documentation. All simulations and experiments were conducted in a Python environment, providing a flexible and efficient framework for data analysis and model development. In particular, data preprocessing and numerical computations were performed using *NumPy* and *Pandas* libraries, while data visualization and result analysis were supported by *Matplotlib*.

The classical baseline model was implemented using the *Prophet* library for time-series forecasting. In addition, machine learning utilities and regression models were supported by *scikit – learn*, which was used for tasks such as model evaluation and training procedures. For the quantum machine learning component, quantum reservoir computing models were developed using *Qiskit* and the *QuantumReservoirPy* library. Simulations were carried out using the AerSimulator for ideal conditions and noise-aware backends such as Fake Adonis, as well as executions on real quantum hardware (IQM Spark).

IDE - Integrated Development Environment - like PyCharm, Google Colab, Visual Studio Code were used for coding, testing, and debugging the models, offering flexibility between cloud-based and local execution. These platforms facilitated rapid prototyping, efficient resource management, and an interactive workflow during the development process.

To ensure an organized development process and a coherent workflow throughout the project, GitLab was adopted as the primary version control and collaboration platform. Its use enabled systematic tracking of code changes, facilitating iterative improvements and reducing the risk of errors or data loss. Moreover, GitLab supported efficient management of different development stages through branching and versioning, allowing experimental features and model variations to be tested in a controlled manner. This approach contributed to maintaining a clear structure of the project, improving reproducibility, and ensuring a consistent progression of the work.

From a documentation perspective, the entire work was written using *LaTeX*, ensuring a clear and professional presentation of mathematical formulations, figures, and references. This integrated set of tools enabled a reproducible workflow, supporting both the experimental analysis and the structured communication of the results.

Overall, this ecosystem of libraries provided a robust and scalable environment for the implementation and comparison of classical and quantum approaches to time-series

forecasting.

All references and citations included in this work were selected through a careful review of academic and scientific literature. In particular, sources were identified by consulting *Google Scholar* and accessing materials available through the university library *Pico PoliTo*, including peer-reviewed journal articles, conference papers, and specialized books. Each reference was critically evaluated to ensure its relevance and reliability, with the aim of grounding the study in established research and providing a solid theoretical and methodological foundation.

Appendix L

L.1 Project repository structure

The project repository is organized following a modular and hierarchical structure, designed to ensure clarity, reproducibility, and ease of navigation. The main components are divided into dedicated directories, each corresponding to a specific stage of the workflow.

Each folder presents the codes and the corresponding plots obtained by running them; specifically, the temporal evolution of a bank balance sheet for several ATMs was analyzed. Each folder solves the time series prediction problem using classical (Prophet) and quantum (Quantum Reservoir) computational techniques.

Each folder contains the input file needed to export the same dataset consisting of 359 ATMs with balance sheet variations at different times of the day for a week.

Each folder contains a sub-folder named "img" containing the results and plots of the various simulations; only a few example results are shown, as the total number of images obtained is quite high.

Each folder contains the files for the correct simulation of the program; to make it work, export all the elements of the considered repository and run the files respectively: "Prophet.py" and/or "qrc_RM.py" and/or "qrc_MERA.py", "qrc_RM_bit_string.py" and/or "qrc_MERA_bit_string.py".

For the quantum part, two possible applications were considered: Repeated Measurement and MERA ansatz. Both versions are available in the repositories.

For each simulation, different types of errors were calculated and plotted on graphs.

The QRC_EVOLUTION folder represents an upgrade from the standard version of QRC analyzed previously: it uses, as `x_train`, the probability distribution of the states '00', '01', '10', '11' within the result string, instead of the simple probability distribution of '0' and '1' used in the previous case.

The QRC_EVO_2 folder represents an upgrade from the QRC_EVOLUTION as it considers, in addition to the probability distribution used in the case mentioned above, the dyadic correlation and the Pearson coefficient calculated directly from the counts of the pairs '00', '01', '10', '11' to be used for `x_train`.

