Quantum reservoir computing with many body systems

A Thesis

submitted to

Indian Institute of Science Education and Research Pune in partial fulfilment of the requirements for the BS-MS Dual Degree Programme

by

Dharmesh Yadav

Indian Institute of Science Education and Research Pune

Dr. Homi Bhabha Road,

Pashan, Pune 411008, INDIA.

April, 2025

Supervisor: M.S. Santhanam

Dharmesh Yadav

Certificate

This is to certify that this dissertation entitled Quantum reservoir computing with many-body systems towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Dharmesh Yadav at the Indian Institute of Science Education and Research, Pune under the supervision of M.S. Santhanam, Professor, Department of Physics, during the academic year 2024-2025.

Dharmesh Yadav

Date: 27 March 2025

Dr. M.S. Santhanam

Professor

Department of Physics

IISER Pune-411008

Declaration

I hereby declare that the matter embodied in the report entitled Quantum reservoir computing with many-body systems are the results of the work carried out by me at the Department of Physics, Indian Institute of Science Education and Research, Pune, under the supervision of M.S. Santhanam and the same has not been submitted elsewhere for any other degree

Dharmesh Yadav

Date: 27 March 2025

Dr. M.S. Santhanam

Professor

Department of Physics

IISER Pune-411008

Abstract

This thesis explores the implementation of Quantum Reservoir Computing (QRC) using random matrices and the Transverse Field Ising Model for time series prediction tasks. Given the limited theoretical understanding of QRC, this work investigates the dependence of reservoir dynamics on the performance of chaotic and memory-intensive time series. Our findings reveal that the optimal reservoir for learning is highly task-dependent, where chaotic time series are best predicted using random matrices, while memory-intensive tasks achieve optimal performance at the boundary between integrable and chaotic regimes. Additionally, we explore the connection between QRC and the Volterra expansion, demonstrating that QRC can be interpreted within this framework. This interpretation offers a deeper insight into chaos-boundary enhancement and, more broadly, the performance of various time series modelled using QRC.

Acknowledgements

I want to express my deepest gratitude, first and foremost, to Dr. M.S. Santhanam for his exceptional mentorship over the past two and a half years. His invaluable guidance and support have been instrumental in shaping this project.

I am also sincerely grateful to Dr. Bijay Kumar Agarwalla for being my TAC member and providing insightful feedback.

Furthermore, I truly appreciate the thoughtful discussions and suggestions from Nisarg Vyas and Bharathi Kannan, which were crucial for this research.

Lastly, I extend my heartfelt thanks to my family and friends for their unwavering support and encouragement during my most challenging moments. Their belief in me has been a source of strength throughout this journey.

Contents

List of Figures

Introduction

Machine learning [34] has transformed the world around us with its remarkable capabilities and applications in different areas. It has given the world the tools to approach complex problems in a completely new way. It offers a methodology for analyzing available data to identify underlying patterns and leverage this acquired knowledge to make informed predictions on new data. Due to the versatility of the techniques used, it has found applications in almost every aspect of technology and science [4] [33]. The problems in which it is predominantly used are image and speech recognition [21], time series prediction [3] and forecasting, and natural language processing [7].

One of the most powerful approaches in machine learning is artificial neural networks (ANNs) [9]. Started around the 1940s, it constitutes the building blocks of artificial intelligence and machine learning and has made remarkable progress in recent years. The design of these ANNs is inspired by the neurons present in the brain. It constitutes inter-connected networks with varying weights and activation functions to process information. Based on the design of the networks and weights constructed, we can classify neural networks into two major classifications - Feedforward neural networks (FNNs) and recurrent neural networks (RNNs). Feedforward neural networks (FNNs) [27] are mainly used for static (non-temporal) data processing, whereas RNNs [25] are suited for dynamic (temporal) data processing tasks.

Reservoir computing (RC) [20] is a technique based on the RNN architecture that is suitable for temporal tasks. The primary advantage of Reservoir Computing (RC) over traditional Recurrent Neural Networks is the reduction in training time. This efficiency is achieved by a reduction in the number of hidden layers and the training steps in backpropagation. In RC, only the weights in the output layer undergo training, whereas, in RNNs, the weights of all hidden layers must be updated through backpropagation. As a result, RC reduces both the number of training steps and the computational complexity associated with optimizing multiple layers, making it a more efficient alternative for time-series and sequential data processing. The two major architectures of reservoir computing are echo state networks (ESNs) and Liquid state machines (LSMs). ESN [20] is based on randomly connected networks and is used predominantly in reservoir computing, whereas LSM [13] is derived from spiking neurons in the brain and is used in very specific scenarios.

In this thesis, we have investigated quantum reservoir computing for various temporal processing tasks. Quantum reservoir computing is based on the classical reservoir computing architecture, with the reservoir being a dynamical and complex quantum system [11]. It demonstrates substantial potential for providing significant benefits across a wide range of classical

and quantum applications. This has been shown in many cases [5] [37], but very limited theoretical arguments are found that explain its limits and source of quantum advantage. This work aims to solve this problem in some special cases.

We have performed quantum reservoir computing on qubit-based systems, with the Hamiltonian being a random matrix and the Ising model. The dependence of these reservoirs on the performance of temporal learning tasks, such as time series prediction, was analyzed using quantum chaotic measures. The results indicate that the performance of memory-intensive tasks is model-dependent, whereas nonlinear tasks achieve optimal performance in quantum chaotic reservoirs regardless of the underlying model. Furthermore, the similarity between quantum reservoir computing and Volterra expansion is explored. It is shown that the performance of time series prediction derived using quantum reservoir computing can be explained by considering the Volterra expansion of the particular time series.

The chapters of the thesis are organized as follows.

- Chapter 1: Introduction to classical reservoir computing and its two major types- virtual reservoir and physical reservoir.
- Chapter 2: A brief overview of measures used to quantify quantum chaos is explained.
- Chapter 3: The general architecture of quantum reservoir computing, along with the scheme used throughout the study, is presented.
- Chapter 4: Quantum reservoir computing using random matrices is performed, and the results are compared with the transverse field Ising model.
- Chapter 5: Study of chaos - boundary enhancement is performed for various tasks and different reservoirs.
- Chapter 6: Study of chaotic time series prediction using random matrices as the reservoir.
- Chapter 7: The possibility of quantum reservoir computing to be written as a Volterra expansion is explored.
- Chapter 8: Results from the complete study are presented.

Chapter 1

Classical Reservoir Computing

Classical Reservoir Computing is an algorithm based on neural networks that computes the time evolution of dynamical systems. It was developed in 2001 by Herbert Jaeger [20] to integrate dynamical systems with high accuracy. To understand the advantages of this algorithm, we first need to have some context for neural networks.

1.1 Neural Networks

Neural networks [8] are machine learning models inspired by the brain's workings, mainly neurons. It is an interconnected network in which an input is fed, and it generates a desired output. It can be used in various machine learning problems, such as image processing, pattern recognition, natural language understanding, and time series prediction. The basic architecture of neural networks is shown in the figure 1.1.

It consists of many interconnected nodes that can take values from 0 to 1 based on some scheme. Each connection between nodes has an associated weight, and each node has a bias, both adjusted during training. These interconnected nodes are organized in layers that process the information sequentially. The layers can be classified into three primary layers.

1. **Input layer -** It takes the input, preprocesses it and passes it to hidden layers.
2. **Hidden layer -** These layers give neural networks their computational capabilities. They receive input from the input layer and generate various nonlinear combinations by applying weights, biases, and activation functions
3. **Output layer -** It receives the nonlinear output from the hidden layers and uses it to generate the final desired outcome.

The evolution equation of the nodes is $y = f(WX + b)$, where X is the input fed, W is the weights matrix of the interconnected network, b is the bias, and f is the activation func-

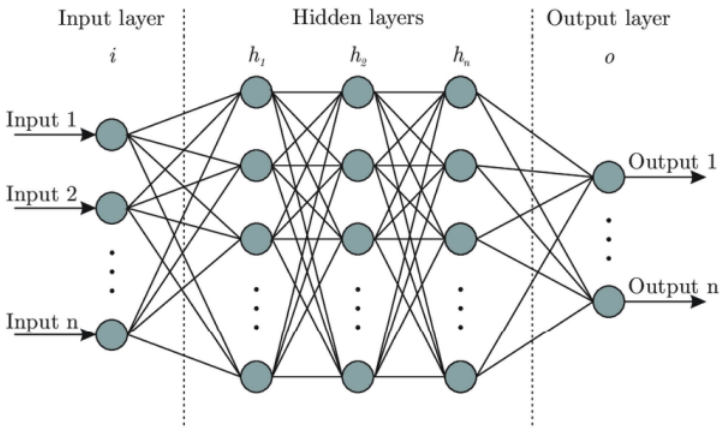

Figure 1.1: Pictorial representation of the architecture of a generic neural network. [8]

tion. This evolution scheme gives the neural network the nonlinearity needed for learning. The neural network learns the desired output by adjusting the weights corresponding to every connection through backpropagation. Various other processes exist for training the neural networks, but backpropagation is the most commonly used.

Depending on the type of hidden layers and the connections between the nodes, we can have many kinds of neural networks, each useful for a particular class of tasks. These different types are Feedforward Neural Networks (FNNs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and transformers. As RC is based on RNN, we will briefly introduce RNNs.

Recurrent Neural Networks (RNNs) [2] are a class of neural networks designed to process sequential data like a time series by maintaining an internal memory of past inputs. In this architecture, the output generated in one iteration is fed back to the neural network for further processing, thus preserving the memory of the inputs across many time steps. The evolution equation for RNN at a given time step depends on the current input and previous computations. It can be written as,

$$h_t = f(Wh_{t-1} + Ux_t + b)$$

where x_t is the input at time t, h_{t-1} is the hidden state from the previous time step, W and U are weight matrices, b is the bias, and f is an activation function, typically tanh or ReLU.

1.2 Classical Reservoir Computing

Classical Reservoir Computing (CRC) is an algorithm based on RNN architecture, but it significantly reduces the training time. The major problem with RNN and neural networks is that one

has to perform backpropagation, which is very computationally expensive. In backpropagation, the algorithm must compute the weights of all the connections in the hidden and output layers, increasing the computational complexity. Moreover, in recurrent neural networks (RNNs), the vanishing gradient problem arises when attempting to maintain a long-term memory. Reservoir computing architecture significantly reduces these problems without any compromise in performance.

The basic design of an RC algorithm consists of a fixed, randomly initialized reservoir. This reservoir takes the input data and generates a higher dimensional and much more complex representation of the data, which is used to train the target output series. This reservoir is not trained in the training process, i.e., the weights of the connections present are fixed at the beginning and are not trained for different input datasets. This greatly reduces the computational complexity as the reservoir doesn't require training, contrary to the hidden layers in the neural networks. RC only requires training on the output for the target series.

The RC can be classified into two types depending on the nature of the reservoirs.
1. **Virtual reservoir -** Reservoir consists of a large collection of interconnected recurrent units called nodes with random weights.
2. **Physical System reservoir -** A physical system with complex dynamics is taken as a reservoir.

1.2.1 Virtual reservoir

This RC architecture consists of a reservoir with randomly interconnected nodes. The connection weights between nodes are randomly chosen to follow echo state properties, which we stated below—the complete algorithm depicting all the steps shown in Figure 1.2.

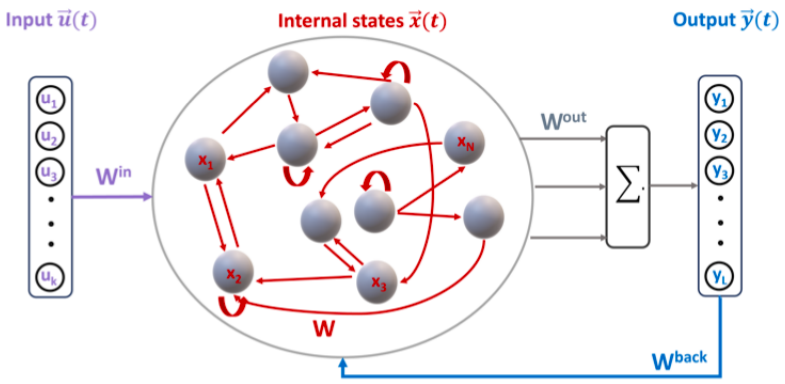

Figure 1.2: Pictorial representation of the architecture of a classical reservoir computing. [8]

Initially, input is fed into the reservoir by multiplying the input weights (W^{in}) to the input vector. Then, the reservoir is evolved using a similar scheme used in RNN; only here, in place of hidden layers, do we have a randomly connected reservoir.

$$x(t) = f[W^{in}\vec{u}(t) + W\vec{x}(t-1) + W^{back}\vec{y}(t-1)]$$

Where W^{in} is the input weight vector, W is the matrix holding all the weights of the connections between the nodes, W^{back} is the weight matrix of the output that is fed back to the reservoir, $\vec{u}(t)$ is the input at time t, $\vec{x}(t-1)$ and $\vec{y}(t-1)$ are the reservoir node and the output generated at a previous time respectively.

The output can be calculated using $y(t) = f^{out}[W^{out}x(t)]$, where x(t) are reservoir nodes generated above and W^{out} is found by performing linear regression concerning the desired target output. The performance of RC depends on the ability of the reservoir network to learn the input-output dynamics. Thus, performance depends on the structure of the chosen. Jaeger studied this in his 2001 paper [20], in which he came up with certain conditions that a reservoir needs to fulfil to give good performance. These conditions are the existence of echo states. The echo states represent the states whose dependence on the initial conditions decays as time increases. For proper functioning of RC, the reservoir nodes should follow echo state properties. We can define echo states by neglecting output feedback based on the following definition.

Definition [20] - Assuming all standard compactness conditions and the network has no output feedback connections. Then, the network has echo states if the network state $\mathbf{x}(n)$ is uniquely determined by any left-infinite input sequence $\mathbf{u}^{-\infty}$. It means that for every input sequence $\ldots, \mathbf{u}(n-1), \mathbf{u}(n) \in U^{-N}$, for all state sequences $\ldots, \mathbf{x}(n-1), \mathbf{x}(n)$ and $\mathbf{x}'(n-1), \mathbf{x}'(n) \in A^{-N}$, where $\mathbf{x}(i) = T(\mathbf{x}(i-1), \mathbf{u}(i))$ and $\mathbf{x}'(i) = T(\mathbf{x}'(i-1), \mathbf{u}(i))$, it holds that $\mathbf{x}(n) = \mathbf{x}'(n)$.

The above definition is difficult to check in practice for a random reservoir. So, it can be transformed into practical rules ensuring that the reservoir follows echo state properties. Assuming that a network has no output feedback connections and has a sigmoid network with output functions $f_i = \tanh$. We can write the conditions for the echo state properties as follows.

- If weight matrix/adjacency matrix(W) has $\sigma_{max} < 1$, where σ_{max} is the largest singular value. Then, the reservoir states form echo states.

- The echo states are not guaranteed if the W has $\lambda_{max} > 1$, where λ_{max} is the eigenvalue of W with the largest absolute value.

1.2.1.1 Demonstration of CRC

To demonstrate CRC, we perform sine series prediction using a random reservoir. In this task, we would predict a sine sequence with many frequencies given the time (t) as input to the reservoir. We first train the reservoir for some training time and predict the series using the optimized weights for some testing time. The output sequence is:

$$y(t) = A_1 \sin(2\pi f_1 t) + A_2 \sin(2\pi f_2 t) + A_3 \sin(2\pi f_3 t)$$

where:

$$\begin{aligned} A_1 &= 1.0, \quad f_1 = 1 \text{ Hz}, \\ A_2 &= 0.7, \quad f_2 = 3 \text{ Hz}, \\ A_3 &= 0.5, \quad f_3 = 5 \text{ Hz}. \end{aligned}$$

The reservoir is constructed such that its entries are chosen from $\{0, 0.4, -0.4\}$ with probabilities $\{0.9875, 0.00625, 0.00625\}$. Thus, the adjacency matrix becomes a sparse matrix and follows the echo state properties, with $\lambda_{max} = 0.012$ and $\sigma_{max} = 0.93$. As the singular value(σ_{max}) is close to 1, this implies that the reservoir has higher memory.

This reservoir is used to perform the learning of sine series, and the results are shown in figure 1.3. In this figure, we can see that the reservoir is able to predict the target sequence with a root mean squared error(RMSE) of 0.0004. It demonstrates that the CRC is able to perform temporal machine-learning tasks.

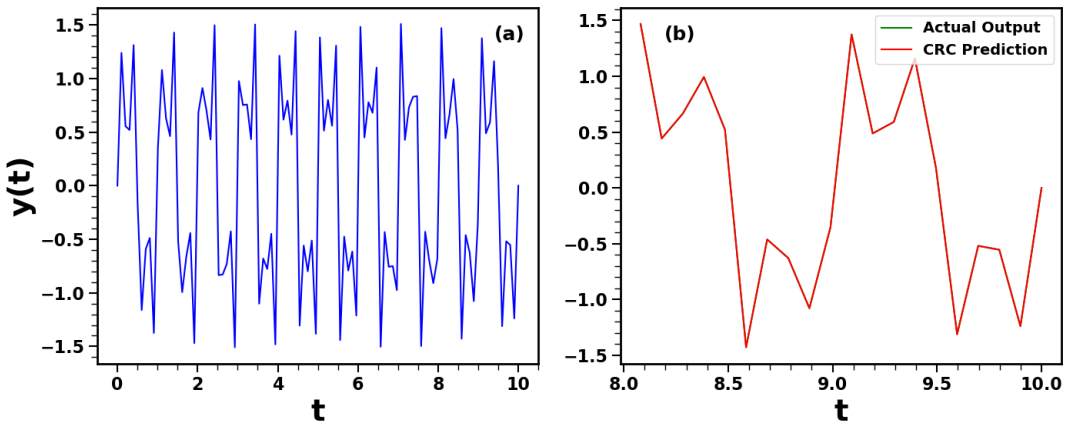

Figure 1.3: Classical reservoir computing is used to predict a sine series. In plot (a), the sine sequence required as output is shown. Whereas in plot (b), the predicted output for the testing part is compared with the actual output. There are 10 nodes in the reservoir, training time = 100 and testing time = 100

1.2.2 Physical Reservoir Computing

Physical Reservoir Computing (PRC) [38] is an extension of Reservoir Computing (RC) that leverages the natural dynamics of physical systems in place of virtual reservoirs. Unlike regular digital implementations of RC, physical reservoir computing utilizes real-world physical systems as reservoirs, taking advantage of their inherent complex dynamics, nonlinearity, and memory effects.

In PRC, the input signal is fed into a physical system that acts as the reservoir, where fixed input weights transform the data into a high-dimensional representation. The reservoir, governed by the intrinsic physical dynamics of the system, processes the input through its nonlinear interactions and memory effects. Unlike traditional machine learning models, the reservoir itself remains unchanged, and only a simple linear readout layer is trained to extract meaningful output. This trained readout maps the reservoir states to the desired predictions or classifications, enabling efficient learning. By leveraging the natural physics of the reservoir, PRC provides a low-power, real-time computational framework for tasks such as time-series forecasting and pattern recognition.

Some examples of PRC are optical reservoirs [6], where light propagation through a nonlinear medium enables complex transformations of input signals, and spintronic reservoirs [39], which use spin waves in magnetic materials to process information. Another example is mechanical reservoir computing, where soft or elastic materials exhibit nonlinear deformations that can be exploited for computation [1]. Quantum reservoir computing, which is the central focus of the study, is also an example of PRC. It leverages quantum systems, such as qubits or trapped ions, to enhance information processing capabilities through quantum superposition and entanglement.

