Breaking Implicit Assumptions of Physical Delay-Feedback Reservoir Computing

Tian Gan

Doctor of Philosophy

University of York
School of Physics, Engineering and Technology

September 2023

Abstract

The Reservoir Computing (RC) paradigm is a supervised machine learning scheme using the natural computational ability of dynamical systems. Such dynamical systems incorporate time delays showcasing intricate dynamics. This richness in dynamics, particularly the system's transient response to external stimuli makes them suitable for RC. A subset of RCs, Delay-Feedback Reservoir Computing (DFRC), is distinguished by its unique features: a system that consists of a single nonlinear node and a delay-line, with 'virtual' nodes defined along the delay-line by time-multiplexing procedure of the input. These characteristics render DFRC particularly useful for hardware integration. In this thesis, the aim is to break the implicit assumptions made in the design of physical DFRC based on Mackey-Glass dynamical system.

The first assumption we address is the performance of DFRC is not affected by the attenuation in physcial delay-line as the nodes defined along it are 'virtual'. However, our experimental results contradict this. To mitigate the impact of losses along the delay line, we propose a methodology 'Devirtualisation', which describes the procedure of directly tapping into the delay lines at the position of a 'virtual' node, rather than at the delay line's end. It trade-offs the DFRC system's read-out frequency and the quantity of output lines. Masking plays a crucial role in DFRC, as it defines 'virtual' nodes along the delay-line. The second assumption is that the mask used should randomly generated numbers uniformly distributed between $[-u, u]$. We experimentally compare Binary Weight Mask (BWM) vs. Random Weight Mask (RWM) under different scenarios; and investigate the randomness of BWM signal distribution's impact. The third implicit assumption is that, DFRC is designed to solve time series prediction tasks involving a single input and output with no external feedback. To break this assumption, we propose two approaches to mix multi-input signals into DFRC; to validate these approaches, a novel task for DFRC that inherently necessitates multiple inputs: the control of a forced Van der Pol oscillator system, is proposed.

Acknowledgements

Writing this thesis has been one of the most challenging and rewarding endeavors of my life. I could not have reached the end of this journey without the unwavering support of numerous individuals.

First and foremost, I wish to express my deepest gratitude to my supervisors, Prof. Martin Trefzer and Prof. Susan Stepney. Their expertise, guidance, patience, and mentorship have been instrumental in shaping my research. I am continuously inspired by their passion and dedication to academic excellence. Their feedback and encouragement have pushed me to heights I never imagined possible.

To my loving parents, Quan Gan and Yanxia Wang: thank you for instilling in me the values of hard work and perseverance. Your unwavering belief in my abilities, even when I doubted myself, has been my anchor throughout this journey. I also want to thank my grandparents Xingyi Wang, Shuzhi Yang, Fuzhou Gan, Min Wang; my aunt Yanli Wang and uncle-in-law Nils Christian Ellingsen, for their constant encouragement and for always being there with a listening ear and comforting words. I dedicate this achievement to you.

To my dearest partner, Yuyao Wang, this journey would not have been the same without you. Thank you for being my soulmate.

Lastly, to my friends: Hui-Ting, Junbo, Tianyuan, Yunlong, your support, in its myriad forms – particularly our discussions over take away dinners and Mahjoon sessions during weekends – has been invaluable. To my colleagues from BIST, and everyone who has been a part of this journey in any capacity, thank you.

In reflection, this journey has been a collaborative effort, and I am forever grateful to all who have walked alongside me.

Declaration and Related publications

I hereby declare that this thesis represents my own original work, and I am the sole author. This work has not been submitted previously for any award at this or any other University. All sources are properly acknowledged in the References section.

Chapters 4 and 5 are based on the following conference paper.

> T. Gan, S. Stepney and M. Trefzer. "Tradeoffs with physical delay feedback reservoir computing." *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, page 1-8.

Chapter 6 is based on the following conference paper.

> T. Gan, S. Stepney and M. Trefzer. "Combining Multiple Inputs to a Delay-line Reservoir Computer: Control of a Forced Van der Pol Oscillator System" *2023 International Joint Conference on Neural Networks (IJCNN)*, page 1-7.

Contents

List of Tables

List of Figures

1 Introduction

We are living in an information-driven era where innovative information processing techniques are in high demand. It is becoming increasingly apparent that "smart" hardware is on the rise, with even phones being capable of comprehending people's language and responding appropriately to given commands, e.g., small-sized machines can detect if people are wearing masks properly during the COVID-19 pandemic.

In 1965, Gordon Moore, the co-founder of Intel, predicted that the number of transistors on a microchip would double approximately every 18 months [5, 6]. Shortly after the term 'Moore's Law' was brought up by Carver Mead from Caltech [7,8], it was eventually adopted as a benchmark for the semiconductor industry, and it has been used by competitive semiconductor manufacturers as a driving force to increase processing power until today.

Over the past half-century, we have realised that the electronics industry has undergone remarkable expansion, and it has been a crucial contributor to the worldwide technological revolution. According to Moor's Law, the semiconductor industry has prioritised shrinking transistor sizes to reduce power and cost per device. Computing capability of microprocessors has significantly increased with the increase in transistor density, leading to several new application areas, from embedded low-power to machine learning.

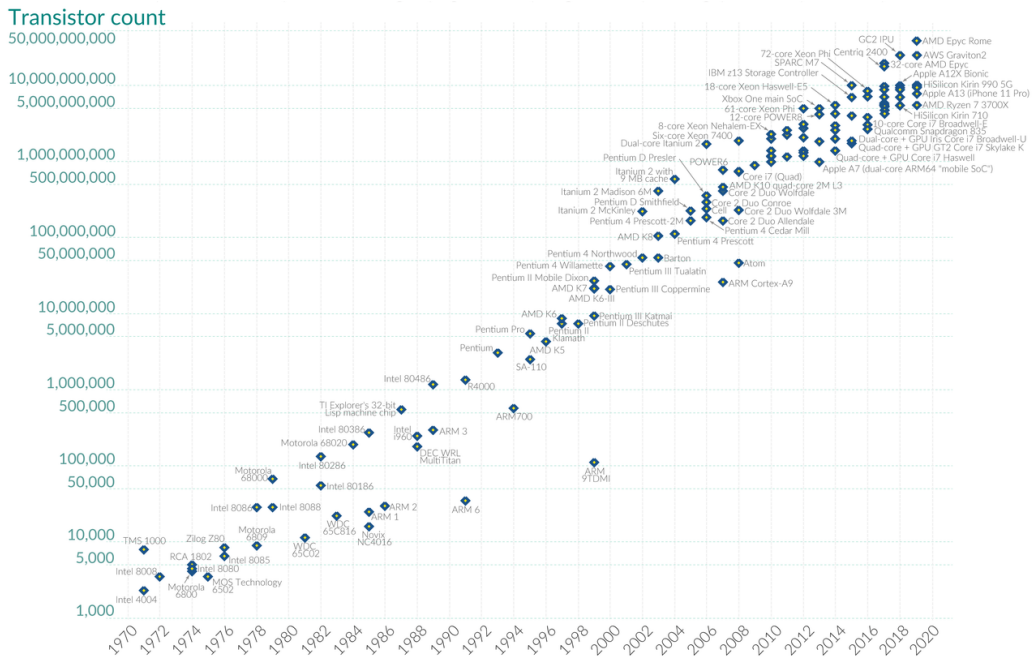


Figure 1.1: Moore's law: The number of transistors on microchips doubles every 18 months [1].

As the clock ticks away, time has swiftly moved to the year 2023, and we are witnessing an increase in the number of Artificial Intelligence (AI) techniques such as ChatGPT and MidJourney. It is reasonable to anticipate that the scaling of transistors will continue to enhance the development of emerging AI applications.

However, recent research suggests that Moore's law may have reached its end due to physical limits: simply reducing the gate length of a transistor to the point where the gate dielectric length is only a few atomic spacings or the silicon lattice constant (0.54 nm) [9–11] can lead to a significant increase in leakage current, and there is a risk of the chip short-circuiting.

Therefore, increasing the quantity of transistors has the potential to limit the development of computationally-intensive AI applications in the future.

This may necessitate a shift in focus towards alternative materials and device architectures, such as quantum computing or optoelectronic devices, to continue the pace of computational improvement. On the other hand, reaching the physical limits of miniaturisation could also trigger a paradigm shift in how we approach energy efficiency and data processing, possibly leading to more distributed, specialized hardware solutions, or a move towards more energy-efficient algorithms and software-level optimisations.

1.1 Unconventional Computing

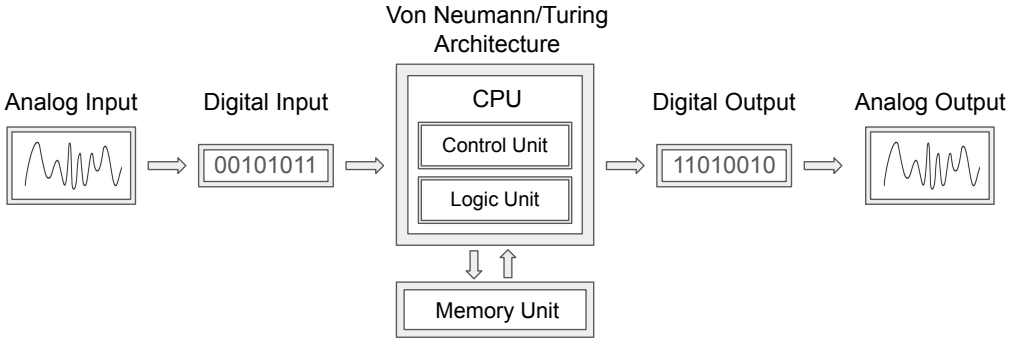
Traditional Von Neumann computer architectures and Turing techniques [12] based on semiconductor devices are effective at executing simple mathematical instructions, but they struggle with highly complicated computational tasks, such as speech recognition and facial recognition, particularly in terms of efficiency and energy consumption. In an effort to overcome the limitations, **unconventional computing** endeavors to offer alternative architectures and systems that leverage the underlying physics, chemistry and multi-scale interactions of the real world.

Our brain appears to be optimally adapted for various kinds of tasks and performs differently compared to the "binary machines". As we walk on a city street, we are continually bombarded with sensory inputs. All of these exterior stimuli cause rapid neuronal activity in the brain, allowing us to detect, e.g., a passing vehicle, professor who taught us FPGA, squirrels arguing, and the aroma of freshly baked croissants, among other things. Moreover, it can be stated that all the information perceived by our brains is in the form of analog signals.

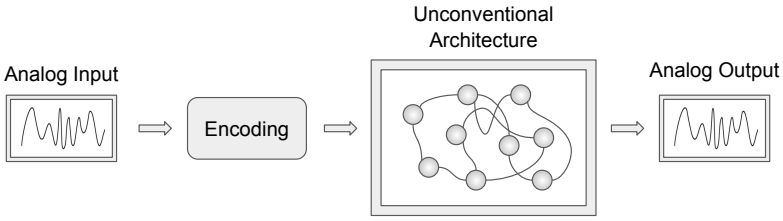
The field of unconventional computing aims to mimic the information processing methods and capabilities of the brain, which is different from current computer architecture (see Figure. 1.2). For example, in Figure. 1.2a, a Von Neumann architecture is depicted. After digitizing an analog input from the external environment, a pre-programmed computational unit processes it based on predefined instructions in a program, typically using a combination of logical blocks. After the processing, the output digital signal is converted back into an analog signal. Unconventional computing architecture, depicted in Figure. 1.2b, on the other hand, involves the use of novel and non-standard computing methods to process information. These methods are often inspired by biological or physical systems, such as the human brain [13], quantum mechanics [14], or even the behavior of slime molds [15]. Unconventional computing relies on the dynamics of the substrates, instead of converting between analog and digits. It uses analog input and output signals to process information. Unconventional computing can be implemented in various ways, including optical, chemical, biological, or mechanical systems.