The QRC_INCREMENTAL folder represents another version of the reservoir computing that uses the class Incremental for the simulation of the algorithm. It considers both the noiseless and noisy models for the application of the QRC using the fake provider IQMFakeAdonis.

The QRC_MODEL folder contains simulations for various types of models such as LinearRegression, Ridge, SGDRegressor, ElasticNet, Lasso. In the previous repositories, LinearRegression has been referred to as a learning model.

The LAGRANGE_IQM folder contains simulations of the implemented code exploiting the IQM SPARK machine situated on Politecnico di Torino (i.e. Lagrange Quantum Computer).

The MEMORY_COMPARISON and STANDARD_DEV_DATA folders serve to obtain final comparison plot in terms of performance for the different simulations.

The dataset is divided and analyzed individually for each ATM; in particular, it contains 84 total balances over time for each ATM.

The training portion is considered to represent 16/21 of the dataset, meaning ≈ 64 rows of total balance.

All the details are also reported in the README.md file, available like all the rest of the files on the GitLab repository.

This structured organization enables a clear separation of concerns, allowing each component of the project to be developed, tested, and extended independently, while maintaining a coherent and efficient workflow.

Appendix M

M.1 Project contributions and supporting resources

This work has been carried out by the author in collaboration with the Professor and supervisors of LINKS Foundation; it is inspired by existing research and academic literature addressing similar topics in quantum reservoir computing and quantum machine learning. The study builds upon these foundations while proposing and analyzing its own implementation and experimental results.

For readers interested in gaining a deeper understanding of the methodologies adopted, or in emulating and extending the proposed framework, a dedicated GitHub repository (available online at the link: <https://github.com/OpenQuantumComputing/quantumreservoirpy>) contains relevant materials related to the basic implementation of quantum reservoir computing models, including code, examples, and supporting resources.

Further details on the theoretical background and related works can be found in the references cited throughout this document. For additional information, clarifications, or potential collaborations, the author can be contacted via email on marco.venditti@studenti.polito.it. All experiments have been designed to ensure reproducibility: the code, parameter configurations, and evaluation procedures are made available in the associated repository, allowing other researchers to replicate and validate the results. Detailed documentation is also provided to guide the execution of the simulations.

The proposed framework can be extended in several directions, including the integration of more advanced quantum circuits, the exploration of hybrid classical–quantum training strategies, and the application to larger and more complex datasets. Remember that the dataset used in this study is publicly available, ensuring transparency and enabling further comparative studies.

Beyond the theoretical contribution, the explored methods have potential applications in real-world scenarios, particularly in financial forecasting, where capturing complex temporal dependencies is crucial.

This work approach aims to promote transparency, reproducibility, and accessibility, encouraging further developments in the field.