Chapter 2

Quantum Chaos

This study analyses the reservoir computing performance for reservoirs displaying different quantum chaos properties. To explore this approach, it is essential to develop an understanding of quantum chaos and its corresponding measures across different Hamiltonian systems.

Quantum chaos studies the dynamics of quantum systems whose classical counterparts are chaotic [35]. Even though the name states it is the study of chaos, it differs fundamentally from classical chaos. Classical chaos represents the sensitivity of the trajectories of dynamical systems to the initial conditions. However, this effect does not directly translate to the quantum regime, for instance, a slight perturbation to a quantum wavefunction does not result in drastically different evolution due to the unitarity of quantum mechanics. Therefore, quantum chaos studies the departures from the classical signatures in the quantum regime. While originally defined in relation to classically chaotic systems, these concepts can be extended to purely quantum systems as well.

2.1 Models considered for the study

For this study, we selected systems that exhibit clear quantum chaotic signatures, such as random matrices and Ising Hamiltonians. Quantum chaotic systems follow the random matrix theory results due to the BGS (Bohigas-Giannoni-Schmit) conjecture [28]. The BGS conjecture states that the spectral statistics of quantum systems whose classical counterparts exhibit chaotic dynamics follow the predictions of random matrix theory (RMT) corresponding to the system's symmetry class. It forms the basis for classifying quantum systems as chaotic and integrable. The random matrix as Hamiltonian will follow the BGS conjecture by definition, and Ising Hamiltonians follow the BGS conjecture in certain special cases.

2.1.1 Random Matrix

Random matrices form the basis for classifying quantum chaos in Hamiltonian systems. By definition, these are the matrices whose entries are chosen independently and identically (iid) from a distribution. Random Matrix Theory (RMT), developed around the 1990s by Wigner, Dyson, and Mehta, studies the spectral properties of these matrices [24]. It was initially developed to explain the spectral properties of heavy atom's nuclei.

These random matrices can be classified into different classes based on their entries' distribution and their symmetries. Major classes of random matrices that are widely studied are as follows.

1. **Gaussian Ensembles -** They represent hermitian or real symmetric matrices whose entries are drawn from a Gaussian distribution.

2. **Circular Ensembles -** They contain all unitary matrices with complex eigenvalues that lie on the unit circle in the complex plane.

We primarily focus on the Gaussian ensembles, as we consider Hamiltonians that are Hermitian. Depending on the symmetry considered, we can have three types of Gaussian ensembles.

- **Gaussian Unitary Ensemble (GUE) -** This class contains matrices whose probability distribution is invariant under unitary transformations and are used to model Hamiltonians lacking time-reversal symmetry. The probability distribution for its elements is,

$$\rho_{GUE} = \frac{1}{Z_{GUE(n)}} \exp\left(-\frac{n}{2}\operatorname{Tr} H^2\right) \tag{2.1}$$

  defined on the space of $n \times n$ Hermitian matrices $H = (H_{ij})_{i,j=1}^{n}$, and $Z_{GUE(n)} = 2^{n/2}\pi^{n^2/2}$ is a normalization constant

- **Gaussian Orthogonal Ensemble (GOE) -** It includes matrices whose probability distribution is invariant under orthogonal transformation and are used to model Hamiltonians preserving time-reversal symmetry. The probability distribution for its elements is,

$$\rho_{GOE} = \frac{1}{Z_{GOE(n)}} \exp\left(-\frac{n}{4}\operatorname{Tr} H^2\right) \tag{2.2}$$

  defined on the space of $n \times n$ real symmetric matrices $H = (H_{ij})_{i,j=1}^{n}$

- **Gaussian Symplectic Ensemble (GSE) -** This class contains matrices whose probability distribution is invariant under transformation by the symplectic group, and it is used to model Hamiltonians with time-reversal symmetry but no rotational symmetry. The

probability distribution for its elements is,

$$\rho_{GSE} = \frac{1}{Z_{GSE(n)}} \exp\left(-n \operatorname{Tr} H^2\right) \tag{2.3}$$

defined on the space of $n \times n$ quaternionic Hermitian matrices $H = (H_{ij})_{i,j=1}^{n}$. These are not commonly used in quantum chaos studies as the physical systems with quaternionic Hermitian matrices are rare.

2.1.2 Transverse Field Ising Model

Apart from using random matrices as the reservoirs, we also use real physical systems, mainly the Transverse Field Ising Model (TFIM). It describes a system of many interacting spins on a lattice subjected to spin-spin interaction and an interaction with the external transverse magnetic field. The interactions can be nearest-neighbour or all-to-all; depending on these interactions, we can have a system which shows quantum chaos or remains integrable.

Hamiltonian of the system is,

$$H = \sum_{i<j}^{N} J_{ij} \sigma_i^x \sigma_j^x + h \sum_{i=1}^{N} \sigma_i^z \tag{2.4}$$

where J_{ij} is the interaction strength between ith spin and jth spin, h is the transverse magnetic field applied along the z-direction, and σ_i^z and σ_i^x are the Pauli matrices acting on ith site.

This model is integrable when only nearest-neighbour couplings are present between spins for all disorder's strengths. This is because it can be exactly solved using Jordan-Wigner transformations, which map it to a system of free fermions [23]. But if we introduce all-to-all couplings, the model shows chaotic behaviours in certain cases. [14]
When J_{ij} is sampled uniformly from an interval $[J_0 \pm \delta/2]$, and h is constant for all sites, then we can classify the two phases(chaotic and integrable) as follows:

- Integrable regime: If $\frac{J_0}{h} \approx 0$, then the model shows integrable behaviour.
- Chaotic regime: If $\frac{J_0}{h} \gtrapprox 0.5$, then the model shows chaotic behaviour.

The above classification can be carried out using spectral or dynamical statistics. We have used some of them in this study, such as level spacing distribution, Krylov complexity, and inverse participation ratio.

2.2 Quantities to capture quantum chaos

Various tests can be conducted to determine whether the given model exhibits quantum chaos; these include statical and dynamical statistics. All these tests, in some way, use eigenvalues and eigenvectors of the corresponding model to classify Hamiltonian as chaotic and integrable. They employ BGS conjecture for the classifications. Depending on the scenario, using a specific quantity over the other may be beneficial. In this study, we used level spacing distribution (LSD) to classify quantum chaos in TFIM and random matrices. Also, Krylov complexity (KC), out-of-time-order correlators (OTOC), and inverse participation ratio (IPR) are used to capture other aspects of the reservoir.

2.2.1 Level Spacing Distribution

Level Spacing Distribution captures the Hamiltonian's spectral statistics, utilizing its eigenvectors and eigenvalues to classify the model as chaotic or integrable. It uses the histogram of the spacings between the eigenvalues to deduce the nature of the model.

In order to calculate level spacing distribution (LSD) [17], we must first separate the eigenvalues in the appropriate symmetry class to calculate this quantity. If the eigenvalues from different symmetry classes get mixed, then we would not see the appropriate behaviour of spacings.

The random matrices don't have any symmetry class, so all of their eigenvalues together can be used as a whole for the analysis. Meanwhile, in the TFIM, the eigenvalues need to be classified into two subspaces using parity symmetry (Z_2). This symmetry tells whether the state is symmetric or invariant under spin flip. The parity operator corresponding to this symmetry is

$$P = \prod_{i=1}^{n} \sigma_i^z$$

This operator commutes with Hamiltonian, i.e. $[H,P] = 0$, classifying eigenvectors into two subsectors based on the parity. An eigenvector $|\psi\rangle$ belongs to the even parity sector if $\langle\psi|P|\psi\rangle$ is $+1$ and to the odd parity sector if $\langle\psi|P|\psi\rangle$ is -1. This will classify the eigenvalues into appropriate classes, and we can use any one class for the analysis.

After separating eigenvalues, we need to unfold the spectra into appropriate symmetry sectors. This should be performed so that the local density of the eigenvalues becomes one, and we can compare the spacing for different parameters and systems. This is performed by ordering the spectrum in increasing energy values, then separating it into several smaller sets of eigenvalues and dividing each eigenvalue by the mean level spacing of its particular set. Thus, the obtained spectrum will have a local density of 1. Using this unfolded spectrum $\{E_i\}$, we

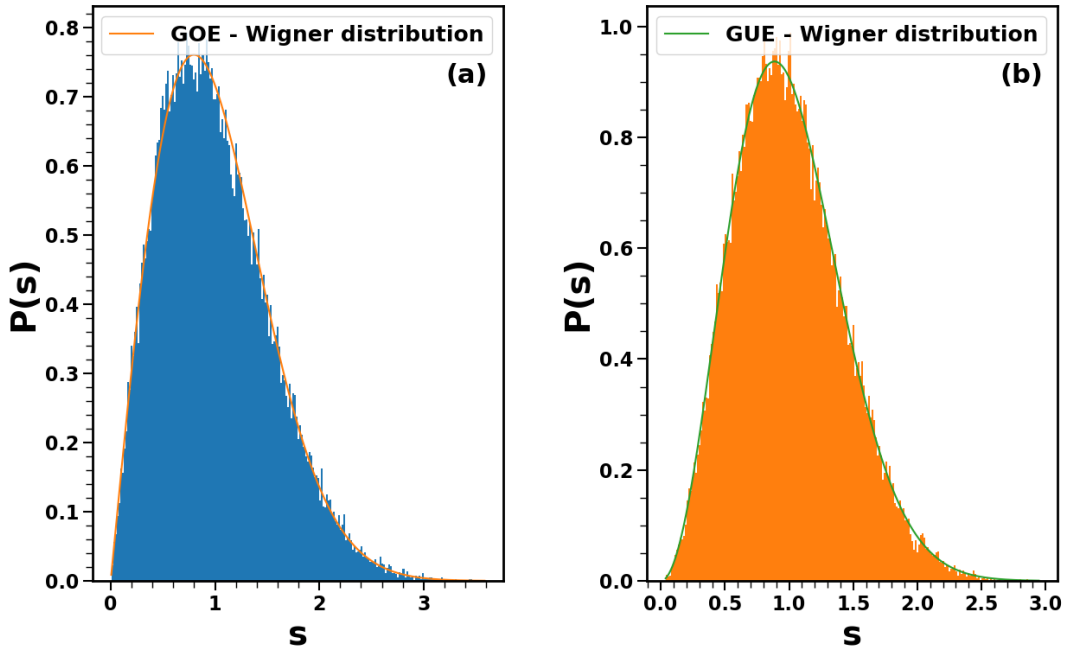

Figure 2.1: Level Spacing distribution of random matrices are plotted. In plots (a) and (b), the LSD of a GOE and GUE matrix is plotted, respectively. The size of the matrix in both cases is 512.

calculate the nearest neighbour spacing between them as $s_i = E_{i+1} - E_i$.

The probability distribution of these spacings classifies the model into chaotic and integrable regimes. For the random matrices, the spacing distribution, also known as the Wigner-Dyson distribution, is,

- GOE :

$$P_{\text{GOE}}(s) = \frac{\pi}{2} s \exp\left(-\frac{\pi}{4} s^2\right) \tag{2.5}$$

- GUE :

$$P_{\text{GUE}}(s) = \frac{32}{\pi^2} s^2 \exp\left(-\frac{4}{\pi} s^2\right) \tag{2.6}$$

Using the BGS conjecture, if Hamiltonian is in a chaotic regime, then the histogram of spacings must follow the spacing distribution for the appropriate class. The TFIM has time-reversal symmetry, so it must obey Gaussian Orthogonal Ensemble (GOE) statistics. In figure 2.1, we can see that the level spacing distribution of random matrices (both GUE and GOE) follows the Wigner distribution.

In figure 2.2, the Level Spacing distribution of TFIM is plotted in both integrable and chaotic limits. We can see that in plot (a), the LSD follows a Poisson distribution, imply-

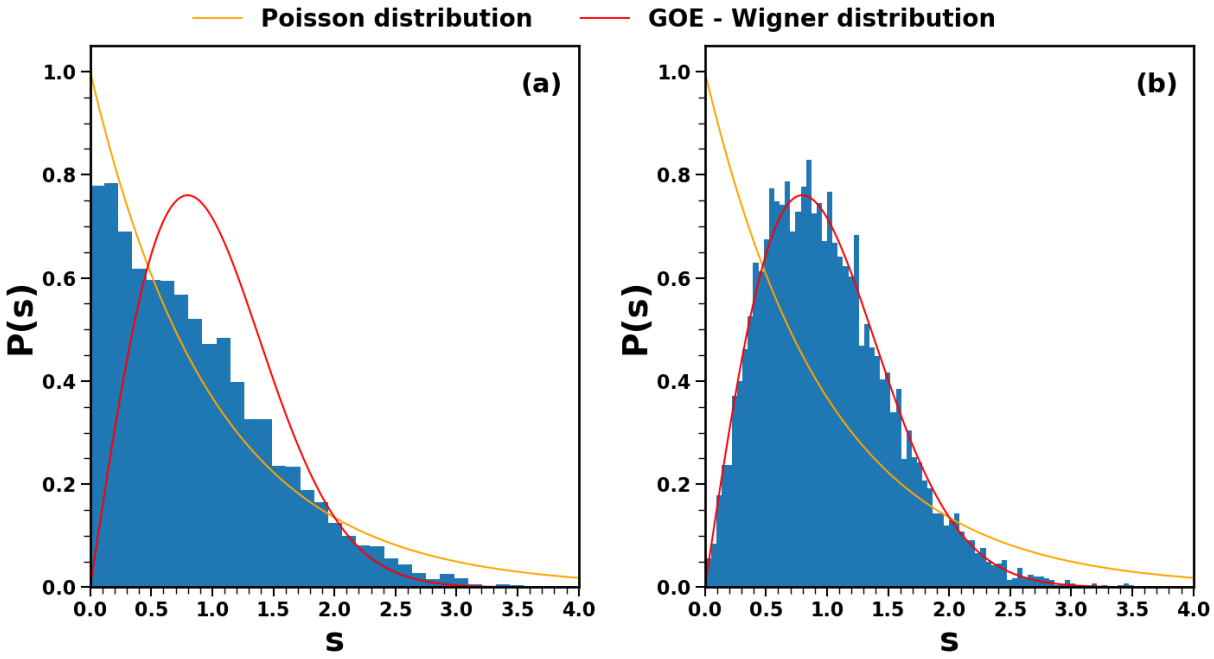

Figure 2.2: Level Spacing distribution of TFIM is plotted in both integrable and chaotic limits. In plot (a), the LSD of integrable TFIM is shown for $J_0 = 1$ and $h = 5$, while in (b), the LSD of chaotic TFIM is shown for $J_0 = 5$ and $h = 5$. In both cases, a 10 qubit reservoir is used, which will be a 1024-dimensional Hamiltonian.

ing that $J_0 = 1$ and $h = 5$ correspond to an integrable regime. Similarly, in plot (b), the LSD follows GOE Wigner-Dyson distribution, indicating $J_0 = 5$ and $h = 5$ correspond to a quantum chaotic regime. Therefore, we observe a transition from integrable to chaos when J_0 goes from 1 to 5, keeping $h = 5$ fixed.

2.2.2 Inverse Participation Ratio

This quantity captures the model's spectral statistics while relying solely on the Hamiltonian's eigenvectors for analysis, capturing their delocalization on a chosen basis [35]. In the chaotic regime, eigenvectors are fully delocalized, whereas in the integrable regime, they remain localized.

Consider an eigenvector $|\psi_i\rangle$ and the basis $|\phi_j\rangle$, such that the expansion of $|\psi_i\rangle$ over the basis is,

$$|\psi_i\rangle = \sum_{j=1}^{N} c_{ij} |\phi_j\rangle$$

then, the inverse participation ratio (IPR) can be calculated as,

$$IPR(|\psi_i\rangle) = \sum_{j=1}^{N} |c_{ij}|^4$$

If IPR is high, then the eigenfunction is localized over a few sites, whereas if IPR is low, the eigenfunction is delocalized over the basis sites.

- **Random Matrices:**
  Random matrices for all classes have delocalized eigenvectors over the basis, resulting in a low IPR. Also, the IPR for all the eigenvectors is the same and equal to $3/N$, where N=basis size. It is shown in figure 2.3, in which nearly all eigenvectors show IPR=$3/N$.

- **Transverse Field Ising Model:**
  If the transverse field Ising model (TFIM) operates in a chaotic regime ($J_0 = 10$), the IPR of its eigenvectors is relatively low. Conversely, in the integrable regime ($J_0 = 1$), the IPR is significantly higher. This trend is illustrated in Figure 2.4, where the IPR in the integrable limit is notably greater than in the chaotic regime. While this pattern holds for the majority of eigenvectors, it is not strictly universal, as some eigenvectors exhibit deviations from the expected behaviour.

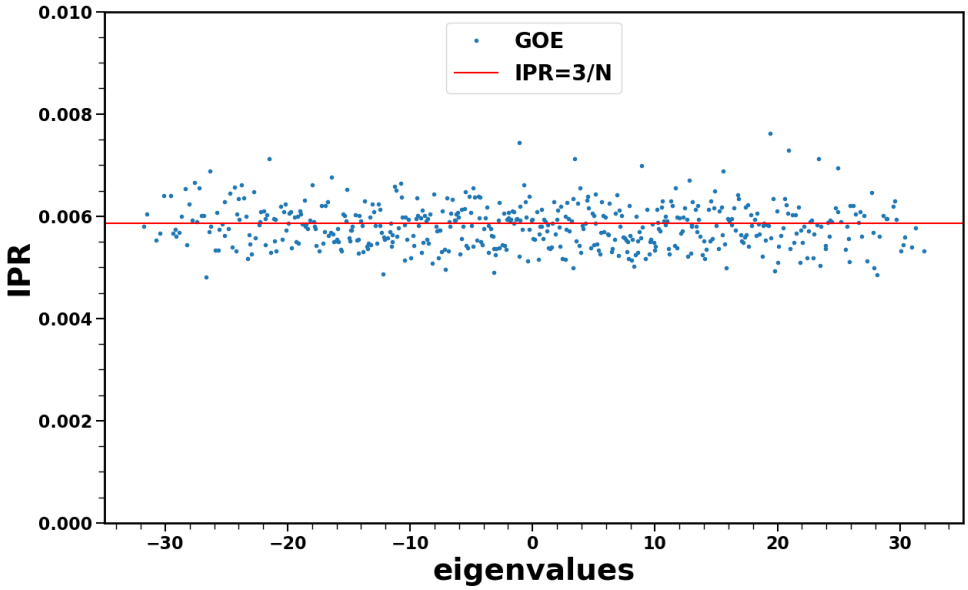

Figure 2.3: IPR of GOE is plotted for various eigenvectors. The red line denotes $3/N$, where $N = 512$, around which most of the IPR of different eigenvectors lies. On the x-axis, the corresponding eigenvalues are plotted.