1.2 Research Hypothesis and Objectives

The Reservoir Computing (RC) paradigm is an unconventional computing scheme using natural computational ability of dynamical systems. Delayed chaotic systems provide rich dynamics for information processing by using the system's transient response to an external input, so have been identified as suitable systems for reservoir computing. Delay-feedback reservoir computing ($DFRC$) with a feedback loop avoid the problem of massive in-



(a)



(b)

Figure 1.2: Von Neumann vs Unconventional architectures.

terconnections and efficiently saves area, thus can be more hardware friendly.

These properties of DFRC in the context of physical substrates leads to the formulation of the hypothesis of this thesis:

Hypothesis: By exploring the implicit assumptions currently made in the design of dynamical system based physical delay-feedback reservoir computing (*DFRC*), it is possible to achieve better computational performance and efficiency without increasing manufacturing complexity.

In this thesis, three specific assumptions made are investigaed and formulated as three sub-hypothesises.

Assumption 1: The nodes defined along the delay-line in DFRC are considered 'virtual', meaning that the performance of reservoir computing isn't affected by physical effects, e.g., the attenuation, that usually happen in physical delay-lines.

Assumption 2: Time-multiplexing plays an important role in DFRC as it defines the virtual nodes along the delay-line. The mask used should be comprised of randomly generated numbers uniformly distributed between $[-u, u]$.

Assumption 3: DFRC is typically used to solve time series prediction tasks involving a single input and output with no external feedback, i.e., different from internal delay-line of DFRC.

To test these hypotheses, the following objectives will be accomplished:

Objective 1: Demonstrate the feasibility of 'Devirtualisation' approach and investigate the performance implications and trade-offs of this approach on physical delay-lines.

Objective 2: Investigate various masking techniques such as random

mask and binary mask, with multiple configurations of uniform/non-uniform number distributions between $[-u, u]$, $[0, 2u]$, $[u, 3u]$ in relation to the performance of the delay-feedback reservoir computing system.

Objective 3: Develop a novel task, e.g. control task, that necessitates multiple inputs to operate the DFRC system and feedback to self-modulation, in order to evaluate the feasibility of using DFRC system as a controller.

1.3 Contributions

Throughout the process of this work, several contributions have been made to the academic discipline of physical Delay-Feedback Reservoir Computing (*DFRC*):

- A methodology of architecting a computational framework within Simulink, which serves as an intermediary stage transitioning from mathematical models based on delay differential equations (DDEs) to the actual construction of a physical electronic circuit.

- A methodology referred to as 'Devirtualisation' which trades-off the DFRC system's read-out frequency and the quantity of output lines, aiming to mitigate the impact of attenuation within the physical delay-line.

- The knowledge of impact on the performance of DFRC resulting from the utilisation of diverse masking techniques featuring multiple configurations of uniform/non-uniform distributions across different input ranges.

- The creation of two effective methodologies facilitating the combination of multiple inputs in the time domain: *Interleave* and *Sequential*, designed to enhance the input masking process used in DFRC.

- The creation of a novel benchmark task in the field of DFRC: control of Forced Van der Pol oscillator system which requires multi-input to DFRC. This task bridges the gap between theoretical research and practical application, ensuring that developments of DFRC system are aligned with genuine needs.

- An innovative application of using trained multi-input DFRC as a system controller, which demonstrate the ability of DFRC beyond predicting time domain data.

- The creation of a MATLAB/SIMULINK model: Mackey-Glass based DFRC system. Github: https://github.com/Tian21/MG-DFRC-model.

1.4 Thesis Outline

The structure of this thesis is outlined as follows, spanning across seven chapters:

- Chapter 2 provides a background review of Machine Learning, Neural Networks, and recent developments in Reservoir Computing, as well as Chaotic Dynamical Systems used in this work.

- Chapter 3 presents the concept of Delay-feedback Reservoir Computing (DFRC) and the methodology followed for the experimental work in this thesis, which include the modelling frameworks and the parameter

configurations. The benchmark tasks used to evaluate the performance of DFRC are introduced in this chapter.

- Chapter 4 demonstrates an approach called **Devirtualisation**. This approach navigates a trade-off between read-out frequency and the number of output lines to mitigate the effects of attenuation in the physical delay-line, thereby achieving improved performance on DFRC.

- Chapter 5, an investigation on masking, a crucial procedure of DFRC is presented. Two types of masking signals are experimentally compared: *Binary Weight Mask* and *Random Weight Mask*, which are selected implicitly in previous studies. Bias in the randomness of distribution of *Binary Weight Mask* is explored.

- Chapter 6 presents a novel task in the field of DFRC: control of Forced Van der Pol oscillator system. This task requires multiple inputs to train the DFRC, therefore, two effective methodologies: *Interleave* and *Sequential* are designed to facilitate the combination of multiple inputs in the time domain of DFRC.

- Chapter 7 concludes the thesis and discusses directions for future work.

2 Background

2.1 Machine Learning

Machine learning is a sub-field of artificial intelligence that focuses on creating algorithms and models that enable computers to learn from data and improve their performance on a specific task without being explicitly programmed [16]. The goal of machine learning is to develop algorithms that can automatically identify patterns and make predictions or decisions based on data [17].

Machine learning involves different types of techniques, including supervised learning, unsupervised learning, and reinforcement learning [18]. It is used in a wide range of applications, such as image recognition [19], natural language processing [20], recommendation systems [21], fraud detection [22], and many more.

For instance, a machine learning algorithm can be trained to recognize alphabetic handwritten characters by providing the computer with some handwritten samples during a training process [23]. The machine will learn the patterns and can then recognize unseen samples of handwritten characters during a testing process. The algorithm must be able to generalize to samples that were not present during the training process in order to be useful [24].

Machine learning algorithms are also used in data mining, such as in medical records, to detect trends and improve medical practice [25]. They are commonly used in online transactions to detect fraud and make product recommendations based on user preferences [26]. The core of machine learning is concerned with the representation of data samples and generalization to new, unseen data [27]. The sub-field of computational learning theory studies the conditions under which generalization can be guaranteed [28].

Overall, machine learning is a rapidly growing field with many potential applications in various industries. As more data becomes available, machine learning algorithms are becoming increasingly sophisticated and powerful [29]. However, the success of machine learning relies on the availability of high-quality data and optimised algorithms that are suited for the specific task at hand [30].

In addition, machine learning algorithms must be carefully evaluated to ensure that they are accurate, reliable, and do not exhibit bias or discrimination [28]. This is particularly important in fields such as healthcare, finance and autonomous vehicles, where decisions based on machine learning can have significant real-world consequences [30].

Despite these challenges, machine learning has the potential to revolutionize many aspects of our lives and drive significant progress in fields such as healthcare, transportation, and energy [31]. A brief literature of some typical branches of Machine Learning: Bayesian Statistics, Supervised Learning, Unsupervised Learning and Online Learning is given in the following sections.

2.1.1 Supervised Learning

Supervised learning is a fundamental machine learning approach where a model is trained on labeled data, consisting of input-output pairs, to learn the underlying relationship between inputs and outputs [27]. The primary objective of supervised learning is to build a model that can generalize well to unseen data, making accurate predictions on new inputs. The process involves minimizing a loss function that quantifies the discrepancy between the model's predictions and the actual outputs. Supervised learning has been extensively studied and applied in various domains, including computer vision, natural language processing, speech recognition, and medical diagnosis.

One of the core techniques in supervised learning is linear regression, which seeks to model the relationship between input features and a continuous output variable using a linear combination of the input features [32]. Linear regression models are simple, interpretable, and computationally efficient, but they may not capture complex relationships in the data. To address this limitation, researchers have developed more flexible models, such as polynomial regression and basis function expansions, which can model non-linear relationships between inputs and outputs.

Classification is another central problem in supervised learning, where the goal is to assign input data to one of several discrete classes [33]. Many classification algorithms have been proposed, including logistic regression, *support vector machines* (SVM), and decision trees. Logistic regression models the probability of class membership using a logistic function and is particularly useful for binary classification tasks [34]. Support vector machines aim to find the optimal decision boundary that maximizes the margin between

classes, providing robust classification even when classes are not linearly separable [35]. Decision trees recursively partition the input space based on feature values, leading to a hierarchical structure that is easy to interpret and can handle both categorical and continuous inputs [36].

In recent years, deep learning has emerged as a dominant technique in supervised learning, particularly for high-dimensional and complex data, such as images, text, and speech [37]. Deep learning methods, like *convolutional neural networks* (CNNs) and *recurrent neural networks* (RNNs), leverage the hierarchical representation of data to learn abstract and meaningful features automatically. CNNs have been particularly successful in computer vision tasks, such as image classification, object detection, and semantic segmentation [19, 38, 39]. RNNs, on the other hand, excel at processing sequential data and have been widely applied in natural language processing and speech recognition [40, 41], which is discussed in detail in Section 2.2.

Ensemble methods have gained popularity in supervised learning due to their ability to improve the performance of individual models by combining their predictions [42]. Techniques like bagging, boosting, and stacking can be employed to construct diverse ensembles of models, often leading to better generalization and more accurate predictions. Random forests, an extension of decision trees, use bagging to construct an ensemble of trees and aggregate their predictions through majority voting [43].

Feature selection and dimensionality reduction play crucial roles in supervised learning, as they help to identify relevant features and reduce computational complexity [44]. Techniques such as filter methods, wrapper methods, and embedded methods can be used to select a subset of features that

contribute most to the model's performance. Filter methods evaluate feature importance using measures like information gain, correlation, or mutual information, and select a subset of features independently of the learning algorithm [45]. Embedded methods integrate feature selection within the learning algorithm, as seen in LASSO and decision trees, where feature selection occurs during model training [46, 47].

Evaluating the performance of supervised learning models is essential for understanding their effectiveness and comparing different algorithms [48]. Cross-validation, a common evaluation technique, involves partitioning the dataset into multiple subsets and training the model on different combinations of these subsets while testing on the remaining data. This approach helps to obtain a more reliable estimate of the model's generalization performance. Common performance metrics include accuracy, precision, recall, and F1-score for classification tasks, and *mean squared error* (MSE), *mean absolute error* (MAE), and R-squared for regression tasks.

2.1.2 Unsupervised Learning

Unsupervised learning is a machine learning paradigm that aims to learn from data without explicit labels or targets, often revealing complex structures and relationships within datasets [49]. The field has rapidly evolved, with various techniques and methods proposed in the literature. This subsection provides an overview of the core concepts, techniques, and applications of unsupervised learning, along with the challenges and potential future directions in the field.

A range of techniques and methods have been proposed for unsupervised

learning, with clustering and dimensionality reduction being the most prominent. Clustering algorithms, such as K-means [50], DBSCAN [51], and hierarchical clustering [52], are used to group similar data points based on their features. Dimensionality reduction techniques, including Principal Component Analysis [53], *t-Distributed Stochastic Neighbor Embedding* (t-SNE) [54], and autoencoders [55], aim to reduce the number of features while preserving the structure of the data. Recent developments in generative models, such as *Variational Autoencoders* (VAEs) [56] and *Generative Adversarial Networks* (GANs) [18], have also contributed significantly to the advancement of unsupervised learning.

Unsupervised learning has found applications across various domains, including anomaly detection [57], recommender systems [58], and natural language processing (NLP) [59]. unsupervised learning's ability to identify unusual patterns in data has been instrumental in detecting fraud, network intrusions, and other anomalies. In recommender systems, unsupervised learning techniques are employed to group similar users or items and make personalised recommendations. Additionally, unsupervised learning has improved various NLP tasks, such as topic modeling [60], sentiment analysis [61] and word embeddings [59].

Despite its progress, unsupervised learning faces several challenges. Evaluating unsupervised learning models can be difficult due to the absence of ground truth labels, making the development of appropriate evaluation metrics a crucial area of research [62]. Scalability remains an ongoing challenge, with recent research focusing on parallelization and distributed computing techniques to handle large datasets [63]. Furthermore, understanding the

inner workings of unsupervised learning models, particularly those based on deep learning, is essential for ensuring trust and adoption in various applications [64].