Bibliography

- [1] M. Lukoševičius, H. Jaeger, and B. Schrauwen. “Reservoir computing trends”. In: *KI-Künstliche Intelligenz*, 26(4), 365–371 (2012) (cit. on p. 1).
- [2] K. Fujii and K. Nakajima. “Quantum reservoir computing: a reservoir approach toward quantum machine learning on near-term quantum devices.” In: *In Reservoir Computing: Theory, Physical Implementations, and Applications* (pp. 423–450). Singapore: Springer Singapore (2021) (cit. on p. 2).
- [3] Robert S. Sutor. *Dancing with Qubits*. St Paul’s Square, Birmingham: Packt Publishing, 2024 (cit. on p. 2).
- [4] C. Hughes, J. Isaacson, A. Perry, R. F. Sun, and J. Turner. “What is a qubit?” In: *Springer International Publishing* (2021), In Quantum computing for the quantum curious, pp. 7–16 (cit. on p. 2).
- [5] L. K. Grover. “The advantages of superposition”. In: *Science* 280(5361) (1998), pp. 228–229 (cit. on p. 4).
- [6] C. P. Williams. *Quantum gates. In Explorations in quantum computing*. London: Springer London, 2011, pp. 51–122 (cit. on p. 4).
- [7] H. Zbinden, J. Brendel, W. Tittel, and N. Gisin. “Experimental test of relativistic quantum state collapse with moving reference frames”. In: *Journal of Physics A: Mathematical and General* 34(35) (2001), pp. 7103–7109 (cit. on p. 4).
- [8] R. Horodecki, P. Horodecki, M. Horodecki, and K. Horodecki. “Quantum entanglement”. In: *Reviews of modern physics* 81(2) (2009), pp. 865–942 (cit. on p. 4).
- [9] D. Braun N. Wiebe and S. Lloyd. “Quantum algorithm for data fitting”. In: *Phys. Rev. Lett.* 109, 050505 (2012) (cit. on p. 5).
- [10] S. Schaal and C. G. Atkeson. “Learning control in robotics”. In: *IEEE Robot Autom. Mag.* 17 (2010), pp. 20–29 (cit. on p. 5).
- [11] L. Alchieri, D. Badalotti, P. Bonardi, and S. Bianco. “An introduction to quantum machine learning: from quantum logic to quantum deep learning”. In: *Quantum Machine Intelligence* 3(2) (2021), Article 28 (cit. on p. 7).

- [12] S. Chakraborty, T. Das, S. Sutradhar, M. Das, and S. Deb. “An analytical review of quantum neural network models and relevant research”. In: *2020 5th International Conference on Communication and Electronics Systems (ICCES), IEEE* (2020), pp. 1395–1400 (cit. on p. 7).
- [13] S. C. Kak. “Quantum neural computing”. In: *Adv Imaging Electron Phys* 94 (1995), pp. 259–313 (cit. on p. 7).
- [14] D. Ventura and T. Martinez. “A quantum associative memory based on grover’s algorithm”. In: *In: ICANNGA* (1999) (cit. on p. 7).
- [15] N. Matsui, M. Takai, and H. Nishimura. “A network model based on qubitlike neuron corresponding to quantum circuit”. In: *Electron Commun Jpn (Part III: Fund Electron Sci)* 83(10) (2000), pp. 67–73 (cit. on p. 7).
- [16] M. V. Altaisky. “Quantum neural network”. In: *arXiv:quant-ph/0107012* (2001) (cit. on p. 7).
- [17] S. Lloyd, M. Mohseni, and P. Rebentrost. “Quantum algorithms for supervised and unsupervised machine learning”. In: *arXiv:1307.0411* (2013) (cit. on p. 7).
- [18] D. Anguita, S. Ridella, F. Riviello, and R. Zunino. “Quantum optimization for training support vector machines”. In: *Neural Netw* 16(5-6) (2003), pp. 763–770 (cit. on p. 7).
- [19] P. Wittek. “Quantum machine learning: what quantum computing means to data mining”. In: *Academic Press* (2014) (cit. on p. 7).
- [20] M. Broughton et al. “TensorFlow quantum: a software Framework for quantum machine learning”. In: *arXiv:2003.02989* (2020) (cit. on p. 7).
- [21] Montanaro A. “Quantum algorithms: an overview”. In: *npj Quantum Information* 2 (2016), pp. 1–8 (cit. on p. 8).
- [22] X. Guo, T. Muta, and J. Zhao. “Quantum Circuit Ansatz: Patterns of Abstraction and Reuse of Quantum Algorithm Design”. In: *IEEE International Conference on Quantum Software (QSW)* (2024), pp. 69–80 (cit. on p. 8).
- [23] S. Rizzi M. Montangero and Vidal G. “Simulation of time evolution with multiscale entanglement renormalization ansatz”. In: *Physical Review* 77 (2008) (cit. on p. 8).
- [24] Vidal G. “Class of quantum many-body states that can be efficiently simulated”. In: *Physical review letters* 101 (2008) (cit. on p. 9).
- [25] J. Chen, H. I. Nurdin, and N. Yamamoto. “Temporal Information Processing on Noisy Quantum Computers”. In: *Physical Review Applied* (2020), 14(2) (cit. on p. 9).
- [26] T. Yasuda, Y. Suzuki, T. Kubota, K. Nakajima, Q. Gao, W. ... Zhang, and N. Yamamoto. “Quantum reservoir computing with repeated measurements on superconducting devices”. In: *arXiv preprint* (2023) (cit. on pp. 9, 10, 12).