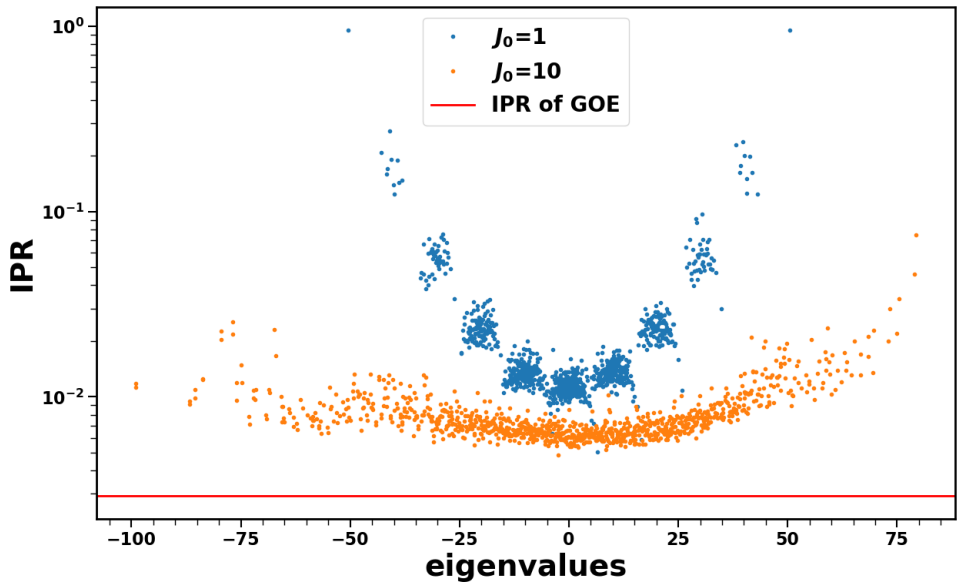

Figure 2.4: IPR of TFIM is plotted for various eigenvectors. The red line denotes $3/N$, where $N = 512$, which is the IPR for random matrices. The blue and orange dots represent IPR for integrable ($J_0 = 1$) and chaotic ($J_0 = 10$) TFIM, respectively. There are 9 qubits in TFIM, which will give a 512-dimensional Hamiltonian, and $h = 5$ is fixed.

2.2.3 Out-Of-Time-Order Correlators

Level Spacing distribution and IPR capture the spectral statistics of Hamiltonian, but they don't capture its dynamical signatures. To capture dynamical evolution, we use other quantities like out-of-time-order correlators (OTOC). It captures information scrambling in many-body quantum systems [12]. When we consider a given Hamiltonian's time evolution, we see that information spreads throughout the system. The nature of this spread of information distinguishes quantum chaotic and integrable behaviour.

Consider two operators $\tilde{W}$ and $\tilde{V}$ at some initial time; the Hamiltonian of the system is H. In the Heisenberg picture, we can write the evolution of operators as $\tilde{W}(t) = U^{\dagger}(t)\tilde{W}U(t)$, where $U(t) = e^{-\iota Ht}$. Then the OTOC is defined as,

$$C_{\tilde{W},\tilde{V}}(t) = \langle [\tilde{W}(t), V]^{\dagger} [\tilde{W}(t), V] \rangle$$

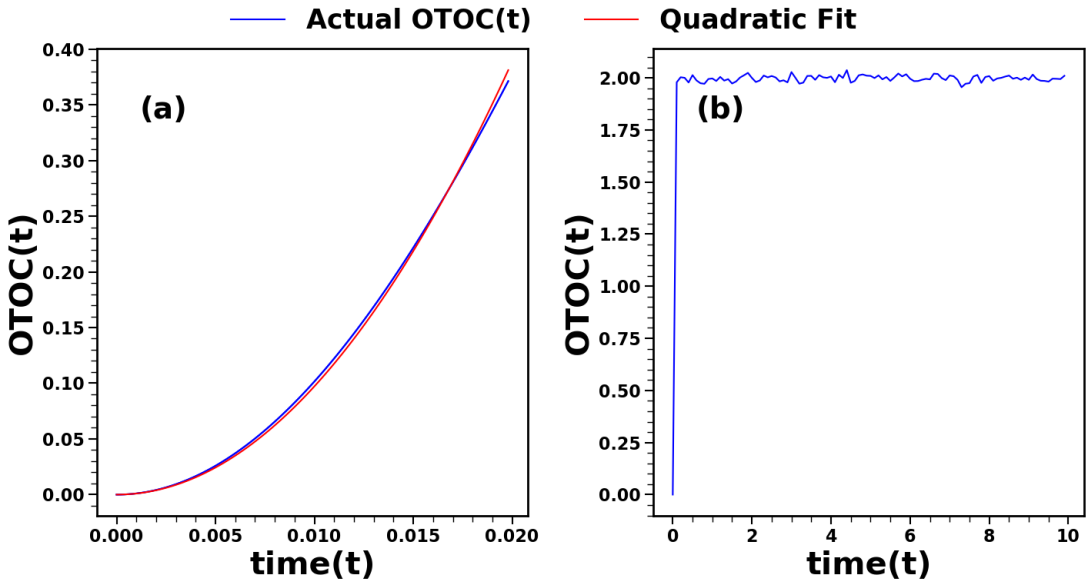

Figure 2.5: OTOC is plotted versus time in both the subplots. In plot (a), short-time growth is shown, which comes out to quadratic, and in plot (b), overall behaviour is shown, which reaches saturation after initial growth. The operator $W = \sigma_1^z$ and $V = \sigma_5^z$ and Hamiltonian (H) is a random GUE matrix. The basis size consists of 8 qubits, which gives a 256-dimensional Hamiltonian. Also, in plot (b), averaged OTOC is calculated to reduce the oscillations seen in the saturation region.

For quantum chaotic systems, OTOC increases exponentially for a very long time and then saturates for a later time. This simply means information spreads very fast and reaches all parts of the system, resulting in saturation for a later time. Meanwhile, we see slow polynomial growth in OTOC in an integrable system, which may sometimes not reach saturation.

We have considered qubit models, and the overall behaviour of OTOC is observed in these models. That is, OTOC initially increases and then saturates and oscillates for a later time. However, the rate of increase is not exponential in general. Depending on the operator considered, we can have polynomial time growth. For example, if we consider a spin chain system and consider σ_i^z, σ_j^z as the two operators $\tilde{w}$ and $\tilde{V}$, then for all random matrices and quantum chaotic TFIM, we see quadratic time growth in OTOC and saturation for a later time, as shown in the figure 2.5.

2.2.4 Krylov Complexity

2.2.4.1 Introduction

OTOC captures operator growth, but it doesn't provide direct information on the dynamical properties of the Hamiltonian (H). If the Hamiltonian's dynamical signatures are needed explicitly, then using other quantities like Krylov complexity would be useful. Krylov complexity (KC) directly captures the Hamiltonian's dynamical properties without the use of two local operators. It reveals the nature of information scrambling in the system for both quantum chaotic and integrable evolution.

2.2.4.2 Calculation of Krylov Complexity

Krylov complexity tracks the spread of an initially chosen operator in the Krylov basis, helping to capture information scrambling [19]. To calculate KC, the Krylov basis for the Hamiltonian is first determined, and then the time evolution of the operator is expanded in this basis.

To construct the Krylov basis, we use the Lanczos algorithm [31]. We start with a Hamiltonian (H) and initial operator ($\mathscr{O}$), such that the $\mathscr{O}(t) = U^\dagger \mathscr{O} U$. We also define the Liouvillian superoperator ($\mathscr{L}(\mathscr{O})$) as $[H, \mathscr{O}]$ and the inner product between operators as $Tr[\mathscr{O}_1^\dagger \mathscr{O}_2]$. Then the Lanczos algorithm is as follows;

1. $|\mathscr{O}_0\rangle = \frac{1}{\sqrt{\langle\mathscr{O}|\mathscr{O}\rangle}}|\mathscr{O}\rangle$.
2. For $n \geq 1$: Compute $|\mathscr{A}_n\rangle = \mathscr{L}|\mathscr{O}_{n-1}\rangle$.
3. Re-orthogonalize $|\mathscr{A}_n\rangle$ explicitly with respect to all previous Krylov elements:

$$|\mathscr{A}_n\rangle \longmapsto |\mathscr{A}_n\rangle - \sum_{m=0}^{n-1} |\mathscr{O}_m\rangle\langle\mathscr{O}_m|\mathscr{A}_n\rangle.$$

4. Repeat step 3.
5. Set $b_n = \sqrt{\langle\mathscr{A}_n|\mathscr{A}_n\rangle}$.

6. If $b_n = 0$, stop; otherwise, set $|\mathscr{O}_n\rangle = \frac{1}{b_n}|\mathscr{A}_n\rangle$ and go to step 2.

If the Krylov dimension ($\mathscr{K}_{\mathscr{O}}$) is finite, then the algorithm stops with $b_{\mathscr{K}_{\mathscr{O}}} = 0$. This algorithm gives us the Krylov basis $\{\mathscr{O}_n\}_{n=0}^{\mathscr{K}_{\mathscr{O}}-1}$ and Lanczos coefficients b_n. Now we expand time evolved operator ($\mathscr{O}(t)$) in terms of Krylov basis;

$$\mathscr{O}(t) = U^{\dagger}\mathscr{O}U = \sum_{n=0}^{\mathscr{K}_{\mathscr{O}}-1} \imath^n \phi_n(t)\mathscr{O}_n$$

Then, the Krylov complexity (KC) is defined as;

$$C_k(t) = \sum_{n=0}^{\mathscr{K}_{\mathscr{O}}-1} n|\phi_n(t)|^2$$

2.2.4.3 Behaviour of $C_k(t)$ in random matrices

In random matrices, Krylov complexity grows exponentially at short times and saturates at a value $D/2$ for later times, where D is the dimension of the Hilbert space. It is shown in plot (b) of figure 2.6 where the Hamiltonian (H) and operator ($\mathscr{O}$) are both GUE matrices. This means that as the operator evolves, it explores the entire Krylov basis, implying that the information has scrambled with the systems very fast. In the integrable systems, although there is no universal pattern for the behaviour of the Krylov complexity, we can characterize them with slow growth and no saturation, even for a longer time.

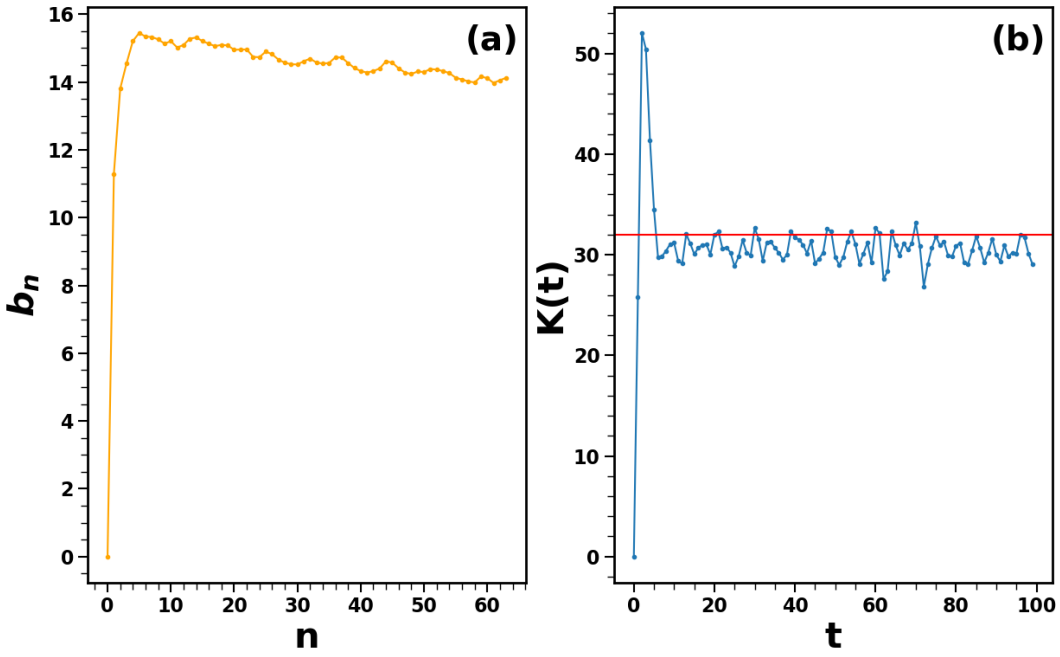

Figure 2.6: Krylov complexity and Lanczos coefficients are plotted for GUE matrices. In plot (a), the growth of Lanczos coefficients is shown versus order n, which increases initially and reaches saturation at some finite value. In plot (b), Krylov complexity versus time is plotted, which shows the expected behaviour of initial rise and saturation at $D/2$, where $D = 64$. The operator ($\mathscr{O}$) and Hamiltonian (H) are both random GUE matrices. The basis size consists of 6 qubits, which gives a 64-dimensional H and $\mathscr{O}$.

Chapter 3

Quantum Reservoir Computing

Quantum reservoir computing (QRC) is a machine learning technique used to solve temporal and non-temporal tasks. The idea is based on the classical reservoir computing (CRC) framework, but the reservoirs considered are quantum systems. QRC is proposed to take advantage of the high dimensionality of Hilbert spaces to efficiently separate input and generate its many complex representations. It is a relatively new field, with the first successful theoretical proposition shown in 2017 by Kohei Nakajima and Keisuke Fujii using a transverse field Ising model (TFIM) [11]. They went on to give the first experimental demonstration of the idea in 2018 by using a nuclear spin ensemble in a molecular solid [26]. There were several other demonstrations of QRC after their study performed on a variety of quantum reservoirs, both numerically and experimentally [5] [10] [15] [30]. In these studies, several instances were shown in which QRC performs better for certain tasks than CRC, considering similar reservoirs in both QRC and CRC. The generality of this quantum advantage considering quantum state tomography and measurement complexity is still not understood.

The basic technique of QRC is shown in figure 3.1. In QRC, input is fed into the reservoir using various encoding techniques. The input can be classical or quantum in nature. The reservoir evolves according to quantum mechanical evolution, exploring the high-dimensional Hilbert space to create complex representations of the input. After letting the reservoir evolve for a sufficient time, we take measurements using some hermitian operators. These measurement outcomes will become our output nodes, which will be used for training the target output series.

3.1 Method of implementation

We have extended the work done by Kohei Nakajima and Keisuke Fujii in 2017 using TFIM to random matrices and use these reservoirs to understand the details of QRC and the source

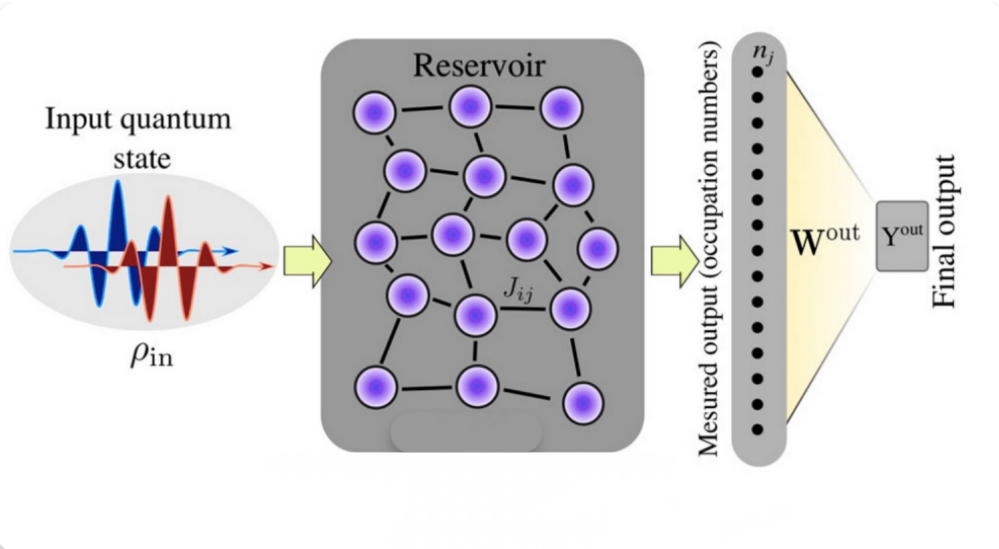

Figure 3.1: Pictorial representation of generic QRC technique. [16]

of quantum advantage. This study is performed on qubit-based systems using a particular encoding scheme. As one changes the quantum reservoir from qubit-based to different platforms, then the results of our work are also expected to change. The performance of the QRC obtained is also dependent on the input encoding scheme, and as it changes, the performance also changes. The details of the QRC scheme used for the complete study are explained below.

3.1.1 Description of quantum systems and dynamics

The quantum systems considered to perform QRC are qubit-based. The qubit is a two-level quantum system which can be written in terms of $|0\rangle, |1\rangle$ as a vector in Hilbert space. If we take N qubits connected together as the reservoir, then the size of the Hilbert space of systems is 2^N, and the general state $|\psi\rangle$ lies in this 2^N-dimensional space. Generally, we don't use a pure state for the analysis; we use a mixed state. These mixed states can be written in terms of a density matrix ρ, which lives in $2^N \times 2^N$-dimensional space and can be calculated as,

$$\rho = \sum_{k=0}^{M} p_k |\psi_k\rangle \langle\psi_k|$$

where, p_k is the probability to choose $|\psi_k\rangle^{th}$ state. The quantum mechanical evolution due to Hamiltonian (H) for a time interval τ of the density matrix is,

$$\rho(t+\tau) = e^{-\iota H\tau}\rho(t)e^{\iota H\tau}$$

3.1.2 Input encoding

In order to exploit quantum dynamics for information processing, we need to introduce an encoding procedure to encode input to the qubits of the reservoir. This input encoding procedure should also be consistent with the echo-state properties needed for the reservoir computing to work.

Let $\{x_k\}_{k=0}^{M}$ and $\{y_k\}_{k=0}^{M}$ be the input and target output sequence respectively. Using QRC, we want to predict y_k by feeding x_k to the reservoir. The x_k can be a discrete or continuous variable with a range that can be anything. As the probability cannot be less than 0 and greater than 1, so the input sequence x_k is encoded into s_k, which is between 0 and 1. This can be done simply by using

$$s_k = \frac{x_k - min\{\{x_k\}_{k=0}^{M}\}}{max\{\{x_k\}_{k=0}^{M}\} - min\{\{x_k\}_{k=0}^{M}\}}$$

This rescaled input s_k is encoded into the density matrix(ρ_{s_k}), which is fed onto the first qubit after replacing its present state. This operation can be written in terms of a CPTP map as,

$$\rho \rightarrow \rho_{s_k} \otimes Tr_1[\rho]$$

where,

$$\rho_{s_k} = |\psi_{s_k}\rangle \langle \psi_{s_k}|$$

$$|\psi_{s_k}\rangle = \sqrt{1-s_k}\,|0\rangle + \sqrt{s_k}\,|1\rangle$$

and Tr_1 denotes the partial trace with respect to the first qubit. After injecting the k^{th} input, the system evolves under the Hamiltonian (H) for time (τ) according to the density matrix evolution. This evolved density matrix is used for measurement and generating output nodes.

This input encoding ensures that the echo state property is preserved, enabling the feasibility of reservoir computing. It is due to the trace operators that cause the system to gradually lose its initial memory after some time steps. The validity of this property can be verified by implementing the aforementioned input encoding scheme on two distinct density matrices. These density matrices, denoted as $\rho_A(0)$ and $\rho_B(0)$, are randomly initialized at the initial time ($t = 0$) and subsequently evolved over time (t). For the fading memory property to hold, $||\rho_A(t) - \rho_B(t)||$ should go to 0 as the reservoir evolves. This is shown for both GUE and TFIM reservoirs in figure 3.2, as the $||\rho_A(t) - \rho_B(t)||$ go to 0 for large time (t).