2.2 Artificial Neural Networks

In recent years, *artificial neural networks* (ANNs) have seen tremendous development, propelling progress in disciplines such as computer vision, natural language processing, and reinforcement learning. Inspired by the biological neural networks present in the human brain, ANNs are designed to learn patterns from huge volumes of data via nodes called neurons that are interconnected [18]. Recent advancements in deep learning techniques, which include stacking many layers of neurons, have enhanced the ability of ANNs to learn complicated patterns and representations [37]. GPT-4, a recent example, displays the capability of large-scale language models to generate text that resembles human language [65].

In spite of their achievements, ANNs continue to confront obstacles, including as interpretability, resource constraints, and robustness. Researchers are currently pursuing solutions, such as explainable AI (XAI) approaches, to increase the transparency of these models' decision-making [66]. As the field of ANNs advances, it will undoubtedly play a significant role in moulding the future of artificial intelligence and its applications.

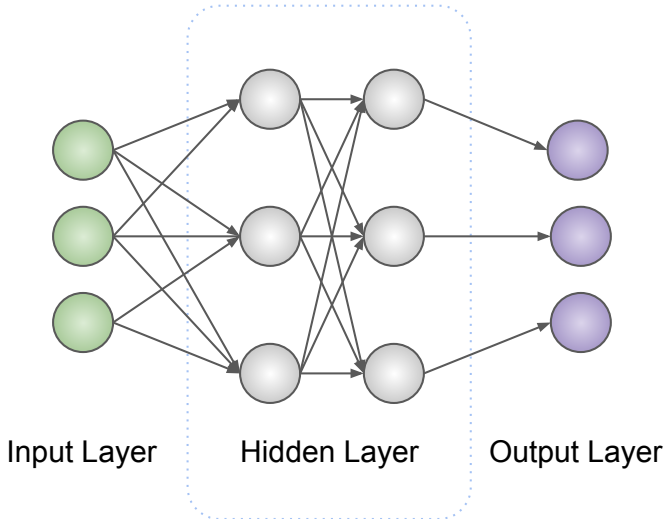


Figure 2.1: Feed-forward Neural Network

2.2.1 Spiking Neural Network

Spiking Neural Networks (SNNs) [67], influenced by the structure of brain circuits, form a group of models designed for neuromorphic computing. Within an SNN model, individual neurons adjust their membrane potential using remembered states and current inputs. These neurons emit spikes upon the membrane potential surpassing a certain threshold. Unlike Artificial Neural Networks (ANNs) with continuous input signals, communication between spiking neurons occurs through binary spike sequences. As a result, SNN models preserve both spatial and temporal information.

2.2.2 Feed-forward Neural Network

ANNs can be structured as feed-forward networks, which do not contain any internal loops that allow signals to pass through the same neuron multiple times. Instead, signals only travel forward through the network, passing

through separate layers of neurons at discrete time steps [18], as shown in Figure 2.1. The basic building block of this type of network is the artificial neuron, which receives input, performs a computation on that input, and produces an output. These neurons are arranged in layers, with the input layer at the front, one or more hidden layers in the middle, and the output layer at the back. The first and last layers of the network are responsible for interacting with the outside world, while the layers in between are known as hidden layers. Feed-forward networks can be trained using linear algorithms like back-propagation [68] until the input examples are correctly classified or a stopping criterion is met. However, because they do not process temporal information, feed-forward networks are only capable of interpreting information from a single moment in the input history and were originally designed to process static spatial patterns of inputs [69]. The connections between the neurons are represented by weights, which are adjusted during training to minimize the error between the network's output and the desired output [70].

2.2.3 Recurrent Neural Network

Recurrent Neural Networks (RNNs), on the other hand, are a type of artificial neural network that are well-suited to processing sequential data, such as time series, speech, and text [71]. They have been widely studied in the literature [18] and have been applied to a wide range of tasks, including speech recognition [41], natural language processing [40] and time series prediction [72].

One of the key features of RNNs is their ability to maintain internal

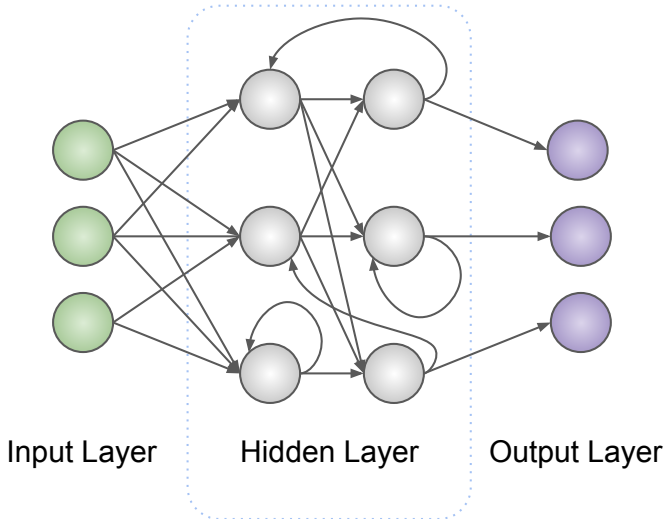


Figure 2.2: Recurrent Neural Network

states, which allows them to process input sequences of varying lengths and to capture temporal dependencies in the data. This is achieved by introducing feedback connections between the neurons, which allow information to flow through the network across multiple time steps [73].

There are several types of RNNs, including the basic RNN, the *Long Short-Term Memory* (LSTM) network [73], and the *Gated Recurrent Unit* (GRU) network [74]. Each of these architectures has its own strengths and weaknesses, and the choice of architecture will depend on the specific task and the nature of the data.

In recent years, there has been a growing interest in using RNNs for natural language processing tasks, such as machine translation [75], text generation [76], and sentiment analysis [77]. These tasks involve processing sequential data with complex structures, such as sentences and paragraphs, and RNNs have been shown to be particularly effective for these types of

tasks.

There have also been many studies on the use of RNNs in time series prediction, such as stock market prediction [78], weather forecasting [79], and traffic prediction [80]. RNNs have shown to be effective in these types of tasks, as they can capture temporal dependencies in the data and make predictions based on historical patterns.

In general, RNNs are a powerful tool for processing sequential data, and their ability to capture temporal dependencies in the data makes them well-suited to a wide range of tasks. However, training RNNs can be challenging due to the presence of gradients with long-term dependencies, which can make the optimization process difficult [81].

2.3 Neuroscience

The rapid progress in both neuroscience and computer technology highlights the remarkable computational power of the human brain. With its exceptional cognitive capabilities and energy efficiency, the brain consists of approximately 10^{11} neurons and 10^{15} synapses, forming a highly intricate and not yet fully understood network [82–84].

Reservoir computing, an approach to machine learning, takes inspiration from the brain's information processing methods. This concept posits that neurons form a complex network influenced by external stimuli, generating persistent network activity that enables information processing based on prior stimuli. This network activity is then transmitted to other brain regions responsible for interpreting or classifying outputs. Consequently, reservoir computing aims to mimic the brain's information processing strategies by

using complex neural networks.

The idea that a simple linear classifier can decipher elaborate computations from distributed neural activity is undoubtedly relevant to the brain's physical processes. The brain consists of a sophisticated network of specialised structures, each carrying out unique functions. As a result, it is worth considering whether reservoir computing serves as a fitting analogy for brain-inspired computing or if it simply provides an alternative approach to using neural networks and improving their trainability. Additionally, when viewing the brain's regions as reservoirs from an evolutionary standpoint, how do these specialised areas evolve if their development does not exclusively rely on learning?

The bio-inspired foundation of reservoir computing encourages the development of concepts like the echo state network and liquid state machine, the original ideas in the field [85, 86]. This perspective also implies that advancements in reservoir computing may foster progress in neuroscience by offering novel ways to model the brain's information processing mechanisms. Thus, the relationship between neuroscience and reservoir computing is mutually beneficial, as each domain has the potential to inform and advance the other.

2.4 Reservoir Computing

Reservoir computing (RC) is a machine learning approach that was developed based on the functioning of the human brain. It originated about 20 years ago and was introduced through the Echo State Network (ESN) by Jaeger in 2001 [85] and the Liquid State Machine (LSM) by Maass in 2002 [87]. Later, similarities between ESN and LSM were demonstrated by Verstraeten

and his team, bringing the two under the RC framework [88]. RC uses randomly connected and recurrent nonlinear nodes in the reservoir layer to make Recurrent Neural Networks (RNNs) more efficient and cost-effective to implement [89].

The reservoir layer creates complex dynamics that map input data into higher-dimensional spatiotemporal patterns, making it easier to differentiate state vectors for different classes. RC excels at handling input data that changes over time, due to the recurrent connections that establish links between current and past neuron dynamics, also known as short-term or fading memory.

An RC substrate may be a physical device or material, a simulated network, or a set of equations. The characteristics of a reservoir computer rely on the underlying dynamics of the substrate a reservoir system is created with [90, 91]. The essential characteristics that physical reservoirs must possess in order to correctly perform various functions are:

1. *High-dimensionality and non-linearity.* This is depending on the quantity of distinct signals retrieved from the reservoir. If the reservoir includes a significant number of nonlinear nodes, the projection of the input data onto the reservoir is functionally equal to a mapping into a high-dimensional space. Hence, the nonlinear mapping transforms non-separable inputs into separable ones.

2. *Fading memory.* This attribute is critical for processing sequential data, since the state of a reservoir is reliant on the recent past signal but independent of the distant past, i.e. its response is dependent on relevant input, but does not become noise over time [92].

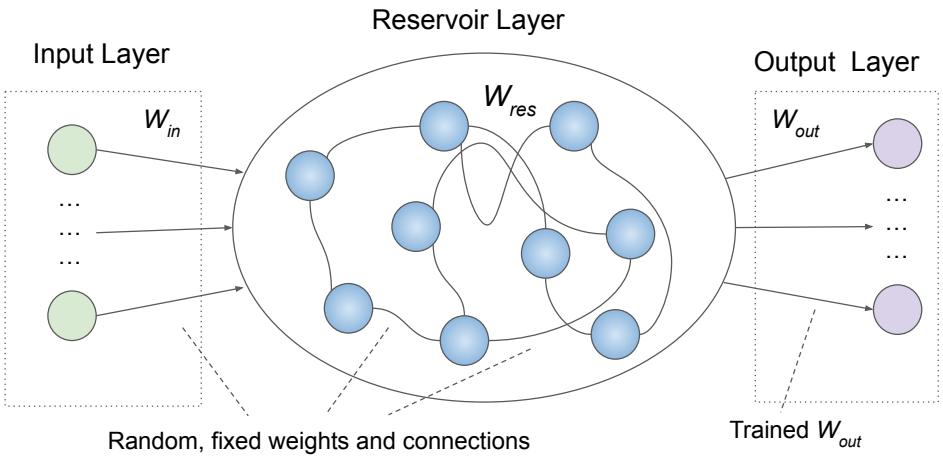


Figure 2.3: Classic Reservoir Computing Schematic.

A classic RC network (see Figure.2.3) consists of an input layer, a reservoir, and an output layer. In a network with $d-dimensional$ input, $l-dimensional$ output, and N neurons, only the coefficients between the output and the reservoir ($W_{out} \in \mathbf{R}^{l \times N}$) need to be trained through linear regression. Conversely, the input coefficients ($W_{in} \in \mathbf{R}^{N \times d}$) and reservoir coefficients ($W_{res} \in \mathbf{R}^{N \times N}$) are generated randomly. The reservoir's complex dynamics and nonlinear transformations map input data into higher-dimensional spaces for classification or prediction purposes, and internal feedback retains past neuron states in fading memory, influencing computations at the current state [93, 94].