- [27] C. Gallicchio and A. Micheli. “Deep echo state network (deepesn): A brief survey”. In: *arXiv preprint* 1712.04323 (2017) (cit. on p. 10).
- [28] W. Maass. “Liquid state machines: motivation, theory, and applications”. In: *Computability in context: computation and logic in the real world* (2011), pp. 275–296 (cit. on p. 10).
- [29] G. Van der Sande, D. Brunner, and M. C. Soriano. “Advances in photonic reservoir computing”. In: *Nanophotonics* 6(3) (2017), pp. 561–576 (cit. on p. 10).
- [30] L. C. G. Govia, G. J. Ribeill, G. E. Rowlands, H. K. Krovi, and T. A. Ohki. “Quantum reservoir computing with a single nonlinear oscillator”. In: *Physical Review Research* 3(1) (2021), p. 013077 (cit. on pp. 10, 11).
- [31] G. Dion, S. Mejaouri, and J. Sylvestre. “Reservoir computing with a single delay-coupled non-linear mechanical oscillator”. In: *J. Appl. Phys.* 152132 (2018), p. 124 (cit. on p. 11).
- [32] Y. Suzuki, Q. Gao, K. C. Pradel, K. Yasuoka, and N. Yamamoto. “Natural quantum reservoir computing for temporal information processing”. In: *Scientific reports* 12(1), 1353 (2022) (cit. on p. 12).
- [33] K. Hornberger. “Introduction to decoherence theory. In Entanglement and Decoherence: Foundations and Modern Trends”. In: *Phys. Rev. Lett.* (2009), pp. 221–276 (cit. on p. 12).
- [34] M. Lukoševičius and H. Jaeger. “Reservoir computing approaches to recurrent neural network training”. In: *Comput. Sci. Rev.* 3 127 (2009) (cit. on p. 13).
- [35] K. Kobayashi, K. Fujii, and N. Yamamoto. “Feedback-driven quantum reservoir computing for time-series analysis”. In: *PRX quantum* 5(4), 040325 (2024) (cit. on p. 13).
- [36] R. Martínez-Peña, G. L. Giorgi, J. Nokkala, M. C. Soriano, and R. Zambrini. “Dynamical Phase Transitions in Quantum Reservoir Computing”. In: *Physical Review Letters* 127(10), Article 100502 (2021) (cit. on pp. 13, 14).
- [37] A.F. Atiya and A.G. Parlos. “New results on recurrent network training: Unifying the algorithms and accelerating convergence”. In: *IEEE Trans. Neural Networks* 11 697 (2000) (cit. on p. 13).
- [38] Yaren Yilmaz. *ATM Transactions - Hour-based Weekly Transactions for the ATMs*. 2019. URL: <https://www.kaggle.com/datasets/yarenyilmaz/atm-transactions> (cit. on p. 15).
- [39] PyPI. *Prophet: Automatic Forecasting Procedure*. 2025. URL: <https://pypi.org/project/prophet> (cit. on p. 17).
- [40] A. Nag, R. Ranjan, and C. N. S. Vinoth Kumar. “An Approach on Cyber Crime Prediction Using Prophet Time Series”. In: *IEEE 7th International Conference for Convergence in Technology (I2CT)* (2022), pp. 1–5 (cit. on p. 17).