3.1.3 Measurement

Measurements of density matrices in quantum systems are performed using trace operations. We choose some projective operators like P_i, which satisfies, completeness ($\sum_i P_i = 1$) and

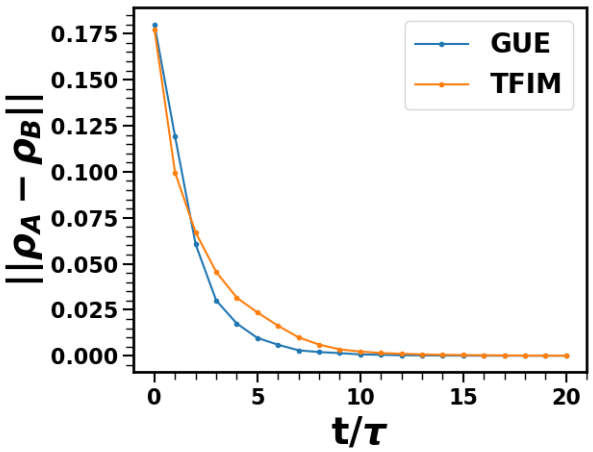

Figure 3.2: Fading memory of the QRC scheme is shown for both GUE and TFIM reservoirs. The orange cure is for the TFIM reservoir, and the blue curve denotes GUE as a reservoir. There are 6 qubits which give a 64-dimensional Hamiltonian and $\tau = 1$.

orthogonality ($P_iP_j = \delta_{ij}P_i$). Trace operations in measurement state that the probability of obtaining outcome i is $\mathrm{Tr}[P_i\rho]$, and the state after measurement becomes,

$$\rho \to \frac{P_i\rho P_i}{Tr[P_i\rho]}$$

After the input encoding and evolution, we get an evolved density matrix $\rho(t+\tau)$, and we perform measurements onto this density matrix. There can be 4^N possible measurement operators in this scenario; these are $\{1,X,Y,Z\}^{\otimes N}$. Choosing Z_i to be the measurement operator, which denotes the Pauli operator Z acting on each i^{th} qubit. Measurement along these operators will provide output nodes to be of the form

$$O_i(t) = Tr[Z_i\rho(t)]$$

These output nodes will used for the learning in the next step. The remaining operators form the hidden nodes ($4^N - N$), as they are not used for the learning.

We can increase the output nodes by taking some more nodes from the hidden nodes, or we can perform time multiplexing. The time multiplexing means that we let the reservoir evolve for a time period τ, but we take measurement V times in that time period. That means we perform all N measurements at regular time intervals (τ/V) within the time period τ. This process will increase the computational nodes from N to NV.

3.1.4 Training readout weights

The output computational nodes generated after measurement are used to predict the target sequence $\{y_k\}_{k=0}^{M}$, which is expected to be some function of inputs fed to the reservoir $f\left(\{s_k\}_{k=0}^{M}\right)$. The output computational nodes (total number = NV) are stacked onto an array (X), which is trained via linear regression or ridge regression to generate the best prediction of the target sequence ($\hat{y_k}$).

- Linear Regression - Loss function to be minimized is,

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

  Weights can be calculated using Normal equation

$$w=(X^TX)^{-1}X^Ty$$

- Ridge regression - Loss function to be minimized is,

$$\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2+\lambda\sum w_i^2$$

  Then the normal equation to calculate weights is modified to

$$w=(X^TX+\lambda I)^{-1}X^Ty$$

Using above regression techniques, we would generate predictions ($\hat{y}$) which can be written as $\hat{y}=\sum_{i=1}^{NV}O_iw_i+\beta$. We want these predictions to come very close to the true target sequence y_k. Therefore, we study the difference between these two sequences. Some of the error measures used throughout the thesis are,

1. Root mean squared error (RMSE) = $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2}$

   RMSE gives the error between the predicted sequence and the target sequence. For the best prediction, RMSE = 0, and as RMSE increases, the fit becomes worse.

2. Normalized Root Mean Square Error (NRMSE) = $\frac{RMSE}{y_{max}-y_{min}}$

   NRMSE normalizes the RMSE calculated above with respect to the scale of the target output series, and it always lies between 0 and 1. It is used to compare the error in prediction between two target sequences of different scales. As in the RMSE, for the best prediction, NRMSE = 0, and as NRMSE increases, the fit becomes worse.

3. Pearson correlation coefficient (Capacity) = $\frac{\sum_{i=1}^{N}(y_i-\bar{y})(\hat{y}_i-\bar{\hat{y}})}{\sqrt{\sum_{i=1}^{N}(y_i-\bar{y})^2}\sqrt{\sum_{i=1}^{N}(\hat{y}_i-\bar{\hat{y}})^2}}$

Capacity measures the linear relationship between the predicted sequence and the target sequence. It lies between -1 and 1. It returns +1 if there is a linear correlation between the two series and -1 if there is an anti-correlation. Capacity becomes 0 when there is no correlation between the two series.

3.2 Optimization Hyperparameters

There are several parameters which, when changed, affect the quality of the fit between the predicted sequence and the target sequence. These need to be optimized to achieve the best performance. Optimization is not a non-universal procedure, as we need different characteristics to be present in the reservoir for different tasks. For example, there is always a tradeoff between memory and the non-linearity of the reservoir, so if a task requires more memory, we should operate the reservoir in an integrable regime.

In most of the tasks considered, we first train the target series up to some training time using either linear or ridge regression. Then, using the optimized weights, we predict the sequence for some later time (testing time). This testing time prediction is used to calculate the testing error, and it tells whether the reservoir has learnt the sequence.

- **Three major time scales of learning -**

    1. Transient time - It is the time up to which we let the reservoir evolve and don't perform any training. This is done to forget any initial input (time=0) dependence that might exist in the output nodes. This time needs to be large, but increasing it too much will increase the computational time.
    2. Training time - This is the time for training. In theory, this time should be greater than the number of computational nodes one has taken. But increasing it very much will also destroy the quality of the fit. A tradeoff exists here that needs to be optimized.
    3. Testing time - This timescale does not have a role in optimization as it uses the optimized weights for the prediction. The error calculated for this time tells whether the reservoir has learnt the target output series or not. It should be large for better learning, but in principle, after some time, the quality of prediction fades. So, this timescale is initialized according to the requirements, and the remaining optimizing parameters are optimized to obtain the best performance in this selected timeframe.

- **Computational nodes** (N,V) **-** By changing the number of computational nodes, we change the dimension of the feature space on which regression will performed. Thus, increasing the computational nodes will improve the fit. However, making it large will lead to overfitting, as one would be using more parameters in feature space than the actual

number present in the target series, and also increases the computational time. This also results in a tradeoff which needs to be optimized.

- **Hamiltonian parameters -** The Hamiltonian we chose can be present in an integrable and chaotic regime depending on the parameters. For the demonstration, we have chosen Hamiltonian (TFIM) and random matrices to be in a chaotic regime, but we will study this later as it is very task-dependent. That is, depending on the task, the Hamiltonian can be in an integrable regime, chaotic regime or anything in between.

- **Evolution time (τ) -** Evolution time (τ) represents the time for which the evolution is performed between the two successive input encodings. It should be large, enabling the reservoir to explore the entire Hilbert space; however, then the computational time of the physical reservoir will also increase. As the case for Hamiltonian parameters, its optimization is also dependent on tasks to be performed, and no general statement can be made about high performance for all tasks.

3.3 Demonstration of QRC for some temporal learning tasks

We will demonstrate the machine learning potential of quantum reservoir computing on several temporal tasks using the TIFM as a reservoir. The TFIM is operated within a chaotic regime. This is ensured by choosing $h = 5$ and $J_0 = 5$ such that J_{ij} is sampled from -J_0 to J_0. The tasks chosen to demonstrate QRC are timer tasks and some time series prediction tasks.

3.3.1 Timer task

In this task, the input is flipped from 0 to 1 at a certain time in step k', and the output should give 1 when the input is flipped after some delay (τ_{delay}). This task demonstrates that memory up to τ_{delay} is retained in the reservoir dynamics. The input and the output of the task are as follows

$$Input = \begin{cases} x_k = 0, & \text{if } k < k' \\ x_k = 1, & \text{if } k \geq k' \end{cases}$$

$$Ouptut = \begin{cases} y_k = 0, & \text{if } k \neq k' + \tau_{delay} \\ y_k = 1, & \text{if } k = k' + \tau_{delay} \end{cases}$$

We perform QRC on the above input and output sequence, and the result is shown in 3.3. There are two components of a QRC task: training and testing. In the training part, we find the optimized weights (w_i), and in the testing part, we use these optimized weights to make the prediction. The error of fit in the testing part is then taken to make a conclusion as to whether QRC is able to implement the task.

For the training part of this timer task, we fix k'=500 and τ_{delay} = 20 to generate a particular input and output sequence, which can be seen in the plot (a) of figure 3.3. After letting the reservoir evolve for a transient period of 400-time steps, we perform the training of QRC to generate optimized weights (w_i). To test the learning, we change the k' to 700-time steps and keep τ_{delay} = 20 and let the reservoir evolve similarly as in the training process.

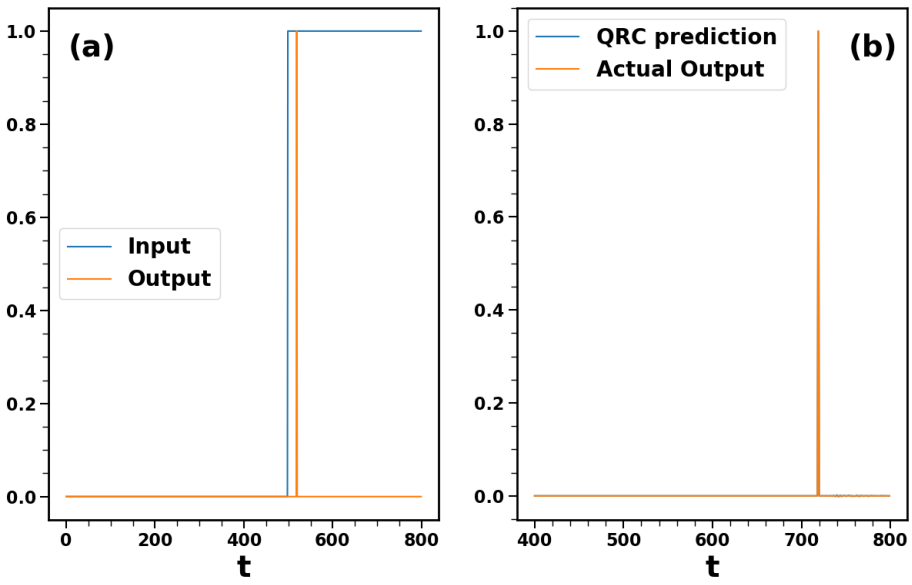

Figure 3.3: Performance of the timer task using QRC is shown. In plot (a), the input and output sequences are shown for the training part of the task. Meanwhile, in (b), the predicted output for the testing part is compared with the actual output. Optimizing parameters are $N = 6, V = 5$, evolution time (τ) = 1, transient time, training time, and testing time, are all 400-time steps.

Then use optimized weights (w_i) to generate predictions. In the plot (b), the predicted output is plotted versus the actual output. We can see that both the curves coincide, indicating that the reservoir is able to predict the output sequence. This is also evident by the RMSE of the fit for the testing part, which is 0.00013.

3.3.2 Time series prediction tasks

One of the most important applications of quantum reservoir computing is its ability to predict complex time series. In this section, we will demonstrate this application of QRC by predicting an output time series that is a time-delayed and nonlinear function of the input series. The nonlinear nature of the time series makes it difficult to predict directly, but the nonlinear and memory properties of the quantum reservoir enable QRC to make accurate predictions. We have chosen two different time series, each varying in nonlinearity, to demonstrate the reservoir's ability to capture both nonlinearity and memory effects. Their analysis is presented

below.

3.3.2.1 NARMA time series prediction

The Nonlinear AutoRegressive Moving Average (NARMA) series [11] is a nonlinear stochastic series specifically designed to evaluate the predictive capabilities of machine learning tasks. This time series gives an output sequence that has long-term memory effects and nonlinear dependencies on the input sequence. The first NARMA series is the following second-order nonlinear dynamical equation:

$$y_{k+1} = 0.4y_k + 0.4y_k y_{k-1} + 0.6x_k^3 + 0.1$$

The second NARMA series of order n is the solution of the following nonlinear dynamical system,

$$y_{k+1} = \alpha y_k + \beta y_k \left(\sum_{j=0}^{n-1} y_{k-j} \right) + \gamma x_{k-n+1} x_k + \delta$$

where $(\alpha, \beta, \gamma, \delta)$ = (0.3,0.05,1.5,0.1)

We use the second NARMA series of order n for the prediction due to its long memory and high nonlinearity. The input (x_k) is taken to be a superimposed sine sequence to stabilize the NARMA system. It can be written as,

$$x_k = 0.1 \left[sin\left(\frac{2\pi\alpha k}{T}\right) sin\left(\frac{2\pi\beta k}{T}\right) sin\left(\frac{2\pi\gamma k}{T}\right) \right]$$

where (α, β, γ) = (2.11,3.73,4.11) and T = 100.

To perform QRC, we first let the reservoir evolve for a transient time to wash out initial time dependencies. After this transient evolution, we train the reservoir to predict the NARMA-10 sequence for some training time. This step will generate the optimized weights that will be used to predict the series at a later time.

The performance of this task is shown in figure 3.4. It shows that QRC using TFIM reservoir is able to predict NARMA-10 time series as the predicted curve closely aligns with the actual output curve in the plot (b) of figure 3.4. This is further supported by the NRMSE, which is 0.0145, indicating a high degree of accuracy in the fit.

3.3.2.2 Lorenz series prediction

Lorenz's system is an example of classical chaos, which was introduced by Edward Lorenz in 1963 to model atmospheric convection [36]. It consists of three nonlinear differential equa-

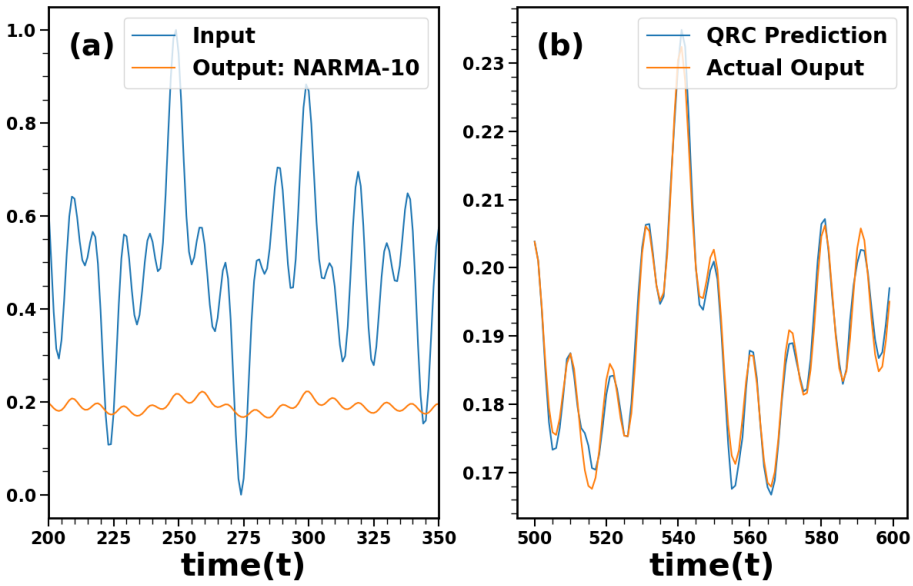

Figure 3.4: Performance of the NARMA-10 series prediction using QRC is shown. In plot (a), the input (scaled between 0 and 1) and target output sequences of NARMA-10 systems are shown for a small duration. Meanwhile, in (b), the predicted output for the testing time is compared with the actual output. Optimizing parameters are $N = 6, V = 5, \tau = 1$, transient time = 200, training time = 200, and testing time = 100.

tions whose numerical solutions show chaotic behaviour. That is, the solution has high initial condition dependence; therefore, when the initial condition is perturbed slightly, the evolved trajectory becomes completely different.

The nonlinear differential equations of the Lorenz system are -

$$\frac{dx}{dt} = \sigma(x - z)$$

$$\frac{dy}{dt} = x(\rho - z) - y$$

$$\frac{dz}{dt} = xy - \beta z$$

where σ, ρ, β control the behaviour of the system between integrable and chaotic regimes. The system exhibits deterministic chaos when $\rho = 28$, $\sigma = 10$, and $\beta = 8/3$. This set of parameters is used throughout the study whenever it is mentioned that the Lorenz system shows chaotic dynamics.

Solving the above nonlinear differential equations in the chaotic regime using numerical integration, we get $x(t), y(t)$ and $z(t)$. These 3 solutions will exhibit chaotic behaviour, and the task we are considering is to predict $z(t)$ using $x(t)$ as input. Due to the chaotic nature of

the time series, there are no analytic relations between $x(t)$ and $z(t)$. This increases the task's difficulty and examines the amount of nonlinearities present in the reservoir.

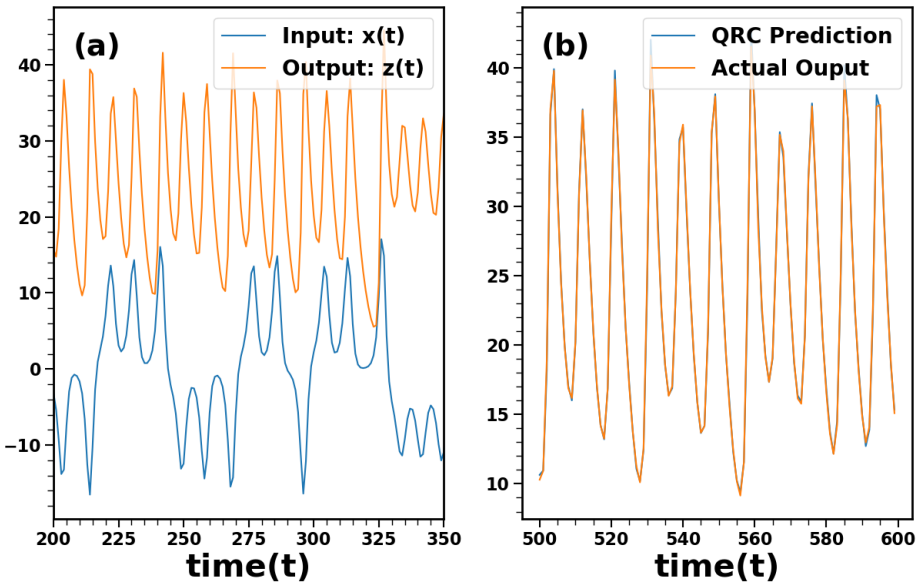

Figure 3.5: Performance of the Lorenz Z series prediction using QRC. In plot (a), the input ($x(t)$) and output sequences ($z(t)$) of Lorenz systems are shown for a small time duration. Meanwhile, in (b), the predicted output for the testing time is compared with the actual output. Optimizing parameters are $N = 7, V = 5, \tau = 2$, transient time = 200, training time = 300, and testing time = 100.

Figure 3.5 shows the obtained performance by predicting the Lorenz $z(t)$ sequence using QRC. The procedure is the same as that used in the NARMA-10 prediction task; only optimizing parameters are changed to increase performance. In plot (a), the input and output sequence is shown, whereas in plot (b), it can be seen that the predicted output matches the actual output, implying that the QRC is able to predict the Lorenz $z(t)$ accurately. This can also be inferred from the NRMSE error of the testing fit, which is 0.0158.

3.4 Emerging Nonlinearity from a linear system

One major question that arises in quantum reservoir computing is how quantum mechanical evolution, being linear, is able to generate non-linearities. It is a very important question and gives important insights into QRC. The linearity of quantum mechanical evolution can be checked as,

Let $\rho = a\rho_1 + b\rho_2$, where ρ_1 and ρ_2 are two different density matrices, then the quantum evolution is, $\rho(t+\tau) = e^{-\iota H\tau}\rho(t)e^{\iota H\tau}$

It becomes, $\rho(t+\tau) = e^{-\iota H\tau}(a\rho_1(t) + b\rho_2(t))e^{\iota H\tau}$

$\Rightarrow \rho(t+\tau) = a(e^{-\iota H\tau}\rho_1(t)e^{\iota H\tau}) + b(e^{-\iota H\tau}\rho_2(t)e^{\iota H\tau})$

So, it is clear that quantum mechanical evolution is linear, and it doesn't generate any non-linearities. Also, in the measurement, nonlinearities do not arise due to the application of trace operators, which produce outputs as linear combinations of the density matrix elements.