2.4.1 Echo State Network

Echo State Networks (ESNs) are a type of *recurrent neural network* (RNN) that has gained significant attention in recent years due to their promising capabilities in handling complex time-series data and their efficient learn-

ing process. As a subset of reservoir computing, ESNs are characterised by their random, fixed recurrent connections, and the training of only output weights [85]. The ESN model has been successfully applied to a variety of tasks, such as speech recognition, time-series prediction, and control tasks, showcasing its potential for solving real-world problems [95–97].

The formulation of ESN model is originally given in [85], without leakiness term and output feedback, can be described as follow:

$$x(t) = f(W_u u(t) + W_x x(t-1)) \tag{2.1}$$

$$v(t) = W_v x(t) \tag{2.2}$$

where the state of the reservoir at time t is denoted as $x(t)$, and the reservoir weight matrix is denoted as W. The input at time t is denoted as $u(t)$, and the input weight matrix is denoted as W_u. The activation function used in ESNs is typically a nonlinear function such as the hyperbolic tangent (tanh) function, denoted as $f()$. The output of the ESN at time t is denoted as $v(t)$, and the output weight matrix is denoted as W_v.

From real-time systems view, Stepney in paper [98] demonstrates the 'Physical' equation of Echo State Network:

$$x(t+1) = f(W_x x(t) + W_u u(t)) \tag{2.3}$$

$$v(t+1) = W_v x(t) \tag{2.4}$$

The input u and the state x in a reservoir computing system are sampled at different times within a time unit. Specifically, the input is sampled at the

beginning of the time unit, while the state is sampled at the end of the time unit. The state update equation (Eq. 2.1) reflects this, as it states that the state at the end of time unit t, $x(t)$, is a function of the state at the end of time unit $t-1$, $x(t-1)$, as well as the effect of the input at the beginning of time unit t, $u(t)$. However, the actual time when u and x are sampled to calculate the update is the same, because the end of time unit $t-1$ is equivalent to the beginning of time unit t [98].

ESN architecture typically comprises an input layer, a reservoir (or hidden) layer, and an output layer. The reservoir is composed of recurrently connected neurons, forming a dynamic system with a rich temporal context [85, 88]. During training, the input and reservoir weights are initialised randomly, and only the output weights are adjusted. This makes the learning process more efficient as it avoids the need for computationally expensive algorithms such as *backpropagation through time* (BPTT) or *real-time recurrent learning* (RTRL) [99, 100].

ESNs have been employed in various applications, including time-series prediction, speech recognition, and robot control, demonstrating their versatility and effectiveness [95–97]. For instance, ESNs have shown great promise in nonlinear system identification and chaotic time-series prediction, outperforming other methods like support vector machines and feed-forward neural networks [95]. In speech recognition tasks, ESNs have successfully modeled the temporal structure of speech signals, achieving competitive results compared to other machine learning techniques [96].

Recent advancements in ESN research have led to the development of various modifications and extensions of the original model. Deep ESNs, for

example, incorporate multiple reservoir layers to capture hierarchical representations of the input data, resulting in improved performance in complex tasks [101]. Other variations include the Leaky Integrator ESN, which introduces a leakage term to the neuron activation function, enabling the network to better adapt to different time scales [102]. The neurons have a decay parameter α, also known as a leaking rate, that can regulate the rate at which the reservoir's update dynamics occur. The state update equation for leaky-integrator neurons with real-time interpretation [98] is expressed as:

$$x(n+1) = (1-\alpha)x(n) + \alpha f(W_{in}u(n) + Wx(n)) \tag{2.5}$$

Furthermore, researchers have explored the use of different types of reservoirs, such as those based on spiking neural networks, to expand ESN's applicability in diverse domains [103].

Furthermore, the study of ESNs' inherent properties, such as stability, robustness, and generalization capabilities, is crucial for a better understanding of their strengths and limitations, ultimately leading to more refined models and applications [102, 104]. Finally, as the demand for energy-efficient and low-latency machine learning solutions increases, researchers should explore hardware implementations of ESNs and other reservoir computing approaches, leveraging emerging technologies like neuromorphic computing and memristive devices [105, 106].

Echo State Networks have emerged as a powerful and efficient approach for handling time-series data and other dynamic processes [107]. Their unique architecture, characterised by random fixed recurrent connections and trainable output weights, has led to their successful application in various do-

mains, such as time-series prediction [100], speech recognition [108], and robot control [96]. Recent advancements have focused on optimizing reservoir properties, exploring different types of reservoirs, and developing variations of the original model to further improve ESN performance. Future research directions include the investigation of new training algorithms, the development of more biologically plausible reservoirs, and the exploration of ESNs for the exploration of ESNs for novel applications and domains [108–110].

2.4.2 Liquid State Machine

The Liquid State Machine (LSM) referenced in [87] is a form of reservoir computing implemented with a spiking neural network (detail of SNN refer Section 2.2.1). LSM consists a vast array of units, often referred to as nodes or neurons. Every node in this system not only receives fluctuating inputs from external entities but also from fellow nodes. The connections between these nodes are random. Their inherent ability to capture spatiotemporal patterns in data has led to their successful application in various domains, including speech recognition, pattern classification, and neuroprosthetics [88, 107, 111].

One notable aspect of LSMs is their adaptability to different network architectures and neuron models [112]. Researchers have explored different types of spiking neurons, such as the *Leaky Integrate-and-Fire* (LIF) and the *Hodgkin-Huxley* (HH) models, to create more biologically inspired LSMs [113, 114]. Moreover, recent advancements in neuromorphic hardware have facilitated the implementation of LSMs in more energy-efficient and real-time processing systems, enhancing their suitability for edge computing and other resource-constrained environments [115, 116].

The potential of LSMs for large-scale networks and their compatibility with emerging neuromorphic technologies has opened new avenues for research and development. Recent work has focused on improving the performance, robustness, and efficiency of LSMs, as well as exploring novel learning algorithms for spiking neural networks [117, 118]. As our understanding of biological neural networks continues to deepen, LSMs are expected to play a crucial role in advancing both the fields of reservoir computing and computational neuroscience.

2.4.3 Reservoir Training

Typically, two techniques are employed for the training of reservoir outputs, referred to as linear readouts. The first technique is termed "off-line" training, wherein the reservoir states are assembled into a matrix X over a duration of T for training purposes, and simple linear or ridge regression techniques are used for a single training iteration. The second approach, termed "on-line" training, typically uses a gradient descent-based algorithm such as *Recursive Least Squares* (RLS) or *Adaptive Moment Estimation* (Adam).

Off-line Training

Off-line training, also known as batch-mode training, is a method employed in reservoir computing for training the output weights of the reservoir. This approach involves driving the reservoir with the input signal, $u(t)$, and recording the corresponding reservoir states, $x(t)$, during a training phase. The reservoir states are collected into a matrix X, with each column representing the reservoir state at a specific time step for the entire training length, T.

Simultaneously, the desired output signal, $y_{\text{target}}(t)$, is collected into a vector Y for the same training length, T.

Mathematically, the matrices can be represented as:

$$X = [x(1), x(2), \ldots, x(T)] \tag{2.6}$$

$$Y = [y_{\text{target}}(1), y_{\text{target}}(2), \ldots, y_{\text{target}}(T)] \tag{2.7}$$

Once the reservoir states matrix X and the desired output vector Y have been collected, the readout weights, W_{out}, we can find the optimal weights that minimise the error between $y(n)$ and y_{target} by solving the overdetermined system:

$$Y_{target} = W_{\text{out}} X \tag{2.8}$$

where W_{out} can be trained using linear regression or ridge regression techniques. Linear regression aims to minimize the *mean squared error* (MSE) between the predicted output, $y(n)$, and the desired output, $y_{\text{target}}(n)$:

$$E(Y, Y_{\text{target}}) = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\frac{(Y(n) - Y_{\text{target}}(n))^2}{\sigma^2(Y_{\text{target}}(n))}} \tag{2.9}$$

In matrix form, Eq. 2.8 can be solved by the linear regression using Ordinary Least Square, which can be written as:

$$W_{\text{out}} = Y_{\text{target}} X^{\text{T}} (X X^{\text{T}})^{-1} \tag{2.10}$$

However, this methods typically encounters problems related to stability when inverting the matrix $(X X^{\text{T}})$.

According to Lukǒsevǐcius [119], it is recommended to use either ridge regression, which involves Tikhonov regularization with a parameter β (as shown in Eq. 2.11), or the Moore-Penrose pseudo-inverse (as shown in Eq. 2.12). Ridge regression is generally preferred as it provides a stable and effective solution. The regularization parameter helps to address the issue of producing excessively large output weights, which can indicate unstable and overly sensitive solutions. The expression for ridge regression with Tikhonov regularization is presented below.

$$W_{\text{out}} = Y_{\text{target}} X^{\text{T}} (X X^{\text{T}} + \lambda I)^{-1} \tag{2.11}$$

Where λ is a regularization parameter (set to zero for simple linear regression) and I is the identity matrix. Ridge regression is employed when the system is ill-conditioned or to prevent overfitting by introducing the regularization term λ.

In some cases, the pseudo-inverse method is used for training due to its ease of implementation in certain programming environments, such as MATLAB. However, the computational cost of the pseudo-inverse method increases for large matrices of X and is typically used when the system is overdetermined. In the case of reservoir computing, the networks are generally comprised of relatively small matrices and overfitting is dependent on the complexity of the task at hand. The training equation for the pseudo-inverse method is provided below.

$$W_{out} = Y_{\text{target}} X^{+} \tag{2.12}$$

where + indicates the pseudo-inverse function in MATLAB.

Off-line training in reservoir computing is advantageous due to its simplicity and efficiency in readout weight optimization, as it does not require iterative updates during the training process. After obtaining the readout weights, W_{out}, the reservoir model can be used to predict the output for new input signals.

Pros:

- Stability: Offline training methods can be more stable because they learn from an entire batch of data at once, reducing the influence of individual outliers or noisy data points.
- Faster Convergence: They often converge faster than online methods as they get a "broader view" of the data.
- Better Performance: Since offline training uses the whole dataset, it can often lead to a more accurate and reliable model, assuming that the training dataset is representative of the overall population.

Cons:

- Memory Intensive: Batch methods can be memory-intensive as they require the entire dataset to be loaded into memory.
- Lack of Real-time Adaptability: Offline training methods might not be suitable for applications where the model needs to continuously adapt to new data.
- Computational Expense: Training on large batches or the entire dataset can be computationally expensive and time-consuming.

On-line Training

On-line training in reservoir computing is a dynamic method for training the output weights of a reservoir model. Unlike off-line training, which uses batch-mode processing to compute the optimal output weights after collecting all reservoir states, on-line training employs iterative algorithms, such as *Recursive Least Squares* (RLS) or *Adaptive Moment Estimation* (Adam), to update the readout weights during the training phase [120]. This approach enables the reservoir model to adapt to changes in the input signal and to refine its predictions in real time, which is particularly beneficial in the context of time-varying or non-stationary input signals.

Adam is a popular optimization algorithm in online learning that has gained considerable attention due to its ability to adaptively update learning rates for individual parameters [121]. By combining the advantages of momentum-based optimization and adaptive learning rates, Adam achieves faster convergence and improved performance on various machine learning tasks. In online learning scenarios, Adam demonstrates robustness in handling noisy and sparse gradients, making it a preferred choice for large-scale and real-time applications. While Adam has proven to be effective, it also faces challenges, such as its sensitivity to hyper-parameters and the need for further improvements in convergence guarantees.