- [41] IQM Quantum Computers. *IQM Spark - 5 Qubits Affordable On-Prem Quantum Computer*. 2024. URL: <https://meetiqm.com/products/iqm-spark> (cit. on pp. 22, 28, 70).
- [42] PyPI. *QReservoirPy*. 2025. URL: <https://quantumreservoirpy.readthedocs.io/> (cit. on p. 23).
- [43] PyPI. *Quantumreservoirpy0.2*. 2024. URL: <https://pypi.org/project/quantumreservoirpy/> (cit. on p. 23).
- [44] Paul Bernard. *ReservoirPy*. 2020. URL: <https://github.com/reservoirpy/reservoirpy> (cit. on p. 23).
- [45] Franz Georg Fuchs. *OpenQuantumComputing*. 2025. URL: <https://github.com/OpenQuantumComputing/quantumreservoirpy> (cit. on p. 25).
- [46] github.io. *AerSimulator - Qiskit Aer 0.17.1 - GitHub Pages*. 2024. URL: https://qiskit.github.io/qiskit-aer/stubs/qiskit_aer.AerSimulator.html (cit. on p. 27).
- [47] IBM Quantum Platform. *Fakeprovider (latest version) | IBM Quantum Documentation*. 2025. URL: <https://quantum.cloud.ibm.com/docs/api/qiskit-ibm-runtime/fake-provider> (cit. on p. 27).
- [48] DataCamp. *Eight Machine Learning Models Explained*. 2022. URL: <https://www.datacamp.com/blog/machine-learning-models-explained> (cit. on p. 28).
- [49] Scikit-learn. *LinearRegression — scikit-learn 1.7.2 documentation*. 2024. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (cit. on p. 29).
- [50] Scikit-learn. *Ridge — scikit-learn 1.7.2 documentation*. 2024. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html (cit. on p. 29).
- [51] Scikit-learn. *Lasso — scikit-learn 1.7.2 documentation*. 2024. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html (cit. on p. 29).
- [52] Scikit-learn. *ElasticNet — scikit-learn 1.7.2 documentation*. 2024. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html (cit. on p. 30).
- [53] Scikit-learn. *SGDRegressor — scikit-learn 1.7.2 documentation*. 2024. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html (cit. on p. 30).
- [54] D. Mandic and J. Chambers. “Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability”. In: *Wiley Online Library* (2001) (cit. on p. 58).
- [55] K. Nakajima and I. Fischer. *Reservoir computing*. Singapore: Springer, 2021 (cit. on p. 60).

- [56] R. F. Werner and M. M. Wolf. “Bell inequalities and entanglement”. In: *arXiv preprint quant-ph 0107093* (2001) (cit. on p. 66).
- [57] Y. C. Liang, Y. H. Yeh, P. E. Mendonça, R. Y. Teh, M. D. Reid, and P. D. Drummond. “Quantum fidelity measures for mixed states”. In: *Reports on Progress in Physics* 82(7), 076001 (2019) (cit. on p. 68).
- [58] B. L. Schumaker. “Quantum mechanical pure states with Gaussian wave functions”. In: *Physics Reports*, 135(6), 317-408 (1986) (cit. on p. 69).

Dedications

— FORMAL DEDICATIONS FOR EVERYONE —

— INFORMAL DEDICATIONS ARE WRITTEN ELSEWHERE IN ITALIAN —

This work is partially supported by the Politecnico di Torino in collaboration with the LINKS Foundations.

I thank Professor Bartolomeo Montrucchio, Dr. Chiara Vercellino and Dr. Giacomo Vitali for valuable comments, suggestions and availability for this project.

I also thank all the Professors, all the colleagues and all the friends who supported and that followed me during these wonderful years.

A special mention goes to my mother, my father, my brother and to all the rest of my family - grandparents, uncles, cousins - who saw me grow and always watched over me, helping me make the right choices and making me who I am today.

Finally, I thank you, dear reader, for having analyzed this document, in the hope that it may be useful for future research and work.