Therefore, the nonlinearities are only generated at the input encoding step. This is associated with the tensor product taken at the input encoding step.
Input encoding is , $\rho \rightarrow \rho_{s_k} \otimes Tr_1[\rho]$

$$\Rightarrow \rho \rightarrow \begin{pmatrix} (1-x_k)\cdot Tr_1(\rho) & 0 \\ 0 & x_k \cdot Tr_1(\rho) \end{pmatrix}$$

We can see that $x_k \cdot Tr_1(\rho)$ is nonlinear and creates all the non-linearities required in the QRC process.

In general, the nonlinearities can be classified into two types for a given input sequence x(t).

1. **Static nonlinearities:** These are nonlinearities that are instantaneous in input and do not contain time-delayed terms. For example, when the input is $x(t)$, then the static nonlinearities can be $x(t-1)^2$, $x(t-3)^4$, etc.

2. **Dynamic nonlinearities:** These nonlinearities contain time-delayed terms. For example, when the input is $x(t)$, then the dynamic nonlinearities can be $x(t-2)\cdot x(t-4)$, $x(t-1)\cdot x(t-3)\cdot x(t-2)$, etc.

Also, only dynamic nonlinearities will be created at the input encoding due to $x_k \cdot Tr_1(\rho)$. Therefore, even though quantum mechanical evolution is linear, QRC is able to create many non-linear combinations of input.

Chapter 4

Quantum Reservoir Computing using random matrices

In the previous section, the potential of QRC in performing temporal tasks is demonstrated. It was shown by presenting a scheme to implement QRC and using a TFIM as the reservoir. TFIM is used in the quantum chaotic regime, and it was able to perform well in all the considered tasks.

There is nothing unique in using the TFIM in a quantum chaotic regime, and, in principle, one can use any model that shows quantum chaos. So, to explore the limits of performance in a more general and system-independent way, we would perform QRC using random matrices. In quantum chaos, the model reproduces the results of random matrices; it would be more general to use random matrices directly as the Hamiltonian and see if there exists any limit on the performance.

4.1 Methodology

The same QRC implementation scheme is used, as demonstrated in Chapter 3, with only modifications in the reservoir and keeping the system qubit-based, consisting of N qubits. The reservoir here is any random matrix (GUE, GOE) that makes the Hamiltonian a random matrix instead of TFIM, as considered in Chapter 3.

Three different types of random matrices are considered to examine whether there is a dependence on the type of random matrix used. Two of these denote Hamiltonian, which is sampled from the Gaussian Orthogonal Ensemble (GOE) and the Gaussian Unitary Ensemble (GUE), while the third consists of unitary matrices sampled from the Haar measure. We chose the same time series prediction tasks (NARMA, Lorenz series) as the benchmark to demon-

strate that we are able to implement QRC using random matrices and compared them with respect to TFIM.

4.2 Time series prediction tasks

For the demonstration of QRC performance using random matrices, the GUE matrix is used as the Hamiltonian. It can be generated by sampling the non-diagonal entries of the matrix from $\mathcal{N}(0, 1/2)$ and diagonal entries from $\mathcal{N}(0, 1)$.

1. NARMA-10 prediction - The second NARMA series of order 10, with the input following a superimposed sine sequence, is used as the task. It is the same as what is considered in Chapter 3. The result is plotted in figure 4.1, which clearly shows that QRC, using the GUE reservoir, is able to learn the NARMA-10 series. The NRMSE error of the testing fit is 0.015.

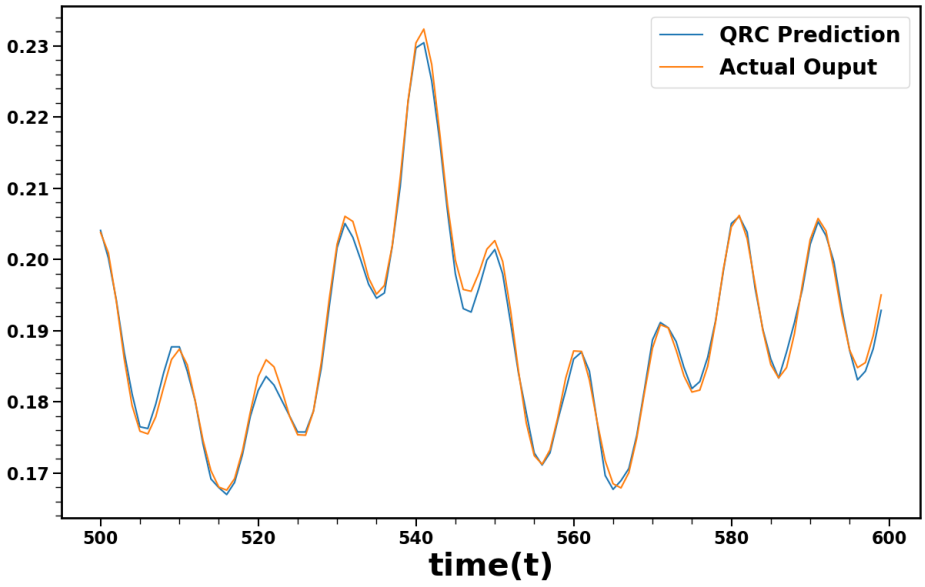

Figure 4.1: Performance of the NARMA-10 prediction using GUE as a reservoir is shown. The predicted output (blue curve) for the testing time is compared with the actual output (orange curve) in the plot. Optimizing parameters are $N = 7, V = 10, \tau = 5$, transient time = 100, training time = 400, and testing time = 100.

2. Lorenz series prediction - The Lorenz sequence $x(t)$ is taken as input, and Lorenz $z(t)$ is required as the output, as demonstrated in Chapter 3. The learning was performed, and the result is plotted in figure 4.2. It also shows that the reservoir is able to learn the chaotic time series, as the predicted output overlaps completely with the actual output. The NRMSE error of the testing fit is 0.016.

3. Memory Capacity - This task is used to evaluate a system's ability to retain and process past inputs over time. A reservoir with high memory capacity can recall past inputs over an extended duration. It involves training the reservoir to reconstruct past inputs from its current state. Thus, the input and output sequence becomes,

$$Input = \Big\{ x_k = U(0,1), \quad \forall k$$

$$Ouptut = \begin{cases} x_k = Input[k - \tau_{delay}], & \text{if } k > \tau_{delay} \\ x_k = 0, & \forall k \leq \tau_{delay} \end{cases}$$

where $U(0,1)$ means sampling from randomly and uniformly between 0 and 1. Using the above input and output sequence, choosing $\tau_{delay} = 3$, we can perform the QRC whose results are shown in figure 4.3. In this figure, we can see that the reservoir is able to reconstruct inputs up to a delay of 3 with an NRMSE error of 0.019.

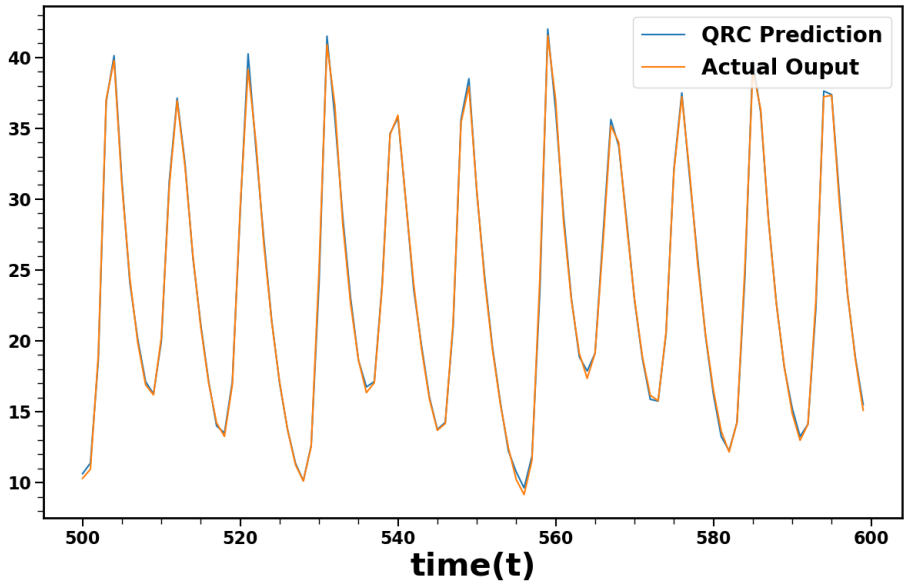

Figure 4.2: Performance of the Lorenz series prediction using GUE as a reservoir is shown. The predicted output (blue curve) for the testing time is compared with the actual output (orange curve) in the plot. Optimizing parameters are $N = 7, V = 10, \tau = 5$, transient time = 100, training time = 400, and testing time = 100.

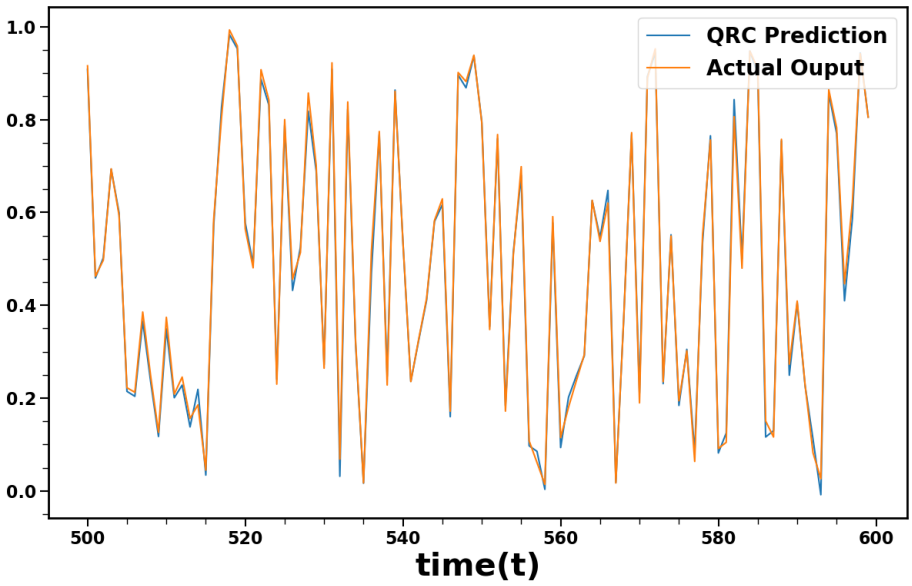

Figure 4.3: Performance of memory capacity with a delay($\tau_{delay} = 3$) using GUE as a reservoir is shown in the figure. The predicted output (blue curve) for the testing time is compared with the actual output (orange curve) in the plot. Optimizing parameters are $N = 7, V = 10, \tau = 5$, transient time = 100, training time = 400, and testing time = 100.

4.3 Comparisons with Ising model

A key question under investigation is whether random matrices impose fundamental limits on the performance of time series prediction tasks. To examine this question, QRC is performed on different qubit size systems (i.e. varying N) for NARMA, Lorenz systems and memory capacity tasks. The chosen reservoirs include the TFIM in the chaotic regime, a GUE matrix, a GOE matrix, and a Haar-random unitary. The performance is measured using the Pearson correlation coefficient, as defined in Chapter 3. It is close to 1 when the prediction is good and is close to 0 when the prediction is worse. Figure 4.4 shows all the above tasks plotted with respect to each other. Using this plot, we can draw the following observations:

1. As the system size increases, the number of computational nodes also increases, which increases the performance of the tasks.

2. All random matrices (GOE, GUE as Hamiltonian and Haar random unitary) produce equal performance for the tasks considered. Therefore, performance is not dependent on the type of random reservoir considered.

3. Random matrices perform better than TFIM for the Lorenz series prediction task, whereas TFIM performs better than the random matrices for the NARMA series prediction task.

4. The gap in the performance between random matrices and TFIM is larger for the NARMA task, whereas it is much less for the Lorenz series prediction tasks.

The conclusion that can be made from the observations is the following -

As the Lorenz series prediction has high nonlinearity compared to NARMA systems, random matrices perform best with very little system dependence. The reverse argument holds for the NARMA series as it has higher memory compared to the chaotic Lorenz system. It shows higher performance for TFIM, with a great system dependence.

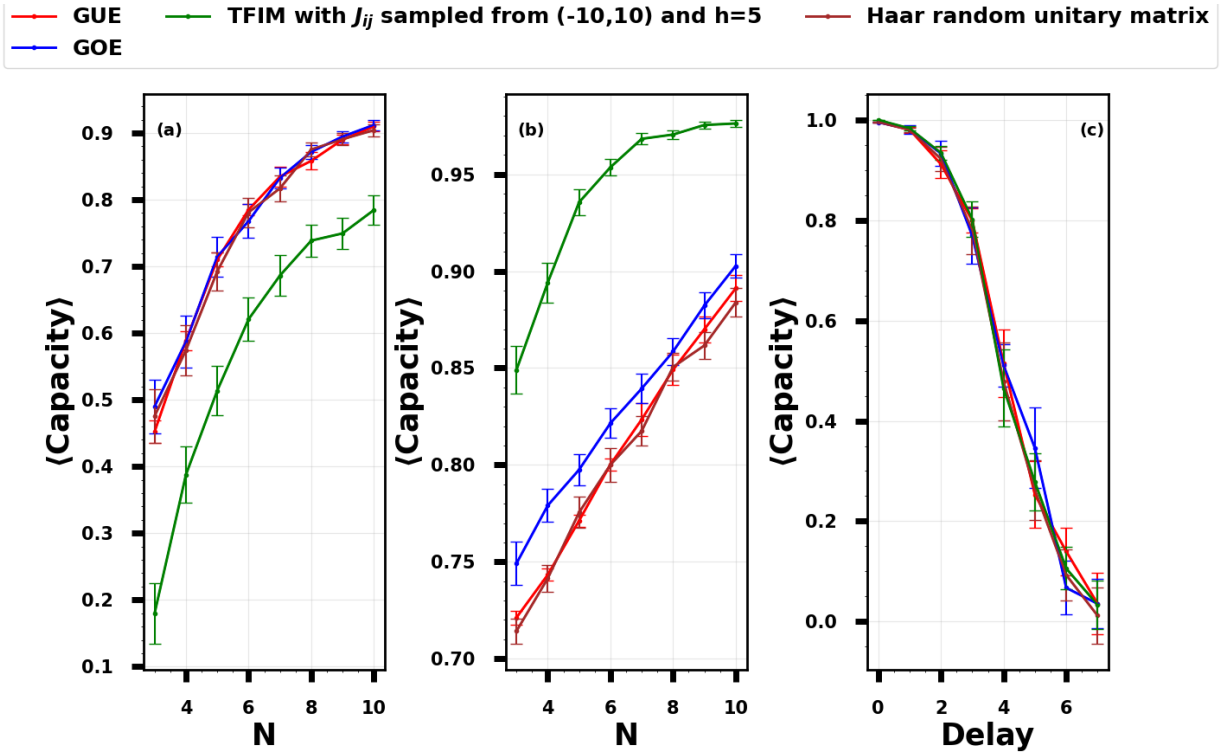

Figure 4.4: The predicted output for 4 different reservoirs are compared with respect to each other for different system sizes. The three subplots (a), (b), and (c) show the performance for three different tasks, which are NARMA-10, Lorenz series prediction, and memory capacity, respectively. Optimizing parameters are $V = 1, \tau = 1$, transient time = 100, training time = 200, and testing time = 100.

Chapter 5

Study of Chaos Boundary enhancement

It has been demonstrated that when the reservoir is taken in the quantum chaotic limit, it can predict various time series that are nonlinear and chaotic. It has also been shown that there is no limit to the performance obtained using a completely random matrix as the reservoir. The performance is highly task-dependent, as TFIM works better for NARMA prediction, while random matrices work better for the Lorenz series prediction. These results provide insight into the dependence of the performance of tasks on the reservoir's state. That is, as the reservoir lies in integrable versus quantum chaotic limit, the prediction of time series can be dependent on the reservoir state.

5.1 Chaos Boundary enhancement

This problem was first studied in 2021 by Rodrigo Martínez-Peña [22]. In their study, they observed a decrease in prediction error when the reservoir was initialized at the boundary between integrable and chaotic regions. They observed this trend in the NARMA series and various information processing capacity sequences. This represents a tradeoff between memory and non-linearity and is known as chaos-boundary enhancement.

According to the literature, chaos-boundary enhancement arises due to the following reasons. When a task requires memory, its performance is expected to improve near the integrable regime. This is because, in integrable regions, the wave function remains localized, implicating the existence of high memory in the reservoir. Conversely, non-linearity benefits from a highly delocalized state, which enhances the system's ability to predict nonlinear time series. Therefore, the performance of nonlinear tasks improves in the chaotic regime.

The preceding argument is qualitative as there exists no way to quantify memory and non-linearity for a time series or to predict the region where the highest performance is expected. However, it can be inferred that if a series requires some memory and non-linearity, then the

performance should be enhanced at the boundary between chaos and integrable region.

In this chapter, this chaos-boundary enhancement is studied for stochastic series (NARMA) and chaotic series (Lorenz system) prediction tasks using both random matrices and the TFIM. Chaos-boundary enhancement is observed during the transition from the integrable to the quantum chaotic limit in the NARMA series prediction task, while no enhancement is observed in the Lorenz series prediction.

5.2 Transition from integrability to chaos

To demonstrate chaos-boundary enhancement, we need a reservoir that transitions from integrable to chaos when we change the system's parameters. We would be performing the analysis using the following two reservoirs.

1. **Transverse field Ising Model:**
   The TFIM with all to all couplings is shown to have both integrable and chaotic regimes. This is found by analyzing spectral statistics like level spacing distribution and spacing ratio. As mentioned in Chapter 2, for the Hamiltonian, $H = \sum_{i<j}^{N} J_{ij}\sigma_i^x\sigma_j^x + h\sum_{i=1}^{N}\sigma_i^z$

   - Integrable regime: $\frac{J_0}{h} \approx 0$
   - Chaotic regime: $\frac{J_0}{h} \approx 0.5$

   So, by fixing $h = 5$ and changing J_0 from 0 to 5, we can transition from an integrable to a chaotic regime.

2. **Random matrices:**
   The random matrices following Gaussian ensemble statistics are quantum chaotic by definition. Also, the diagonal matrix is always integrable irrespective of the values of the diagonal entries. So, by combining diagonal and Gaussian ensemble matrices, we have a transition from an integrable to a chaotic regime. It is done by choosing Hamiltonian (H) as,

   $$H = \alpha D + (1-\alpha)GUE$$

   where D = diagonal matrix with random diagonal entries, GUE = Gaussian unitary ensemble matrix , α is a dimensionless parameter. When $\alpha = 1$, hamiltonian lies in an integrable regime, and it $\alpha = 0$, hamiltonian becomes a GUE matrix.

5.3 Time series prediction tasks

The time series prediction is performed in this section using reservoirs presented above to demonstrate and investigate chaos-boundary enhancement. The time series considered for the

analysis are the second NARMA system with order 10 and the Lorenz system. Performance is measured using the Pearson correlation coefficient, which gives 1 for good fit and 0 for bad fit. Also, the capacities are averaged for many reservoir iterations to measure mean capacity. This mean capacity is plotted with an error bar indicating the error in the particular mean capacity.

- **NARMA-10 series prediction:**
  The NARMA-10 series prediction is performed using both the reservoirs presented above, and the transition is studied. The results are plotted in figure 5.1, with plot (a) showing results for the random reservoir and plot (b) showing results for the TFIM. It can be seen in both plots that there is an enhancement in the performance near the boundary between integrable and chaotic regions.