Similar to "off-line" training, the RLS algorithm is normally used to minimize the MSE between the predicted output, $y(t)$, and the desired output, $y_{\text{target}}(t)$ (See Eq. 2.9). At each time step, the RLS algorithm updates the readout weights, W_{out}, based on the current reservoir state, $x(t)$, and the desired output, $y_{\text{target}}(t)$. The algorithm uses a forgetting factor, λ, to give more

importance to recent observations and an inverse correlation matrix, $P(t)$, to track the information from previous time steps. According to [85,89,120,122], the on-line training method with derived RLS update equations are described as follows:

1. **Initialization**: Before starting the online training process, initialize the output weight matrix W_{out} with random or zero values. Also, initialize the inverse correlation matrix $P(t)$, typically with an identity matrix scaled by a large value (e.g., $P(0) = \delta I$, where δ is a large scalar). Set an appropriate forgetting factor λ, typically close to 1 (e.g., 0.99), which determines how fast the model forgets past data.

2. **Input processing**: For each incoming input data point $u(t)$, calculate the corresponding reservoir state $x(t)$ using the reservoir dynamics.

3. **Output prediction**: Compute the predicted output $y(t)$ using the current output weight matrix W_{out} and the reservoir state $x(t)$ as follows: $y(t) = W_{out}x(t)$.

5. **RLS update**: Update the output weight matrix W_{out} using the RLS algorithm. First, compute the Kalman gain vector.

$$k(t) = \frac{\rho(t-1) * x(t)}{\lambda + x(t)'\rho(t-1) * x(t)} \tag{2.13}$$

Then, update the output weight matrix:

$$W_{\text{out}}(t) = W_{\text{out}}(t-1) + k(t) * (y_{\text{target}}(t) - y(t)) \tag{2.14}$$

Finally, update the inverse correlation matrix:

$$\rho(t) = \frac{1}{\lambda} * (\rho(t-1) - k(t) * x(t)'\rho(t-1)) \tag{2.15}$$

This process continues iteratively throughout the entire training length, T, allowing the reservoir model to adapt to changes in the input signal and refine its predictions during the training phase.

On-line training in reservoir computing offers several advantages over off-line training methods. The ability to adapt to time-varying or non-stationary input signals makes on-line training particularly well-suited for applications where real-time learning is critical. Additionally, the RLS algorithm can provide faster convergence and better tracking of time-varying systems compared to batch-mode linear regression techniques.

Pros:

- Adaptability: Online training allows the model to learn and adapt continuously as new data comes in, which is a great advantage when dealing with time-series or non-stationary data.

- Memory Efficiency: Since online learning involves training on one data point at a time, it can be more memory-efficient than batch learning. This is important when dealing with large datasets that may not fit into memory.

- Real-Time Updates: This method is especially suitable for real-time applications where you want the model to instantly react to new data.

Cons:

- Noise Sensitivity: Online learning can be more sensitive to noise and outliers in the data, as each data point can significantly sway the model.

- Difficulty in Convergence: Online training methods may converge slower than their batch counterparts as they try to learn from every single data point.

- Overfitting Risk: Due to the nature of learning from every data point, there's a risk of overfitting, particularly with noisy data.

2.4.4 Reservoir Metrics

Reservoir Computing are normally evaluated with a various of computational benchmark tasks, and each could yield significantly different result. While these benchmarks are beneficial in assessing the capability of a reservoir to solve a specific task, they provide a little understanding of the reservoir's dynamic behaviors. To enable a more precise evaluation and characterisation of a reservoir computing, a set of task-independent metrics can be computed for a specific system.

Linear Memory Capacity

Linear Memory Capacity (LMC) was first outlined in [86] to quantify the echo property of reservoirs. It is a measure of how well a reservoir can recall the past inputs. For the echo state property to be maintained, it's necessary that the dynamics of the input-driven reservoir eventually eliminate any information derived from initial conditions. This property suggests the existence of a fading memory, which is defined by the LMC. The process

of determining the LMC of a reservoir involves feeding a random uniform distribution of numbers into the reservoir.

The equation of calculating Linear Memory Capacity can be expressed as:

$$MC = \sum_{i=1}^{2N} \frac{\text{cov}^2(u(k-i), y(k))}{\sigma^2(u(k))\sigma^2(y(k))} \tag{2.16}$$

The output is trained to retrieve the prior inputs, denoted as $u(k-i)$, where i ranges from 1 to $2N$. N indicates the total number of nodes within the reservoir. This procedure resulting in the generation of i outputs. Jaeger [86] demonstrates that the maximum memory capacity of a system is MC $\leq$ N.

Kernel Quality and Generalisation Rank

Kernel Quality (KQ) was first introduced by Legenstein and Maass [123]. It is a measure of the reservoir's ability to produce a rich nonlinear representation of input u and its history $u(t-1), u(t-2)$,.... This can be viewed as the system's dimensionality or its ability to effectively distinguish unique input patterns.

As outlined in [124], the evaluation of KR involves a process where the rank r of a matrix M (with dimensions $n \times m$) is determined. The matrix M is constructed using multiple distinct input streams u_i through u_m, leading to the collection of reservoir states x_{u_i}. These states are organized into columns within matrix M, and this entire procedure is repeated m times. By means of Singular Value Decomposition (SVD), the rank r of matrix M is calculated, which corresponds to the number of non-zero diagonal entries present in the unitary matrix.

Generalisation Rank (GR) in Reservoir Computing refers to its ability

to generalise similar input streams. The calculation of GR is using the same ranking measure as the evaluation of KQ. However, each input stream $u_{i+1}, \ldots, u_m$ constitutes a noisy variant of the original u_i. A balance between rich dynamics and stability determines the generalisation capability of RC system [123].

In general, it is believed that a good reservoir comes with a high KQ rank and a low GR [123]. However, the optimal equilibrium might differ based on the specific task that the reservoir is handling. While these two metrics hold significance, each of themselves does not encompass enough amount of information regarding the dynamic properties of the reservoir [125].

2.4.5 Reservoir with Physical Devices and Systems

In the last decade, there has been a growing interest in exploring the potential of RC for creating more efficient information processing by using physical dynamics as computational resources. In software-based RC, the reservoir layer is used to nonlinearly map the current input and a bit of past inputs into a high-dimensional space. This process is being replicated in the field of physical RC, where researchers are seeking physical objects or systems that can perform a similar high-dimensional mapping, making it a significant area of study in neuromorphic computing [126–129]. Various substrates proposed for physical RC implementation (Fig. 2.4).

One of the key areas of interest in physical RC is the use of nanomagnetic ring arrays, in which, the Artificial Spin Ice (ASI) demonstrates as a promising substrate as RC due to its rich dynamics under considerable control and fesibility [130, 131].

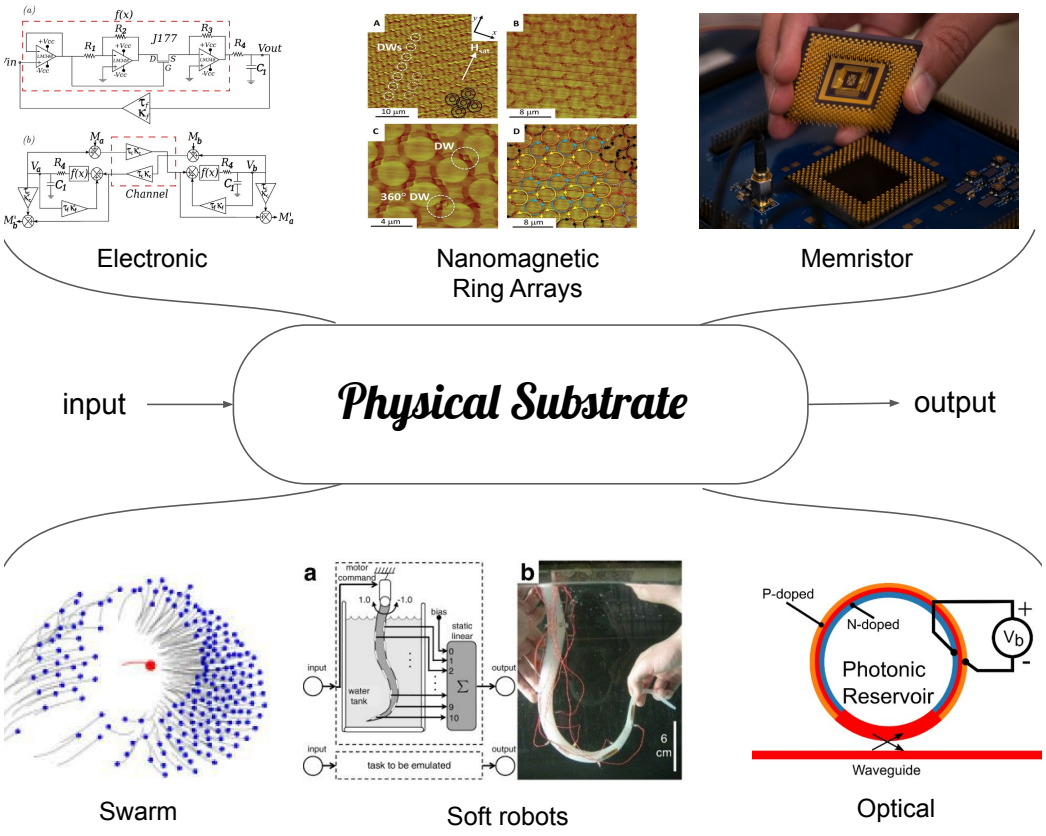


Figure 2.4: Reservoir Computing with physical substrates.

Optical and optoelectronic devices have emerged as potential substrates for physical RC, offering unique advantages such as high-speed parallel processing and low power consumption [132, 133]. These systems leverage the natural properties of light transmission in optical fibers, enabling efficient mapping of input data to high-dimensional spaces for classification and prediction tasks [134].

Traditional electronics [92,135], memristive devices [136–138], and Micro-Electromechanical Systems (MEMS) [139] have also become popular in this field. The unconventional substrates have been suggested, e.g., swarm systems [140] and soft robotics [141]. The material substrates should be able to perform nonlinear transformations and integrations concurrently.

2.5 Delay-line Reservoir Computing

The concept of delay lines was initially integrated into the RC algorithm to represent state updates. In 2011, a pioneering study by Appeltant *et al.* [92] proposed a delay-based reservoir with virtual nodes in a physical implementation of a computational reservoir, referred to as *Delay-feedback Reservoir Computing* (DFRC). This innovative work thoroughly investigated the role of dynamic neurons in physical RC, inspiring alternative approaches and positioning DFRC as a promising candidate in unconventional computing.

DFRC simplifies hardware implementation by significantly reducing the number of nonlinear neurons to a single one, employing physical delay lines and time-multiplexing operations. This is particularly advantageous when combined with analog and optical components for high-speed, low-power computing. Adjusting the ratio between the substrate's time constant and

the time-multiplexing intervals and create better virtual nodes [142]. These nodes, similar to classic reservoirs, generate complex reservoir dynamics by mapping input data to higher-dimensional spaces for classification.

The delayed feedback loop is critical for retaining previous information within the network, contributing to its Memory Capacity (MC). Consequently, DFRC is especially appealing when integrated with optical or opto-electronic devices, as the delay line can be directly implemented using a long optical fiber, subject to the transmission speed of light.

This approach reveals the computational capability inherent in basic delay dynamical systems while also simplifying the experimental development of artificial neural networks for computing applications. Injecting an input into the reservoir layer requires a pre-processing step called 'masking' before the input can pass through the nonlinear node at the beginning of the delay line. The *Masking procedure* is detailed in section 3.1.1.

Mask multiplexing implies that the input (through the mask) and output signals occur at a frequency N times higher than what would be required in a standard reservoir with N nodes and N output lines. This process specifically trades off the number of output lines for an increased output frequency.