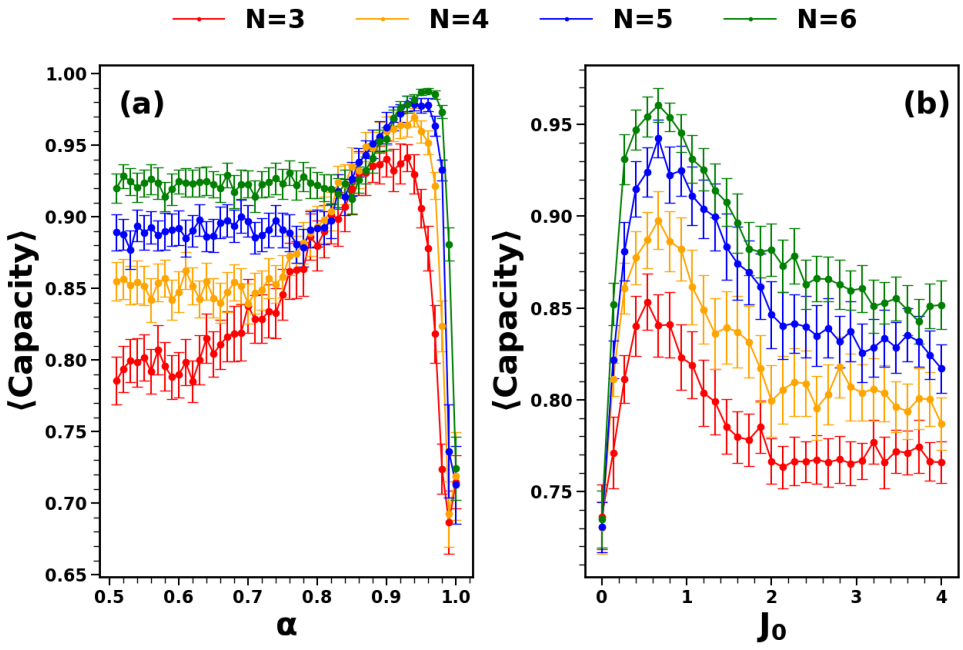

Figure 5.1: QRC performance is plotted for different Hamiltonian parameters to see chaos-boundary enhancement for NARMA-10 prediction. In plot (a), the GUE reservoir with $H = \alpha D + (1-\alpha)GUE$ is used, whereas in plot (b), TFIM is used with $h = 5$ and varying J_0 is used. Optimizing parameters are, $V = 2, \tau = 2$, transient time = 100, training time = 200, and testing time = 50 for both the plots.

- **Lorenz series prediction:**
  The Lorenz system's prediction with $x(t)$ as the input and $z(t)$ as the output is performed using both reservoirs. The results are plotted in figure 5.2, with plot (a) showing results for the random reservoir and plot (b) showing results for the TFIM. It can be seen in both plots that there is no enhancement in the performance near the boundary between integrable and chaotic regions and performance reaches the maximum in the chaotic regime.

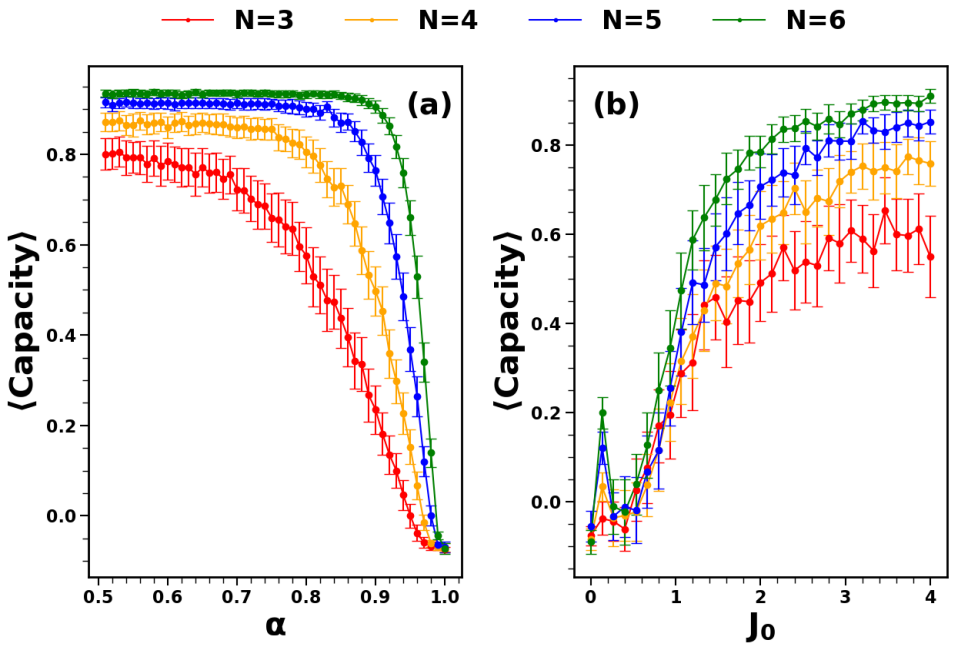

Figure 5.2: Performance of Lorenz time series prediction is plotted for different Hamiltonian parameters to see chaos-boundary enhancement. In plot (a), the GUE reservoir with $H = \alpha D + (1-\alpha)GUE$ is used, whereas in plot (b), TFIM is used with $h = 5$ and varying J_0. Optimizing parameters are, $V = 2, \tau = 2$, transient time = 100, training time = 200, and testing time = 50 for both the plots.

5.4 Krylov Complexity analysis of chaos-boundary enhancement

In the QRC, the computational nodes are measured after the evolution of the reservoir for some evolution time(τ). This suggests that the dynamical signatures of the evolution play a role in the performance of time series prediction. To explore this relationship, Krylov complexity is chosen as a measure and chaos-boundary enhancement is examined. Since the QRC process involves the evolution of the density matrix over time, the complexity of this evolution can directly impact predictive performance. Krylov complexity analyzes the evolution of a Hamiltonian with a given initial operator over a Krylov basis, aligning with the fundamental premise of QRC, making it a natural choice for this investigation.

The performance of the NARMA-10 series and Lorenz series prediction is plotted for different evolution times (τ) with the respective Krylov complexity curve of the reservoir. We have used both reservoirs for the analysis. Figure 5.3 and figure 5.4 show the results for the GUE and TIFM reservoirs, respectively. For illustration purposes, Krylov complexity is scaled with respect to its maximum value attainable, which is 2^N. Therefore, due to this scaling in the plot (a) of 5.3 and figure 5.4, even though Krylov complexity looks to attain a saturation value, it is increasing as the system moves into chaotic regime.

We can make the following observations based on these plots:

1. Although the performance depends on the evolution time (τ), it does not have a direct dependence on Krylov complexity. This is evident from the fact that while KC increases at $\tau = 1$, the performance in the Lorenz task improves and eventually saturates. Furthermore, the performance enhancement does not align with the point where the rate of change of KC is maximum.

2. For longer evolution times, saturation occurs when the Hamiltonian parameters are closer to the integrable regime. This is evident in both plots; comparing plot (a) to plot (b) shows that the enhancement is achieved for Hamiltonian parameters that are much closer to the integrable regime at higher evolution times.

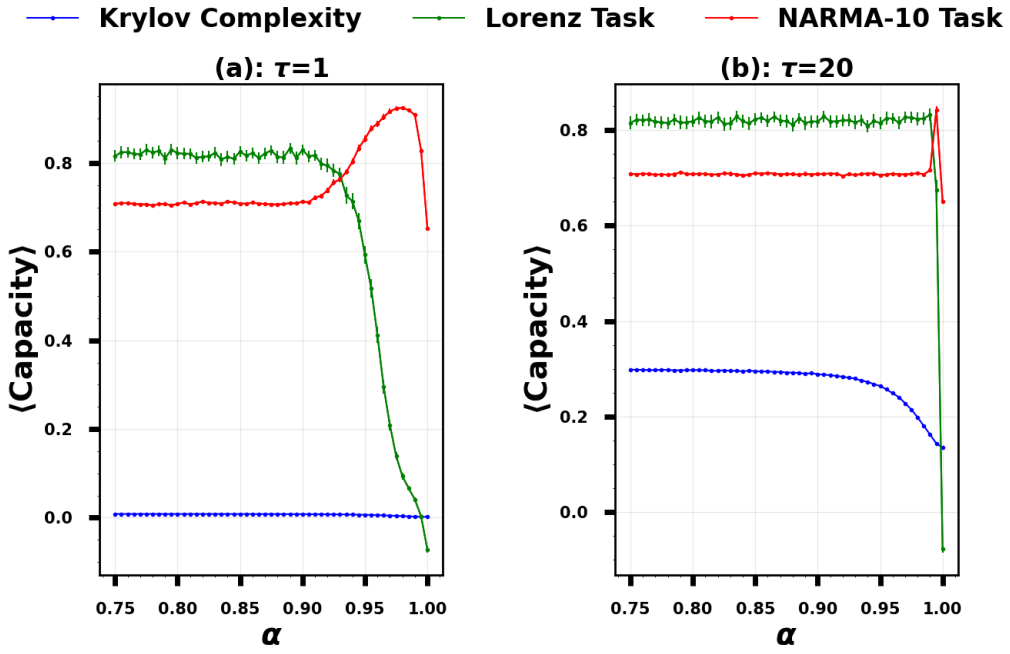

Figure 5.3: Mean Capacities of prediction of NARMA-10 and Lorenz series using QRC is shown for the GUE reservoir that has $H = \alpha D + (1-\alpha)GUE$. In plot (a), the evolution time (τ) = 1, Whereas in (b), the evolution time (τ) = 20. Optimizing parameters are $N = 6, V = 1$, transient time = 100, training time = 300, and testing time = 100

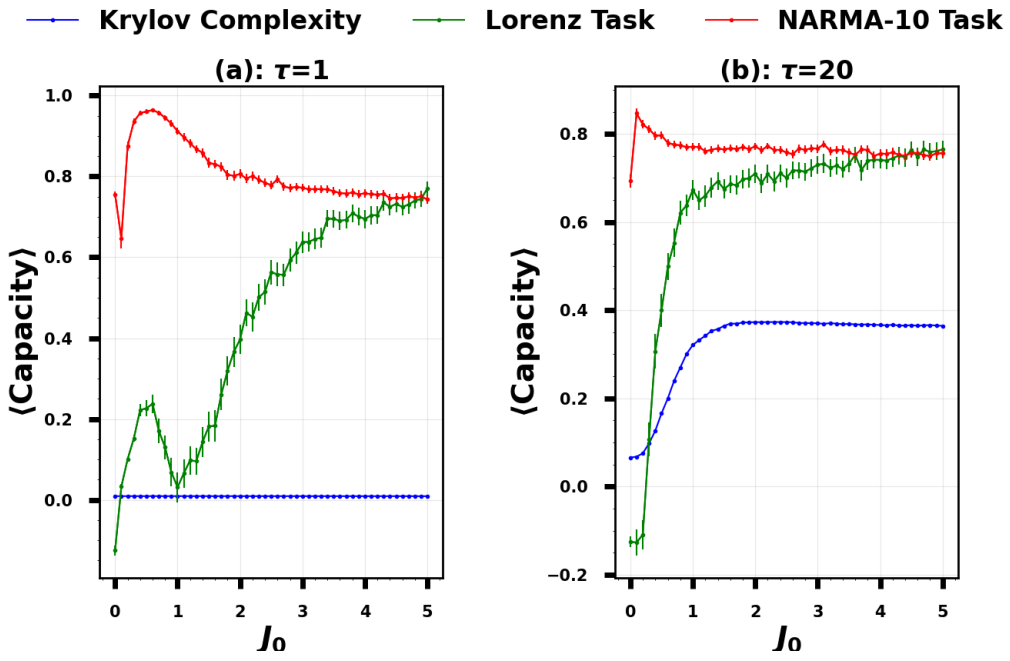

Figure 5.4: Mean Capacities of prediction of NARMA-10 and Lorenz series using QRC is shown for the TFIM reservoir which has fixed $h = 5$ and varying J_0. In plot (a), the evolution time (τ)=1, Whereas in (b), the evolution time (τ) = 20. Optimizing parameters are $N = 6, V = 1$, transient time = 100, training time = 300, and testing time = 100

Chapter 6

Chaotic time series prediction tasks

Chapter 5 (Chaos Boundary Enhancement) shows that the Lorenz series prediction task shows no chaos-boundary enhancement. This result suggests a possibility that the chaotic time series shows no enhancement at the boundary between integrable and chaotic regimes. This chapter investigates this possibility by taking the reservoir as a random GUE reservoir. Its Hamiltonian has the form $\alpha D + (1-\alpha)GUE$, which shows the transition from integrable to chaotic regime as α is varied, as shown in Chapter 5. Due to the unavailability of analytical techniques to prove/disprove the possibility, we would use computational methods. Different types of chaotic time series are analyzed, each having a nonlinear and non-analytic relationship between input and output sequences.

6.1 Classical kicked top

The classical kicked top [18] is a model used for studying chaotic dynamics in both classical and quantum systems. It consists of a spin-like angular momentum system subjected to periodic "kicks," where nonlinear interactions are introduced at discrete intervals. This system shows both classical chaos and integrable dynamics for different parameters. The Hamiltonian of the model is of the form,

$$H(t) = \frac{\hbar p}{\tau} J_y + \frac{\hbar k}{2j} J_z^2 \sum_n \delta(t - n\tau)$$

where J_x, J_y, J_z are angular momentum components, p determines the precession rate, k is the kicking strength, and τ is the period of kicks.

The equations of motion for the classical kicked top can be derived from Hamilton's equations, which is $\dot{J}_i = \{J_i, H\}$. The equations of motion obtained at discrete time intervals can be written in iterative form as,

$$A = J_z \sin p + J_x \cos p$$

$$B = J_z \cos p - J_x \sin p$$

$$J'_x = A\cos(kB) - J_y\sin(kB)$$

$$J'_y = A\sin(kB) + J_y\cos(kB)$$

$$J'_z = B$$

where J_x, J_y, and J_z represent the components of the angular momentum, and the parameters k and p control the dynamics of the system. These equations define the iterative map that evolves the system at discrete time steps.

The transition between integrability and chaos is controlled by k, while the parameter p (which represents precession) mainly affects the phase space structure but does not directly induce chaos. When k is small, the system is near-integrable, and when $k > 3$, the model shows complete classical chaos.

After evolving the system in a chaotic regime ($k > 3$), we will get 3 time series, which are $J_x(t), Jy(t), J_z(t)$. They don't have an analytic dependence among themselves and can be used for chaotic time series prediction. We use $J_x(t)$ as the input and want to predict $J_z(t)$ as the output. The prediction's capacity is plotted versus α in figure 6.1. In subplot (b) of this figure, we can see that no significant enhancement is observed around the chaos-boundary region.

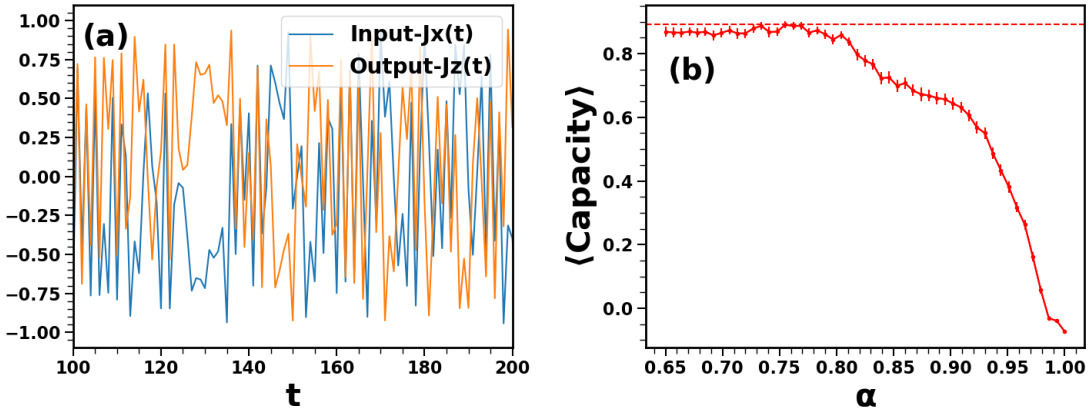

Figure 6.1: Mean Capacities of prediction of the classical kicked top (CKT) using QRC is shown for the GUE reservoir versus varying α. The evolution of CKT is considered within a chaotic regime with $k = 6$ and $p = \pi/2$. In plot (a), the input ($J_x(t)$) and output ($J_z(t)$) sequence are plotted, whereas in (b), the mean capacity of the prediction versus α is plotted. Optimizing parameters are $N = 4, V = 1, \tau = 2$, transient time = 100, training time = 100, and testing time = 200

6.2 Classical kicked rotor

The Classical kicked rotor [32] is also a model used for studying chaotic dynamics in classical and quantum systems, just like the classical kicked top. It consists of a rotor (a pendulum rotating freely) that receives periodic kicks at discrete time intervals. The Hamiltonian of the system is given by:

$$H = \frac{p^2}{2I} + K\cos(\theta)\sum_n \delta(t - nT) \tag{6.1}$$

Where p is the angular momentum, θ is the angular position, I is the moment of inertia (often set to 1 for simplicity), K is the kicking strength, and T is the period between successive kicks. We can derive the equations of the motion by using Hamilton's equations, which give an iterative map to produce p(t) and $\theta(t)$ at the discrete time steps when the kick is applied. It is well known as the Chirikov Standard Map:

$$p_{n+1} = p_n + K\sin(\theta_n)$$

$$\theta_{n+1} = \theta_n + p_{n+1} \pmod{2\pi}$$

The Chirikov standard map exhibits integrable and chaotic dynamics governed by the kick parameter (K). For $K < 0.971$, the system remains in the integrable regime, characterized by regular, quasi-periodic motion. As K increases beyond 0.971, chaotic regions begin to emerge in phase space, and the degree of chaos intensifies with further increases in K. The system becomes fully chaotic when K exceeds 5, with global diffusion in phase space.

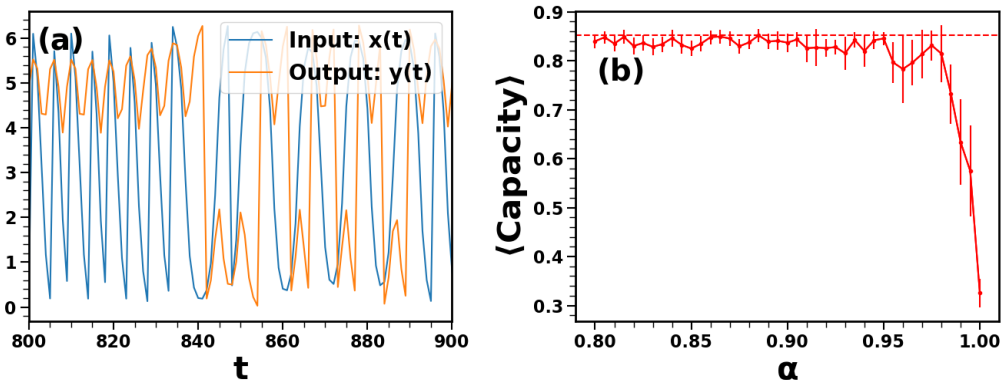

Figure 6.2: Mean Capacities of prediction of the classical kicked rotor (CKR) using QRC is shown for the GUE reservoir versus varying α. The evolution of CKR is considered within a chaotic regime with $K = 1.1$ and $\theta(0) = 0.6$ and $p(0) = 1$. In plot (a), the input ($\theta(t)$) and output sequence ($p(t)$) are plotted, whereas in (b), the mean capacity of the prediction versus α is plotted. Optimizing parameters are $N = 7, V = 10, \tau = 2$, transient time = 100, training time = 1000, and testing time = 100

After evolving the system in a chaotic regime ($k \approx 1.2$), we will get two-time series, which

are $p(t)$ and $\theta(t)$. They don't have an analytic dependence and can be used for chaotic time series prediction. We use $\theta(t)$ as the input and want to predict $p(t)$ as the output. The prediction's capacity is plotted versus α in figure 6.2. In subplot(b) of this figure, we can see that no significant enhancement is observed around the chaos-boundary region. One key observation here is that no set of optimizing parameters is found when the K becomes equal to 5 (complete chaos limit), such that the prediction has a reasonably low error.