Illustrated in figure. 2.5, within a delay period of time τ, the signal passes through the nonlinear node and N equally spaced *Virtual Nodes* along the delay line. The time interval between these nodes is defined as $\theta = \tau/N$. After traversing the delay line for the time interval τ, the signal is re-injected into the loop, forming a delay-feedback reservoir. The system's state matrix is observed and read-out at the end of each virtual node for each loop of delay, i.e., at every time period of θ. The state at time t relies on the output

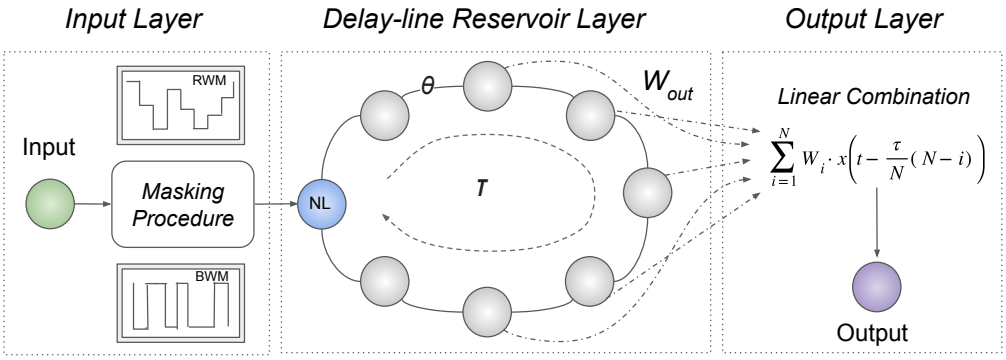


Figure 2.5: A three-layer topology is used in Delay-feedback Reservoir Computing, consisting of an Input layer, a Delay-line Reservoir layer, and an Output layer. Before being sent to the reservoir delay line through a single nonlinear node, the input signal undergoes a pre-processing step called "masking." The state matrix is monitored and read-out at the end of each virtual node, i.e., after a period of θ, to adjust the weights of connections between the reservoir and output layer for optimization purposes.

of the nonlinear node during the continuous time period $[t-\tau, t)$ [143].

Delay systems are particularly attractive for hardware implementation, as they require only a few components, such as a nonlinear node and a delay line [133,134,144–147]. More details on DFRC are given in section 3.1 in the next chapter.

2.6 Chaotic Dynamical Systems

Chaotic dynamical systems are characterised by their sensitivity to initial conditions, deterministic nature, and the absence of periodicity. Over the years, various mathematical models and concepts have been developed to better understand the behaviour of chaotic systems, such as the Mackey–Glass system [148], Lorenz attractor [2], and the Rössler system [3], which are all discussed below.

One of the key methodologies employed in the study of chaotic systems is nonlinear time series analysis, which aims to uncover the underlying structure and dynamics of the system from observed data. Techniques such as phase space reconstruction, Lyapunov exponents, and correlation dimension have been developed to quantify and analyse the chaotic properties of these systems, enabling a deeper understanding of their behaviour.

2.6.1 Mackey–Glass System

The Mackey–Glass dynamic equation has become a prominent subject of study in the fields of mathematics, mathematical biology, and nonlinear dynamics since its introduction to describe the blood cell population by Michael Mackey and Leon Glass in the late 1970s [148]. The original equation, which is a form of delay differential equation, has been widely recognised for its ability to capture both healthy and pathological behaviours in various biological contexts, depending on the values of its parameters. It is defined as follows:

$$\dot{P} = \frac{\beta \theta^n P_\tau}{\theta^n + P_\tau^n} - \gamma P_t, \quad P_\tau \equiv P(t-\tau) \tag{2.17}$$

where the state variable P_t is the homogeneous density of a population of mature blood cells at time t; τ is the time lag between initiating blood cell production and the mature blood cells being released; parameters β, θ, and n are related to the production rate; γ determines the decay rate of the cells. The parameters provided for the Mackey–Glass equation have been simplified to a version that is commonly used in many references. This set of parameters produces a chaotic system that is often of interest for studying

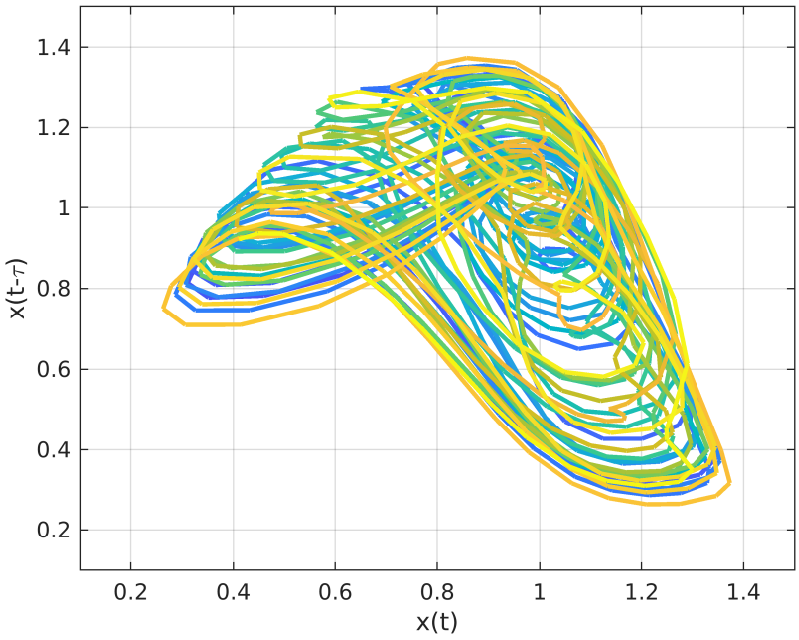


Figure 2.6: Mackey–Glass chaotic system with colour changing over time. $\beta = 0.2$, $\gamma = 0.1$, $n = 10$, and $\tau = 25$, are parameters commonly used in literature.

the system's dynamics. Figure. 2.6 shows the phase plot of Eq. 2.18:

$$\dot{x(t)} = \frac{\beta x_\tau}{1 + x_\tau^n} - \gamma x_t, \quad x_\tau \equiv x(t - \tau) \tag{2.18}$$

One of the primary applications of the Mackey–Glass equation has been in the modelling of hematopoiesis - the process of blood cell production and regulation in the body [148]. The equation has demonstrated its effectiveness in capturing the complex, nonlinear dynamics of hematopoietic systems, providing insights into the mechanisms underlying the regulation of blood cell populations and their responses to disturbances such as diseases or medical treatments.

The study of the Mackey–Glass equation has also contributed significantly to the broader field of nonlinear dynamics and chaos theory. The equation exhibits a rich variety of dynamic behaviours, including periodic, quasi-periodic, and chaotic solutions, depending on the choice of parameters [149]. These properties have made the Mackey–Glass equation not only to be considered as a substrate for reservoir computing but also as an important benchmark problem for the analysis and control of chaos in nonlinear systems.

2.6.2 Lorenz System

The Lorenz system, first introduced by Edward Lorenz in 1963, is a set of three ordinary differential equations that have played a significant role in the study of nonlinear dynamics, chaos theory, and the broader field of mathematical modeling [2]. The system was initially developed as a simplified model of atmospheric convection, and its discovery of chaotic behavior marked a turning point in the understanding of the unpredictability and complexity inherent in many natural systems.

The Lorenz system is defined by three coupled nonlinear differential equations:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x), \\ \frac{dy}{dt} &= x(\rho - z) - y, \\ \frac{dz}{dt} &= xy - \beta z. \end{aligned} \tag{2.19}$$

Here, x, y, and z represent the state variables of the system, while σ, ρ, and β are the system's parameters. These equations exhibit a range of dynamic behaviours, including stable and unstable fixed points, limit cycles, and,

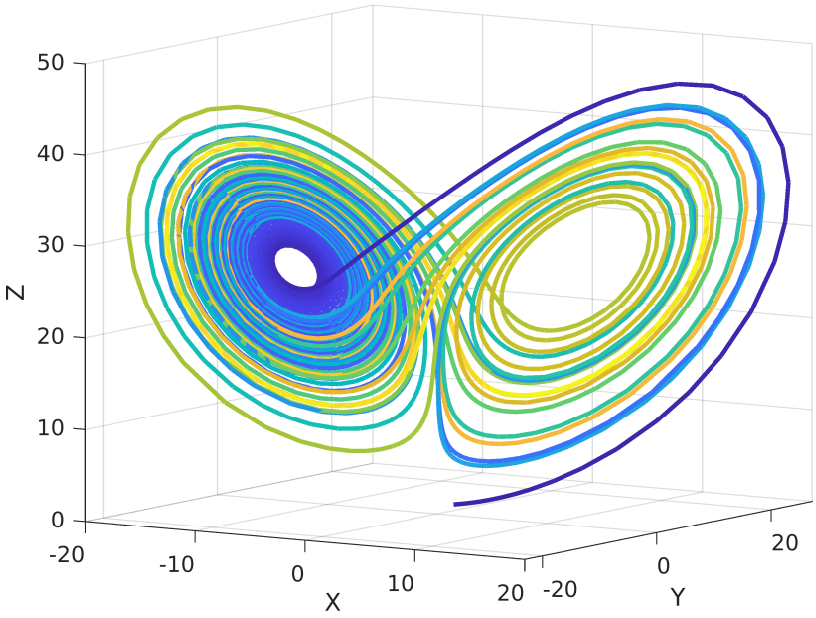


Figure 2.7: Lorenz Attractor with colour changing over time. $\sigma = 10$, $\rho = 8/3$, and $\beta = 28$, parameters taken from [2].

most notably, chaotic trajectories, depending on the values of the system's parameters. Figure. 2.7 shows the phase plot of Lorenz attractor with 3 dimensions.

One of the key features of the Lorenz system is the presence of the so-called Lorenz attractor which is characterised by its butterfly-like shape, that has become a symbol of chaotic behaviour in dynamical systems. The Lorenz attractor represents the long-term behaviour of the system, illustrating the sensitivity to initial conditions that is a hallmark of chaotic dynamics. This sensitivity is often referred to as the "butterfly effect", the idea that small perturbations in a system's initial state can lead to vastly different outcomes over time [2].

2.6.3 Rössler System

The Rössler system, introduced by Otto E. Rössler in 1976 [3], is a set of three coupled first-order nonlinear ordinary differential equations:

$$\begin{aligned} \frac{dx}{dt} &= -y - z, \\ \frac{dy}{dt} &= x + ay, \\ \frac{dz}{dt} &= b + z(x - c), \end{aligned} \tag{2.20}$$

which have played an influential role in the study of dynamical systems and chaos theory. The system was initially proposed as a simpler alternative to the Lorenz system, designed to produce chaotic behavior with a more straightforward mathematical structure.

This simplicity has made the Rössler system a widely used model in the investigation of the properties and mechanisms underlying chaotic dynamics

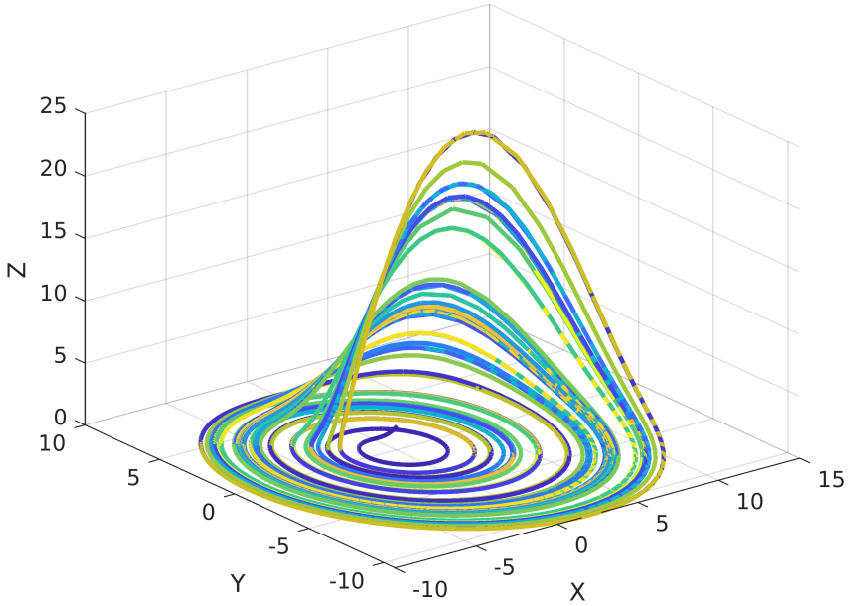


Figure 2.8: Rössler Attractor with color changing over time. $a = 0.2$, $b = 0.2$, and $c = 5.7$, parameters taken from [3].

in various fields, including physics, engineering, and biology.