6.3 Ikeda Map

The Ikeda Map [36] is a discrete-time dynamical system that describes the behaviour of light in an optical cavity with a nonlinear dielectric medium. This system undergoes chaotic dynamics for specific parameters. This map is given by the recurrence relations:

$$x_{n+1} = 1 + u(x_n \cos t_n - y_n \sin t_n), \tag{6.2}$$

$$y_{n+1} = u(x_n \sin t_n + y_n \cos t_n), \tag{6.3}$$

Where, u is the feedback parameter controlling nonlinear feedback strength, κ is Constant phase shift, p is the nonlinearity parameter influencing system behaviour, and the phase t_n is given by:

$$t_n = \kappa - \frac{p}{1 + x_n^2 + y_n^2}. \tag{6.4}$$

The Ikeda map exhibits chaotic dynamics when the feedback parameter u is sufficiently large, typically beyond $u > 1$, leading to period-doubling bifurcations and strange attractors. For values like $\kappa = 0.4$, $p = 6.0$, and $u = 0.9$, the system shows chaotic behavior. Increasing p enhances nonlinearity, further promoting chaos. As u increases, the system transitions from fixed points to periodic cycles and then to chaos.

Evolving the system in a chaotic regime ($\kappa = 0.4$, $p = 6.0$, and $u = 0.9$) produces two-time series, $x(t)$ and $y(t)$, which lack an analytic dependence and can be utilized for chaotic time series prediction. Here, $x(t)$ serves as the input, while $y(t)$ is the target output. The prediction capacity is plotted against α in Figure 6.3. In subplot (b), we observe no significant enhancement near the chaos-boundary region.

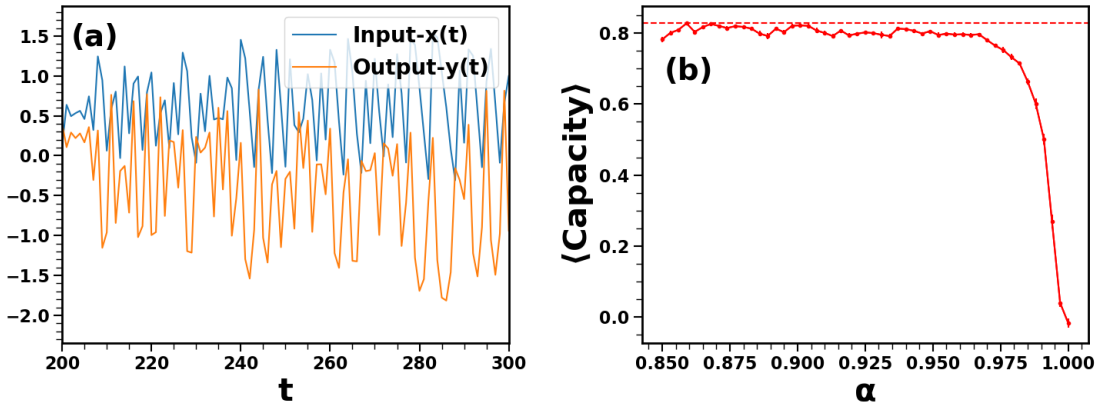

Figure 6.3: Mean Capacities of prediction of the Ikeda map using QRC is shown for the GUE reservoir versus varying α. The evolution of the Ikeda map is considered within a chaotic regime with $u = 0.9$ and $x(0) = 0.1$ and $y(0) = 0.1$. In plot (a), the input $(x(t))$ and output sequence $(y(t))$ are plotted, whereas in (b), the mean capacity of the prediction versus α is plotted. Optimizing parameters are $N = 6, V = 1, \tau = 2$, transient time = 100, training time = 800, and testing time = 100

Chapter 7

Quantum Reservoir Computing as a Volterra Expansion

In this chapter, we have investigated the possibility that quantum reservoir computing can be thought of as a Volterra expansion. That is, we can explain the performance of any time series prediction using quantum reservoir computing based on the expansion of time series in dynamic and static nonlinearities.

7.1 Volterra Expansion

Any continuous or discrete time-varying signal can be decomposed in terms of static and dynamic nonlinearities [29]. A continuous time-invariant system with $x(t)$ as input and $y(t)$ as output can be expanded in the Volterra series as

$$y(t) = h_0 + \sum_{n=1}^{N} \int_a^b \cdots \int_a^b h_n(\tau_1, \ldots, \tau_n) \prod_{j=1}^{n} x(t-\tau_j)\, d\tau_j$$

where, the constant term h_0 is usually taken to be zero and function $h_n(\tau_1, \tau_2, .., \tau_n)$ is called the nth-order Volterra kernel.

The discrete-time series case is similar to the continuous-time case, except that the integrals are replaced by summations:

$$y(n) = h_0 + \sum_{p=1}^{P} \sum_{\tau_1=a}^{b} \cdots \sum_{\tau_p=a}^{b} h_p(\tau_1, \ldots, \tau_p) \prod_{j=1}^{p} x(n-\tau_j),$$

where $P \in \{1, 2, \ldots\} \cup \{\infty\}$ and each function h_p is called a *discrete-time Volterra kernel*. The terms in the expansion (P) can be finite or infinite; usually, in real examples, we deal with finite P, and it gives a truncated Volterra series.

An example of a Volterra series we can consider a discrete-time series. Truncating the expansion to second order, we can write one particular Volterra series as,

$$y(n) = 0.9x(n) + 0.1x(n-1) + 0.4x(n-1)x(n-4) + 0.6x(n-2)x(n-4)$$

where $h_0 = 0, h_1(0) = 0.9, h_1(1) = 0.1, h_2(1,4) = 0.4, h_2(2,4) = 0.6$ and all remaining kernel terms are zero. By modifying the kernels at various orders, different forms of the Volterra series can be obtained, each representing a unique nonlinear system behaviour.

7.2 Error Propagation technique

In QRC, the input $x(t)$ and output series $y(t)$ are discrete in time. So, using the discrete-time Volterra expansion, we can write $y(t)$ as a function of $x(t)$. The exact expansion can involve infinite or very large terms. Truncating the expansion so that it includes a finite number of terms, we get the following expansion,

$$y(t) = \sum_{i=0}^{N} h_i * y_i(t) + \delta(t) \tag{7.1}$$

where index (i) goes over all nonlinear terms like $(x(t-\tau_1) * x(t-\tau_2), ..)$, y_i represents the corresponding nonlinear term and the h_i the kernel associated with it. The error $(\delta(t))$ is the error associated with the truncation of the Volterra series. For a given input $x(t)$ to the reservoir, we get $n_j(t)$ as the output nodes, which will be trained to get the target output series. As there is no output feedback, $n_j(t)$ is independent of the $y(t-1)$, and it is dependent only on the $x(t)$. Performing linear regression on the output nodes, we get the output series with some errors. This can be written as,

$$y(t) = \sum_{j=0}^{M} \tilde{w_j} * n_j(t) + \phi(t) \tag{7.2}$$

The above expansion is what we get in the usual quantum reservoir commuting setup, where M is the total number of output nodes, $n_j(t)$ is a particular output node, and $\tilde{w_j}$ is the weight corresponding to the particular output node. Using the same output nodes $n_j(t)$, we can train individual Volterra terms like $y_i(t)$ in the same way as we train the target output $y(t)$. This expansion can be written as,

$$y_i(t) = \sum_{j=0}^{M} w_{ij} * n_j(t) + \varepsilon_i(t) \tag{7.3}$$

Combining equations 1 and 3, we can write target output series as,

$$y(t) = \sum_{i=0}^{N} \sum_{j=0}^{M} w_{ij} * h_i * n_j(t) + \sum_{i=1}^{N} h_i * \varepsilon_i(t) + \delta(t) \tag{7.4}$$

The above equations give a way to predict the target output series based on its Volterra series expansion and the performance of each of the constituent Volterra terms. Here the fitting error comes out be $\sum_{i=1}^{N} h_i * \varepsilon_i(t) + \delta(t)$ which we denote as $TE(t)$. The $TE(t)$ is a function of time (t) and the particular Volterra terms chosen for the analysis. Minimization of this error with respect to the Volterra terms gives the minimum $TE(t)$ possible.

In order to claim that the QRC can be explained using Volterra expansion, $TE(t)$ should follow $\phi(t)$, which can be inferred effectively if $TE(t)$ primarily satisfies the following conditions:

1. $RMSE(TE(t)) \geq RMSE(\phi(t))$ for all possible volterra combinations.
2. $min(RMSE(TE(t))) = RMSE(\phi(t))$

Due to the lack of analytical techniques to test the above statements, we rely on computational techniques to investigate the hypothesis.

7.3 Computational Analysis

Computational tests are conducted to examine the relationship between $TE(t)$ and $\phi(t)$. For this analysis, a random matrix $[\alpha \cdot D + (1-\alpha) \cdot GUE]$ is selected as the reservoir to facilitate the transition from integrability to chaos. The target series for prediction is the Lorenz $z(t)$ sequence, with the Lorenz $x(t)$ sequence serving as the input. Since the output nodes do not depend on the target series and are solely functions of the input series, the reservoir is first evolved to generate the output nodes $n_j(t)$ over the entire time duration. Subsequently, linear regression is performed to determine all Volterra terms and the Lorenz prediction, yielding $\varepsilon_i(t)$ and $\phi(t)$, respectively. The error $\delta(t)$ is obtained by training the Lorenz $z(t)$ sequence on selected Volterra terms over both the training and testing periods, consistent with the approach used in the quantum reservoir computing case. This methodology is then employed to generate various plots for further analysis.

1. **Volterra decomposition of Output nodes**

   The motivation that quantum reservoir computing can be thought of as Volterra decomposition can be found by performing the Volterra expansion of the output nodes. These

are obtained from the reservoir's evolution given Lorenz $x(t)$ as input. Figure 7.1 shows that all the different output nodes follow the Volterra expansion with minimal deviation.

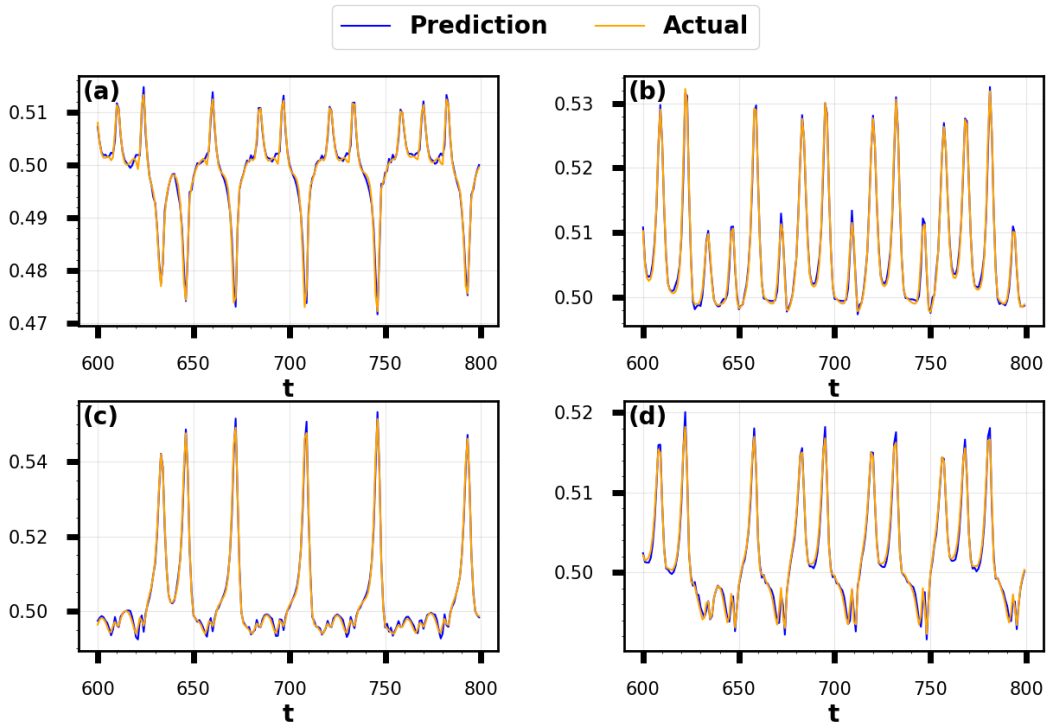

Figure 7.1: Volterra expansion of the output nodes, given Lorenz $x(t)$ series as input for the testing period. Optimizing parameters are $N = 4, V = 1, \tau = 1$, training time = 500, testing time = 100, the reservoir is in GUE limit, i.e. $\alpha = 0$, and only 30 different individual Volterra terms are taken with delay = 10 and order = 2. The (a), (b), (c), (d) plots correspond to different output nodes, which are $\langle z_1 \rangle, \langle z_2 \rangle, \langle z_3 \rangle, \langle z_4 \rangle$ respectively.

2. $TE(t)$ **and** $\phi(t)$ **versus time** (t)**:**
   To further investigate the hypothesis, the temporal evolution of $TE(t)$ and $\phi(t)$ is analyzed by plotting them against time (t). This visualization provides insight into the magnitude of both errors and their relationship. Figure 7.2 presents the $TE(t)$ and $\phi(t)$ curves, indicating that $|TE(t)|$ exceeds $|\phi(t)|$ for most time instances, as reflected in the RMSE values. However, the proximity of the curves suggests that $TE(t)$ may approach $\phi(t)$ under different Volterra expansions.

3. **Scaling of error with different volterra expansion:**
   There are two parts of the total error $(TE(t))$, which are $\sum_{i=1}^{N} h_i \cdot \varepsilon_i(t)$ and $\delta(t)$. In this section, their scaling for the different Volterra expansions is studied.

In Figure 7.3, the variation of RMSE$(\delta(t))$ is plotted against different Volterra sequences for the Lorenz series prediction task. The calculation involves generating mul-

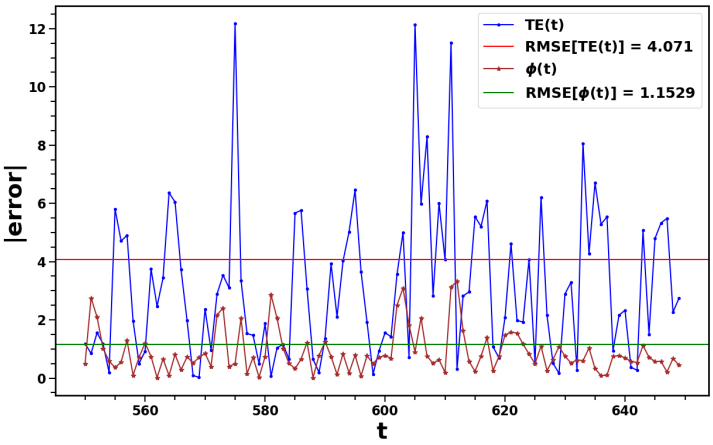

Figure 7.2: $TE(t)$ and $\phi(t)$ is plotted versus time. Optimizing parameters are $N = 7, V = 1, \tau = 1$, training time = 500, testing time = 100, the reservoir is in GUE limit, i.e. $\alpha = 0$, and only 40 different individual Volterra terms are taken with delay = 6 and order = 3.

tiple terms with varying delay and order from the Lorenz $x(t)$ sequence, which are then utilized to predict the Lorenz $z(t)$ sequence. There is no role of quantum reservoir computing in these computations. The plot demonstrates that increasing the number of terms in the Volterra expansion leads to a monotonic reduction in the error ($\delta(t)$). It implies that a lower contribution of $\delta(t)$ in $TE(t)$ is achieved when more dynamic nonlinearities are incorporated.

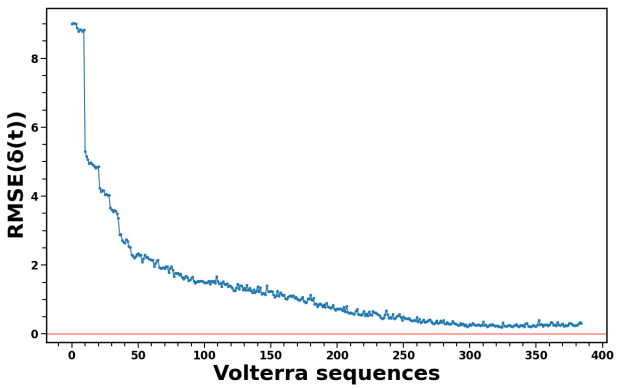

Figure 7.3: RMSE($\delta(t)$) is plotted for different Volterra expansions of Lorenz $z(t)$ prediction using Lorenz $x(t)$ as input. Moving right on the x-axis, the expansion includes more terms with higher delay and order. Training time rises with increasing Volterra terms for proper linear regression while testing time remains fixed at 100 iterations, with a maximum delay of 10 and maximum order of 4.

The other error term in the total error, $TE(t)$, which is given by $\sum_{i=1}^{N} h_i \cdot \varepsilon_i(t)$. Figure 7.4 presents the NRMSE values of $\varepsilon_i(t)$ for different Volterra terms (indexed by i) as a function of α. In Figure 7.4 (a), the errors associated with different Volterra terms vary across α, with terms of lower delay and order exhibiting smaller errors compared to those with higher delay and order in the GUE limit. However, this effect diminishes near the chaos boundary due to the enhancement observed for higher order and nonlinear term at the transition, indicating that the NRMSE error of $\varepsilon_i(t)$ does not monotonically increase but instead saturates in this region. Figure 7.4 (b) further illustrates this behaviour, showing that terms with greater delay and order experience significant chaos-boundary

enhancement, ultimately leading to the saturation of the NRMSE error.

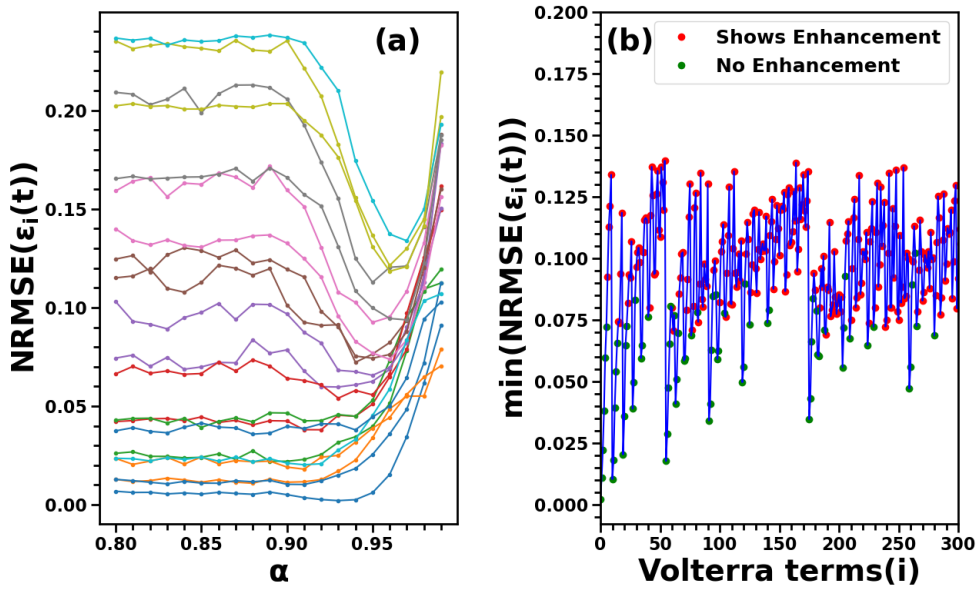

Figure 7.4: Plot(a) shows the NRMSE of $\varepsilon_i(t)$ versus α for various volterra terms($i's$). Various curves plotted in (a) represent different Volterra terms (i), with the terms with higher delay and order having high NRMSE. Whereas (b) shows the minimum NRMSE attained across various α values chosen from various Volterra terms. On the x-axis, as we move to the right, the delay and order of the term increases. The threshold to classify enhancement is 0.05. Optimizing parameters $N = 7, V = 2, \tau = 1$, training time = 250, testing time = 100, the maximum delay is 10, and the maximum order is 4.