The Rössler system has been employed as a model system in various applications, such as the study of synchronization in coupled chaotic oscillators, the control of chaos, secure communication using chaotic signals, and pattern formation in chemical reactions [150–152]. Moreover, it has been used as a benchmark system for the development of numerical methods, time series analysis techniques, and algorithms for the detection and characterization of chaos in experimental data [153, 154]. As research in nonlinear dynamics and chaos theory continues to evolve, the Rössler system remains a valuable tool for understanding the fundamental principles and applications of chaotic systems.

2.6.4 Summary of Chaotic Systems

Each of the chaotic systems discussed in this section has demonstrated rich dynamical behaviour, which make them suitable to be implemented either as a substrate of Reservoir Computing, or the benchmark tasks to evaluate the performance of RC.

To demonstrate our concepts, in this work, the well-studied Mackey-Glass oscillator is implemented as the substrate of Delay-Feedback Reservoir Computing Our selection is rooted in the ease with which this nonlinearity can be tuned using the exponential term. Additionally, it offers the flexibility to adjust the input scale and feedback strength, facilitating the exploration of ideal configurations for specific tasks.

3 Methodology

This chapter introduces the concept of Delay-feedback Reservoir Computing (DFRC) and the methodology followed for the experimental work in this thesis.

First, we go into detail on the principle of DFRC formed of three layers:

- Input layer: basic setups and pre-processing of input signal.
- Reservoir layer: dynamics of reservoir and its interconnection structure.
- Output layer: state matrix read-out and the training strategies.

Second, the nonlinear dynamical system used throughout the work, its modelling frameworks and the parameter configurations are presented.

Finally, a full description of benchmark tasks which give an indication on the computational performance of reservoir is outlined.

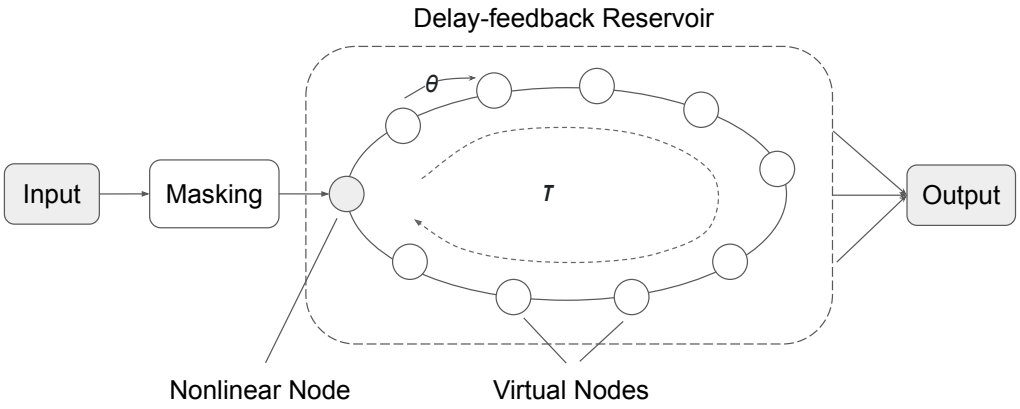


Figure 3.1: Topology of Delay-feedback Reservoir Computing with three layers: *Input*, *Delay-line Reservoir* and *Output*. The input signal undergoes a pre-procedure called 'masking' and transmitted to the reservoir delay line through a single nonlinear node. The state matrix is observed and read-out at the end of delay-line, i.e. after period of τ, in order to optimise the weights of connections between the reservoir and output layer.

3.1 Reservoir Computing based on delayed dynamical systems

In 2011, Appeltant *et al* [92] successfully demonstrates that a single nonlinear node with delayed feedback loop may replace an entire network of connected nonlinear nodes and effectively process information.

Figure 2.5 shows a schematic of delay-feedback reservoir as the equivalent to the classic RC implementation presented in Figure 2.3. This approach reveals the computational capability latent in basic delay dynamical systems while also simplifying the experimental development of artificial neural networks for computing applications.

When injecting an input into the reservoir layer, one must go through a pre-processing called 'masking' and then pass via the nonlinear node at the beginning of delay line. The detail of this *Masking procedure* is explained in

section 3.1.1. Within a delay period of time τ, the signal passes the nonlinear node and N equally spaced positions, the *Virtual Nodes* along the delay line. The time between the virtual nodes is defined by $\theta = \tau/N$. After the signal travels along the delay line for time interval τ, it is then re-injected into the loop, so forming a delay-feedback reservoir. For each loop of delay, the state matrix of the system is observed and read-out at the end of each virtual node, i.e. every time period of θ.

In this work, we demonstrate the idea of 'devirtualisation' by sub-sampling the state space; the detail is given in Chapter 4.

By introducing the characteristic of delay-feedback, dynamical systems which contain one-dimension non-linear node can be augmented with N virtual nodes to achieve an N-dimensional reservoir state space. This is due to the fact that its state at time t is dependent on the output of the non-linear node during the continuous time period $[t-\tau, t)$ [155]. The dynamics of the delay system remains finite-dimensional in practice, but shows high dimensionality and short-term memory, which are necessary for reservoir computing [143]. Delay systems are particularly desirable from a hardware implementation standpoint since they require only a few components, such as a nonlinear node and a delay line [133, 134, 144–147].

3.1.1 *Masking* : pre-processing of input signal

In delay-feedback reservoir computing, the masking procedure plays an essential role as it defines the virtual nodes along the delay line. The time delay between each of the virtual nodes is $\theta = \tau/N$, which is mainly defined by the procedure. Masking mixes the input signal with several sets of scaling

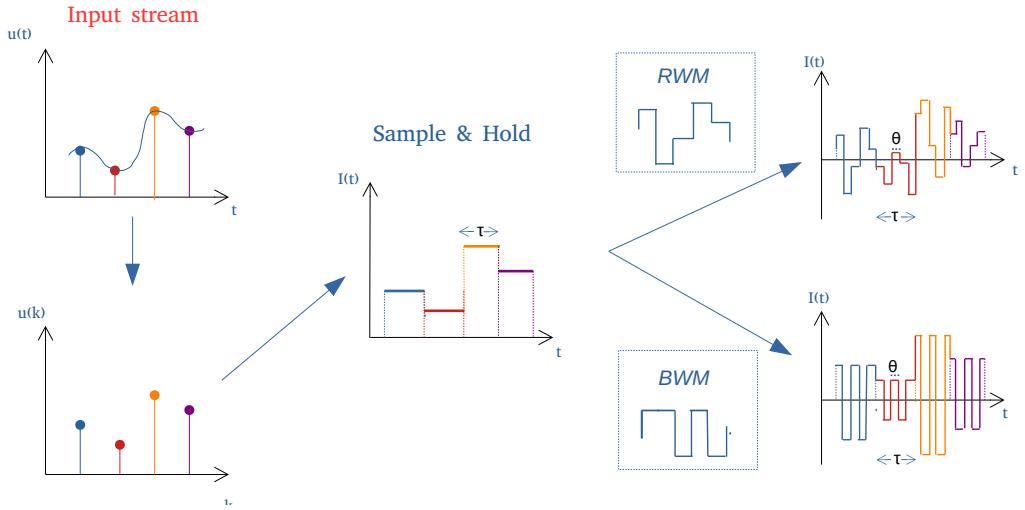


Figure 3.2: Masking procedure. Top left: the discrete inputs of the delayed feedback reservoir model. Bottom left: physical continuous inputs. Middle: 1Sample and Hold' discretising the physical inputs. Right: two masking procedures *Real Weight Mask* or a *Binary Weight Mask* applied to the discretised inputs.

factors to elicit a dynamically rich response from the reservoir. The masking procedure transforms the state multiplexing of N actual nodes in a reservoir to time-multiplexing, resulting in analog dynamics, with the mask being divided into N parts. The model's input stream comprises values indicative of discrete time readings.

A physical reservoir operates in continuous real time. Its input signals are injected into the reservoir layer after pre-processing by *Sample and Hold* (to discretise the input) and the mask-multiplexing procedure (to determine the virtual nodes). As shown in Fig.3.2, the discrete input stream $u(k)$ is interpreted as continuous time input $u(t)$, and converted into a piece-wise signal function $I_0(t)$ for $\tau k \leq t < \tau(k+1)$, which is constant during one delay period τ. A mask $M(t)$ is applied to $I(t)$ at each time interval, to provide the input to the reservoir. The masking $M(t)$ consists of a sequence

of N randomly generated values; the same mask is used for each interval τ.

Kuriki *et al* [132] investigates the influence of different input masking strategies on photonic reservoir computing using semiconductor lasers, using the Santa Fe time-series prediction problem as a benchmark. The effectiveness of masking may vary between different tasks. Typically, the masking method uses either a *Real Weight Mask* (RWM) or a *Binary Weight Mask* (BWM). the RWM is a sequence of random values drawn from the uniform distribution over $[-1, 1]$; The BWM is a random sequence of -1 and $+1$ values.

Due to the randomness of the masks, the results of many realisations may vary, even for the same type of mask. Investigation on the effectiveness of these two input mask types under different scenarios for benchmark tasks, as well as the randomness of masking, are discussed in Chapter 5.

3.1.2 Dynamics of the Reservoir layer

The reservoir consists of a nonlinear node with delayed feedback. The dynamics of the node can be represented by delay-differential equations (DDEs) as [156]:

$$\dot{x}_t = -x_t + F(x_\tau, I_{t,N}), \tau = t - \tau, x \in \mathbb{R} \tag{3.1}$$

where x_t is the value of dynamical variable at time t; x_τ is the delayed term at a certain point in the past (at time $t - \tau$); F is the transfer function of the node. $I_{t,N}$ is the input to be processed by the nonlinear node; N is the number of virtual nodes. Each element of $I_{t,N}$ creates a transient response in the nonlinear node of the system. Concatenating the responses of the nonlinear node to each input $I_{t,N}$ becomes the state matrix S of the system.

As the phenomenon of time delay occurs naturally in a variety of physical systems and the hardware implementation of delay-based reservoir computing requires only a single nonlinear node and a delayed feedback loop, this has resulted in numerous implementations in electronics, optoelectronics, and optics. Previous research indicates that delay differential equations are helpful for effective reservoir computation [92, 133, 134, 144, 157].

3.1.3 The output layer

One of the primary advantages of reservoir computing is its ability to confine the training process exclusively to the output layer. This unique characteristic results in a significant reduction in training overhead. During the training phase, the state matrix of the reservoir is consistently collected and read at intervals of τ [92]. Concurrently, the training algorithm designates an output weight to each virtual node present along the delay line. The objective is to compute a weighted sum of these states, aiming to align it as closely as possible with the expected output.

A more detailed expression of this assignment of output weights to the virtual nodes is captured by:

$$\hat{y} = \sum_{i=1}^{N} w_i \cdot x \left(\tau - \frac{\tau}{N}(N - i) \right) \tag{3.2}$$

In this equation, w_i denotes the weight given to the state of the i^{th} virtual node, x represents the output from the nonlinear node, and $\hat{y}$ is the computed approximation of the target output.

A linear training procedure is used to find the values of w_i. Following

the conventional approach for reservoir computing [85, 158], the readout is trained. The testing phase is then conducted with new input data of the same kind as those used for training.

Determination of Weights

The process of determining appropriate weight values, known as training, may be accomplished in one-shot (offline) learning or by progressively adjusting the weights (online learning). In our work, the former strategy has been used. For N nodes and L time steps, the result is an $[N \times L]$ dimensional state matrix, We refer this $[N \times L]$ dimensional state matrix as S. The target output y is used to produce an $[M \times L]$ dimensional target matrix, where M is the dimension of the target output y; here $M = 1$.