Therefore, combining both the error terms, we would expect a tradeoff with respect to the number of individual Volterra terms considered. This is because an increase in the number of nonlinearities in the Volterra expansion results in a higher $\sum_{i=1}^{N} h_i \cdot \varepsilon_i(t)$ but lower $\delta(t)$ and reverse happens when it is decreased. This establishes an inverse relationship between $(\sum_{i=1}^{N} h_i \cdot \varepsilon_i(t))$ and $(\delta(t))$ as the size of the Volterra expansion increases resulting in a tradeoff. Also, the variation of $\sum_{i=1}^{N} h_i \cdot \varepsilon_i(t)$ versus α results in one more complexity that gives the chaos-boundary enhancement, which is studied below.

4. **Chaos-boundary enhancement and boundedness property of $TE(t)$:**
The chaos-boundary enhancement and boundedness property of $TE(t)$ are analyzed for the Lorenz series prediction task. The system transitions from an integrable regime (diagonal matrix D) to a chaotic regime (Gaussian Unitary Ensemble, GUE) by varying α in the Hamiltonian of the form $\alpha D + (1-\alpha)GUE$. The objective is to determine whether $TE(t)$ bounds $\phi(t)$ and whether it accurately identifies instances where no enhancement occurs in chaotic time-series prediction.

Figure 7.5 presents the RMSE of $TE(t)$ for different Volterra expansions and $\phi(t)$

as a function of α. In subplot (a), for most values of $\alpha < 1$, RMSE($TE(t)$) exceeds RMSE($\phi(t)$) across all considered Volterra expansions. Only for α values close to 1 does RMSE($TE(t)$) become lower than RMSE($\phi(t)$) in certain cases. Additionally, varying the size of the Volterra expansion influences RMSE($TE(t)$). The minimum RMSE($TE(t)$) is computed by testing different Volterra expansion combinations and is shown in subplot (b). The minima curve closely follows the behaviour of RMSE($\phi(t)$), indicating no enhancement as both curves exhibit similar saturation characteristics.

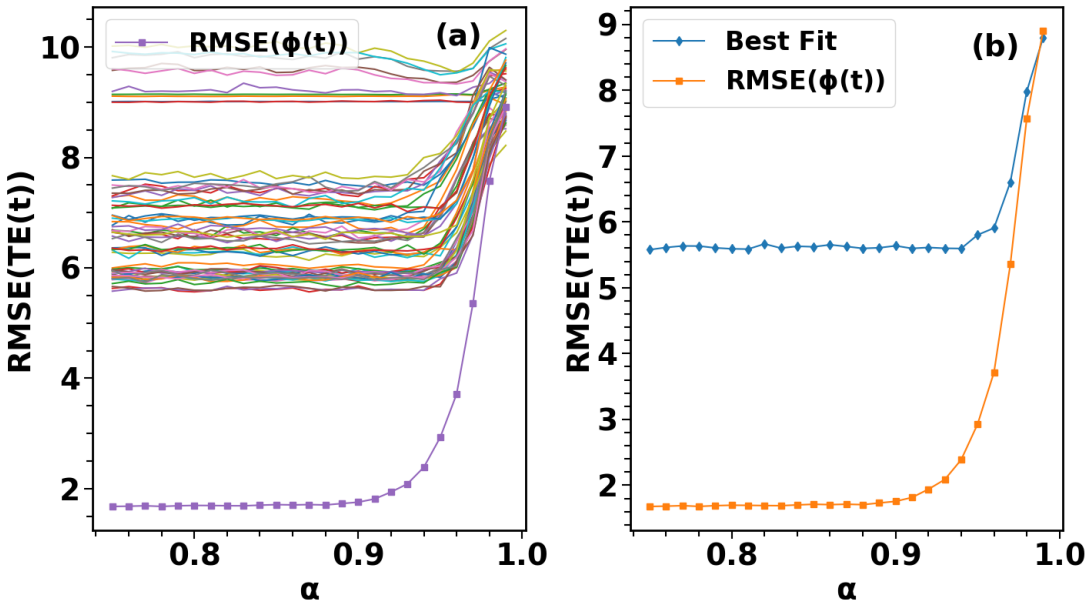

Figure 7.5: Plot (a) shows RMSE($TE(t)$) for different Volterra expansions and varying α for the Lorenz series prediction task. Meanwhile, plot (b) shows the minimum RMSE($TE(t)$) obtained by brute force. RMSE($\phi(t)$) is plotted in both the plots for the comparison. $\alpha = 1$ corresponds to the integrable limit of hamiltonian, and as α decreases, Hamiltonian approaches the GUE limit. Optimizing parameters are $N = 7, V = 2$, transient time = 100, training time = 250, testing time = 100, the maximum delay is 10, and the maximum order is 4.

A similar analysis is conducted for the NARMA-10 task, a stochastic time series exhibiting chaos-boundary enhancement. Figure 7.6 presents the RMSE of $TE(t)$ for different Volterra expansions and $\phi(t)$ as a function of α in the NARMA-10 prediction task, following the same methodology applied in the Lorenz series prediction. The results indicate that, across all Volterra expansions, the RMSE($TE(t)$) curve predominantly remains above the RMSE($\phi(t)$) curve, converging towards it for specific parameter combinations. The minimum value of RMSE($TE(t)$), which closely approximates RMSE($\phi(t)$), is identified and highlighted in the subplot (b) of Figure 7.6. All the RMSE($TE(t)$) curves approaching RMSE($\phi(t)$), including the min(RMSE($TE(t)$)) curve, exhibit chaos-boundary enhancement.

Therefore, numerically, we can infer that the two initial requirements are approximately satis-

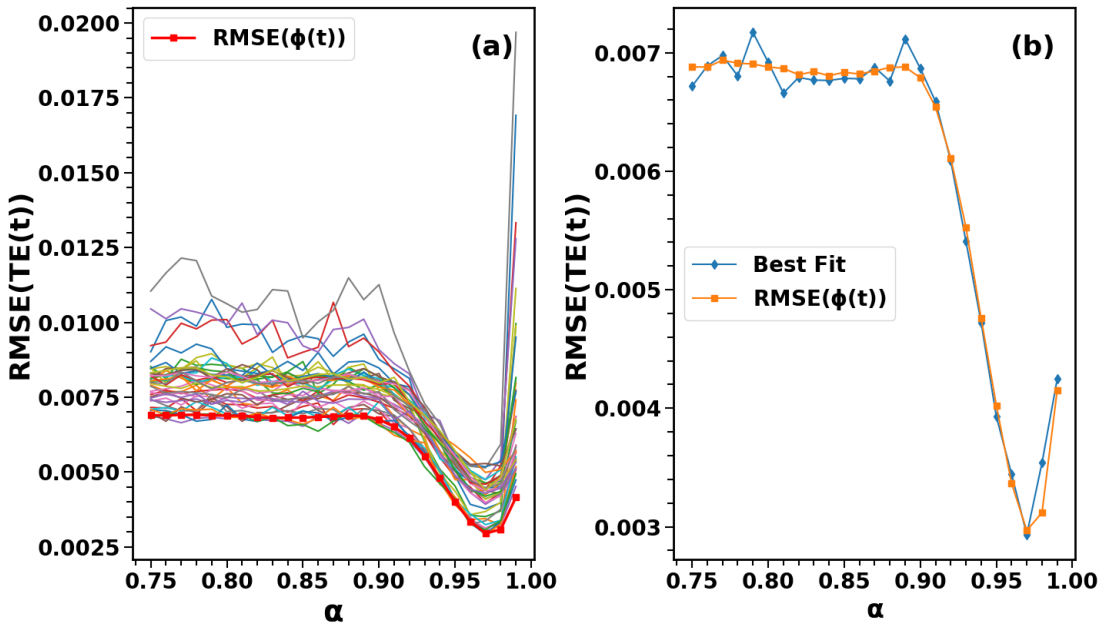

Figure 7.6: Plot (a) shows RMSE($TE(t)$) for different Volterra expansions and varying α for the NARMA-10 prediction task. Meanwhile, plot (b) shows the minimum RMSE($TE(t)$) obtained by brute force. RMSE($\phi(t)$) is plotted in both the plots for the comparison. $\alpha = 1$ corresponds to the integrable limit of hamiltonian, and as α decreases, Hamiltonian approaches the GUE limit. Optimizing parameters are $N = 7, V = 2$, transient time = 100, training time = 250, testing time = 100, the maximum delay is 10, and the maximum order is 4.

fied. Specifically,

1. RMSE($TE(t)$) remains greater than RMSE($\phi(t)$) for all possible Volterra combinations.

2. The minimum value of RMSE($TE(t)$) approaches RMSE($\phi(t)$) closely; however, it cannot be asserted that min(RMSE($TE(t)$)) is exactly equal to RMSE($\phi(t)$).

Chapter 8

Conclusions

This thesis explores quantum reservoir computing (QRC) using qubit systems and investigates the fundamental limitations of its performance. The key conclusions drawn from this study are as follows:

1. **Implementation of QRC**: Quantum reservoir computing can be effectively implemented using the transverse field Ising model and random matrices for a range of temporal tasks.

2. **Task Dependence on the Reservoir**: The choice of the reservoir is task-dependent. The Transverse Field Ising Model is best suited for memory-intensive tasks, whereas random matrices exhibit superior performance in handling nonlinear and chaotic tasks.

3. **Dependence on Evolution Time**: QRC performance is highly sensitive to the evolution time τ. A longer evolution time increases the information scrambling, even in integrable systems, thereby shifting performance towards the trend observed in quantum chaotic regimes.

4. **Performance of Random Matrix Reservoirs**: Random matrix reservoirs, such as those based on the Gaussian Orthogonal Ensemble (GOE) and Gaussian Unitary Ensemble (GUE), demonstrate the highest accuracy in chaotic time-series prediction. These chaotic time series tasks show negligible enhancement at the chaos boundary.

5. **Memory-Intensive Tasks**: For tasks with higher memory requirements, chaos-boundary enhancement is observed. Also, different systems have different chaos-boundary natures, therefore, the performance of these tasks shows high dependence on the structure of the quantum reservoir.

6. **Volterra Expansion Perspective**: The performance obtained by QRC can be understood through the Volterra expansion framework. The chaos-boundary enhancement and error dynamics are captured in the Volterra expansion approach and serve as a way to standardise QRC.

Chapter 9

Future Directions

Quantum reservoir computing is a relatively new field, and there exist plenty of questions that need to be explored. Our thesis mainly focuses on standardizing the reservoir for different time series prediction tasks on qubit-based systems. We have obtained strong numerical evidence for the reservoir dynamics' dependencies on chaotic and memory-intensive tasks. One of the directions for future work is to prove, in a more concrete manner, that chaotic time series shows the best performance for random reservoirs.

The major direction introduced in this study is the connection between QRC and the Volterra expansion. It is shown numerically that the Volterra expansion technique yields results similar to those obtained using QRC. This shows strong potential for standardizing the reservoir and providing insights into the functioning of QRC. The major questions that need to be addressed here are:

1. Generalizability of this result.
2. Finding a method to calculate minimum $RMSE(TE(t))$
3. Explanation of chaos-boundary enhancement for both chaotic and memory-intensive tasks.
4. Analytical proof of the two conditions between $RMSE(TE(t))$ and $RMSE(\phi(t))$

Bibliography

[1] Priyanka Bhovad and Suyi Li. Physical reservoir computing with origami and its application to robotic crawling. *Scientific Reports*, 11(1):13002, 2021.

[2] Benedikt Bollig, Martin Leucker, and Daniel Neider. A survey of model learning techniques for recurrent neural networks. *A Journey from Process Algebra via Timed Automata to Model Learning: Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday*, pages 81–97, 2022.

[3] Angelo Casolaro, Vincenzo Capone, Gennaro Iannuzzo, and Francesco Camastra. Deep learning for time series forecasting: Advances and open problems. *Information*, 14(11):598, 2023.

[4] Kamal Choudhary, Brian DeCost, Chi Chen, Anubhav Jain, Francesca Tavazza, Ryan Cohn, Cheol Woo Park, Alok Choudhary, Ankit Agrawal, Simon JL Billinge, et al. Recent advances and applications of deep learning methods in materials science. *npj Computational Materials*, 8(1):59, 2022.

[5] Julien Dudas, Baptiste Carles, Erwan Plouet, Frank Alice Mizrahi, Julie Grollier, and Danijela Marković. Quantum reservoir computing implementation on coherently coupled quantum oscillators. *npj Quantum Information*, 9(1):64, 2023.

[6] François Duport, Bendix Schneider, Anteo Smerieri, Marc Haelterman, and Serge Massar. All-optical reservoir computing. *Optics express*, 20(20):22783–22795, 2012.

[7] Anuj Kumar Dwivedi and Mani Dwivedi. A study on the role of machine learning in natural language processing. *International Journal of Scientific Research in Computer Science Engineering and Information Technology*, 2022.

[8] Duccio Fanelli, Luca Bindi, Lorenzo Chicchi, Claudio Pereti, Roberta Sessoli, and Simone Tommasini. A short introduction to neural networks and their application to earth and materials science. *Rendiconti Lincei. Scienze Fisiche e Naturali*, pages 1–12, 2024.

[9] AG Farizawani, M Puteh, Y Marina, and A Rivaie. A review of artificial neural network learning rule based on multiple variant of conjugate gradient approaches. In *Journal of Physics: Conference Series*, volume 1529, page 022040. IOP Publishing, 2020.

[10] Daniel Fry, Amol Deshmukh, Samuel Yen-Chi Chen, Vladimir Rastunkov, and Vanio Markov. Optimizing quantum noise-induced reservoir computing for nonlinear and chaotic time series prediction. *Scientific Reports*, 13(1):19326, 2023.

[11] Keisuke Fujii and Kohei Nakajima. Harnessing disordered-ensemble quantum dynamics for machine learning. *Physical Review Applied*, 8(2):024030, 2017.

[12] Ignacio García-Mata, Rodolfo A. Jalabert, and Diego A. Wisniacki. Out-of-time-order correlators and quantum chaos. *Scholarpedia*, 18:55237, 2023.

[13] Gideon Gbenga Oladipupo. Research on the concept of liquid state machine. *arXiv e-prints*, pages arXiv–1910, 2019.

[14] Bertrand Georgeot and Dima L Shepelyansky. Quantum chaos border for quantum computing. *Physical Review E*, 62(3):3504, 2000.

[15] Sanjib Ghosh, Tanjung Krisnanda, Tomasz Paterek, and Timothy CH Liew. Realising and compressing quantum circuits with quantum reservoir computing. *Communications Physics*, 4(1):105, 2021.

[16] Sanjib Ghosh, Andrzej Opala, Michał Matuszewski, Tomasz Paterek, and Timothy CH Liew. Quantum reservoir processing. *npj Quantum Information*, 5(1):35, 2019.

[17] Aviva Gubin and Lea F Santos. Quantum chaos: An introduction via chains of interacting spins 1/2. *American Journal of Physics*, 80(3):246–251, 2012.

[18] Fritz Haake, M Kuś, and Rainer Scharf. Classical and quantum chaos for a kicked top. *Zeitschrift für Physik B Condensed Matter*, 65:381–395, 1987.

[19] Koji Hashimoto, Keiju Murata, Norihiro Tanahashi, and Ryota Watanabe. Krylov complexity and chaos in quantum mechanics. *Journal of High Energy Physics*, 2023(11):1–41, 2023.

[20] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German national research center for information technology gmd technical report*, 148(34):13, 2001.

[21] Thubten Jamtsho, Krishna Powdyel, Reshan Kumar Powrel, Rakesh Bhujel, and Kazuhiro Muramatsu. Ocr and speech recognition system using machine learning. In *2021 Innovations in Power and Advanced Computing Technologies (i-PACT)*, pages 1–5. IEEE, 2021.

[22] Rodrigo Martínez-Peña, Gian Luca Giorgi, Johannes Nokkala, Miguel C Soriano, and Roberta Zambrini. Dynamical phase transitions in quantum reservoir computing. *Physical Review Letters*, 127(10):100502, 2021.

[23] Glen Bigan Mbeng, Angelo Russomanno, and Giuseppe E Santoro. The quantum ising chain for beginners. *SciPost Physics Lecture Notes*, page 082, 2024.

[24] Madan Lal Mehta. *Random matrices*, volume 142. Elsevier, 2004.

[25] Ibomoiye Domor Mienye, Theo G Swart, and George Obaido. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information*, 15(9):517, 2024.

[26] Makoto Negoro, Kosuke Mitarai, Kohei Nakajima, and Keisuke Fujii. Toward nmr quantum reservoir computing. *Reservoir computing: theory, physical implementations, and applications*, pages 451–458, 2021.

[27] Varun Kumar Ojha, Ajith Abraham, and Václav Snášel. Metaheuristic design of feedforward neural networks: A review of two decades of research. *Engineering Applications of Artificial Intelligence*, 60:97–116, 2017.

[28] Akhilesh Pandey, Avanish Kumar, and Sanjay Puri. Quantum chaotic systems and random matrix theory. *arXiv e-prints*, pages arXiv–1905, 2019.

[29] Wilson John Rugh. *Nonlinear system theory*. Johns Hopkins University Press Baltimore, 1981.

[30] Akitada Sakurai, Marta P Estarellas, William J Munro, and Kae Nemoto. Quantum extreme reservoir computation utilizing scale-free networks. *Physical Review Applied*, 17(6):064044, 2022.

[31] A Sánchez-Garrido. On krylov complexity. *arXiv preprint arXiv:2407.03866*, 2024.

[32] MS Santhanam, Sanku Paul, and J Bharathi Kannan. Quantum kicked rotor and its variants: Chaos, localization and beyond. *Physics Reports*, 956:1–87, 2022.

[33] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN computer science*, 2(3):160, 2021.

[34] Maria Schuld and Francesco Petruccione. *Machine learning with quantum computers*, volume 676. Springer, 2021.

[35] Hans-Jürgen Stöckmann. Quantum chaos. *Quantum Chaos*, 2007.

[36] S.H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering*. Studies in Nonlinearity. Westview, 2000.

[37] Yudai Suzuki, Qi Gao, Ken C Pradel, Kenji Yasuoka, and Naoki Yamamoto. Natural quantum reservoir computing for temporal information processing. *Scientific reports*, 12(1):1353, 2022.

[38] Gouhei Tanaka, Toshiyuki Yamane, Jean Benoit Héroux, Ryosho Nakane, Naoki Kanazawa, Seiji Takeda, Hidetoshi Numata, Daiju Nakano, and Akira Hirose. Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123, 2019.

[39] Tomohiro Taniguchi, Sumito Tsunegi, Shinji Miwa, Keisuke Fujii, Hitoshi Kubota, and Kohei Nakajima. Reservoir computing based on spintronics technology. *Reservoir Computing: Theory, Physical Implementations, and Applications*, pages 331–360, 2021.