To calculate the optimal output weights W, the mean square error $\|WS - Y\|^2$ should be minimised. Ridge regression is applied to avoid problems with ill-conditioned matrices through the following formula:

$$W_{opt} = YS^T(SS^T - \lambda I)^{-1} \tag{3.3}$$

where T is transpose operation, λ is the regression parameter, I is the $N \times N$ identity matrix. This result can also be obtained by using the Moore–Penrose pseudo-inverse: $W_{opt} = YS^+$, where '+' indicates the pseudo-inverse function [158], but can have instability problems resulting in large weight values. The system trained output $\hat{y}(t)$ is given by:

$$\hat{y}(t) = W_{opt}x(t) \tag{3.4}$$

After training, the computational performance of the system is evaluated by injecting the test set of input signals to the reservoir, and the error calculation strategies (details in Section 3.3.1 and 3.3.2) is obtained using the trained output weights.

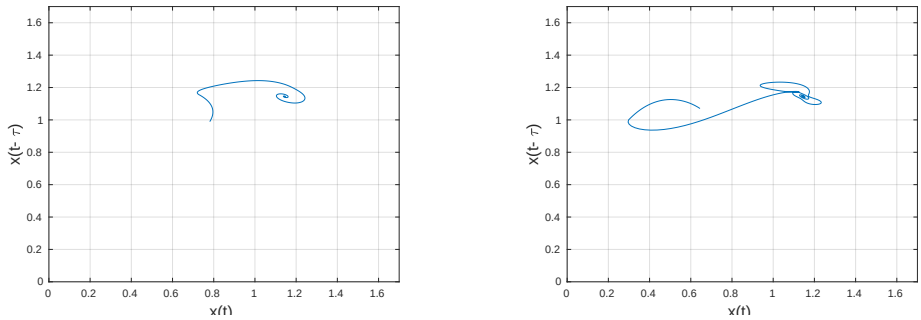
3.2 Reservoir Dynamical System: Mackey–Glass Model

The Mackey–Glass model [148] was introduced in the context of respiratory and hematopoietic diseases in which time delay plays a significant role [159–161]. The model is a first-order nonlinear delay differential equation

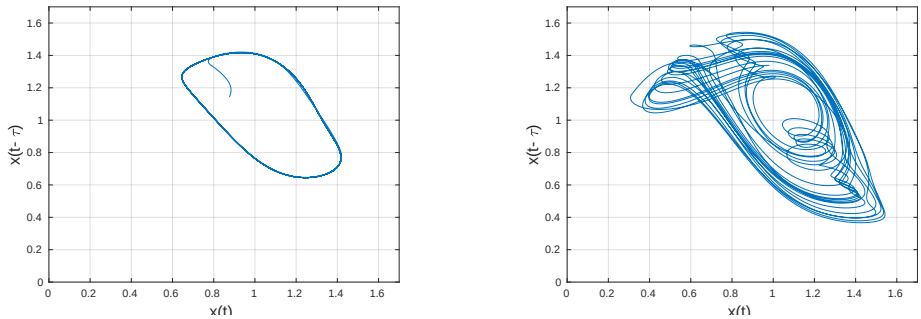
$$\dot{P} = \frac{\beta \theta^n P_\tau}{\theta^n + P_\tau^n} - \gamma P_t, \quad P_\tau \equiv P(t-\tau) \tag{3.5}$$

where the state variable P_t is the homogeneous density of a population of mature blood cells at time t; τ is the time lag between initiating blood cell production and the mature blood cells being released; parameters β, θ, and n are related to the production rate; γ determines the decay rate of the cells [148].

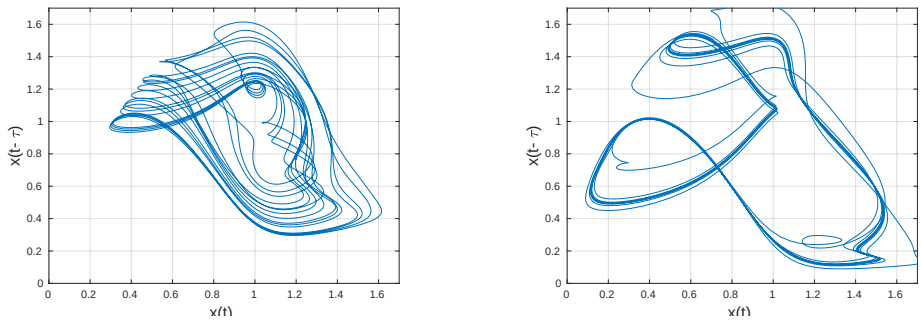
Depending on parameter values, the equation displays a range of aperiodic and chaotic dynamics (see Fig. 3.3). The model is suitable for use in delay-line reservoir computing, due to its rich dynamics and ability to be realised in hardware [125, 159, 162]. Appeltant et al. [92] investigate the time-normalised equation with state variable x_t in the context of delay line reservoir computing, by adding an external input I_t to the delayed feedback value, $x_\tau \to x_\tau + \delta I_t$, where δ is an input scaling parameter.



(a) $n = 3$, $\tau = 1.6$ (b) $n = 3$, $\tau = 3.2$



(c) $n = 6$, $\tau = 1.6$ (d) $n = 6$, $\tau = 3.2$



(e) $n = 12$, $\tau = 1.6$ (f) $n = 12$, $\tau = 3.2$

Figure 3.3: Mackey-Glass mathematics model phase plots with different nonlinear values n and delay period τ given by Eq.3.6, and $\gamma = 1$, $\beta = 2.5$.

Here, we follow [92] in introducing inputs, but we do not normalise the time, as we wish to use physical time units. We rename and scale the state variable ($x_t = P_t/\theta$) in Equation 2.17 to obtain an equation that is suited for physical system implementation based on voltage changes:

$$\dot{x}_t = \frac{\beta(x_\tau + \delta I_t)}{1 + (x_\tau + \delta I_t)^n} - \gamma x_t, \quad x_\tau \equiv x(t - \tau) \tag{3.6}$$

Here, x_t is the normalised voltage at physical time t; τ is the physical time delay in the feedback loop; the parameters β, n and γ denotes feedback strength, nonlinearity and decay rate, respectively.

3.2.1 Framework and Simulation

For numerically simulating the performance evaluation of delayed feedback reservoirs, we seperate them into two key steps.

As the MATLAB tool Simulink provides a versatile simulation environment including physical time, we start by implementing a circuit-like first-order differential equation as our dynamical model, i.e. the substrate of reservoir, with this tool [163].

The simulation algorithm pre-processes the input and simulates the reservoir states. When the reservoir states are generated, they are transferred to the second section, which is training. All of the training approaches and processes we use are independent of the design of the reservoir and rely only on the discrete time reservoir states in MATLAB [164].

As a first step towards the physical implementation of Mackey–Glass delay-dynamical system (Eq. 3.6), here we define the reservoir using Simulink (10.1/2020b Models). This circuit-like model, shown in Fig.3.4, is an inter-

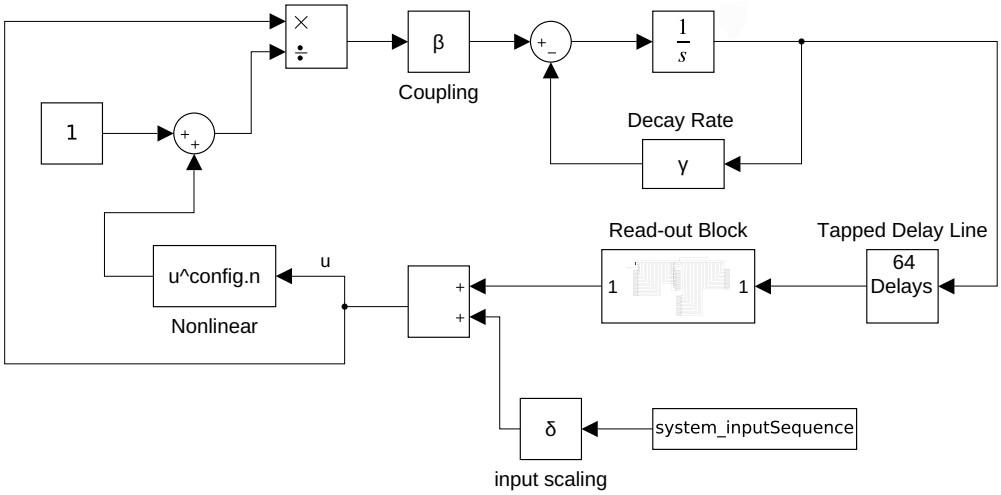


Figure 3.4: Simulink circuit schematic of ideal Mackey–Glass delay-feedback system.

mediate step from the mathematical DDE model to a physical electronic circuit.

Model blocks are selected for implementation according to the variables in Eq.3.6, and are connected to implement the various terms. The block labelled $1/s$ is the integrator. The delay line of virtual nodes and time delay τ is modelled with the 'tapped delay' block; this allows direct access to the information from virtual nodes for the devirtualisation experiments. The input signal is generated in Matlab, and injected into the system. via a 'From Workspace' model named 'system inputSequence' in the figure. The state matrix of virtual nodes is collected from a sub-system 'Read-out block' (not part of the mathematical model, but needed in the Simulink implementation), where a 'To Workspace' model is used inside the block, sending information to Matlab for training and evaluation processes.

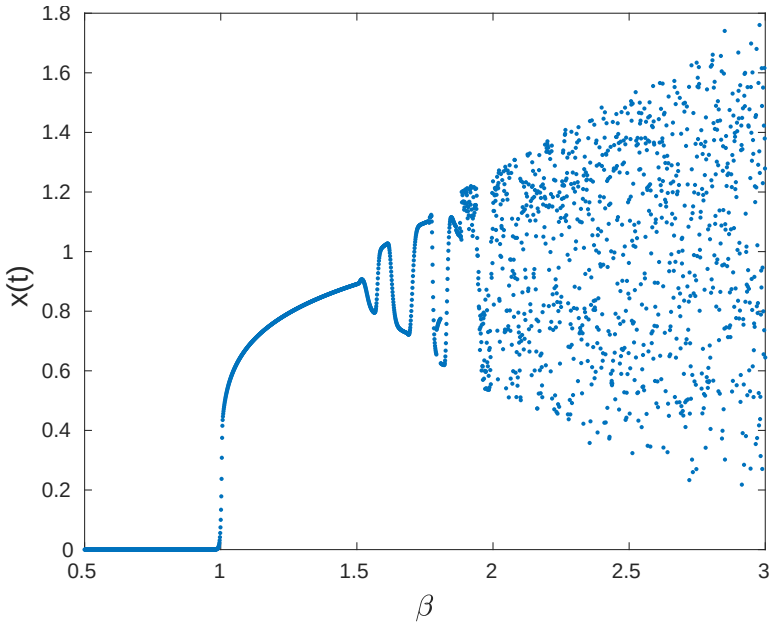


Figure 3.5: Orbit diagram of Mackey-Glass mathematical model.

3.2.2 Parameter Configuration

Feedback Strength : β

The chaotic regime of nonlinear function generates a rich and diverse set of temporal patterns that can be used to train the reservoir. These patterns can then be used to make predictions or perform other machine learning tasks.

Figure 3.5 shows the bifurcation diagram of Equation 3.6 where we can see a fix-point dynamics for $0 < \beta < 1.58$, this fix point is zero for $\beta < 1$. Beyond the fix-point dynamics, limit cycles develop a deterministic chaotic dynamics.

In a chaotic regime, the Mackey-Glass equation 3.6 produces highly nonlinear and dynamic temporal patterns that are difficult to predict. By using a chaotic regime of the Mackey-Glass equation as the substrate for the reservoir, it is then able to create a high-dimensional, nonlinear mapping of the

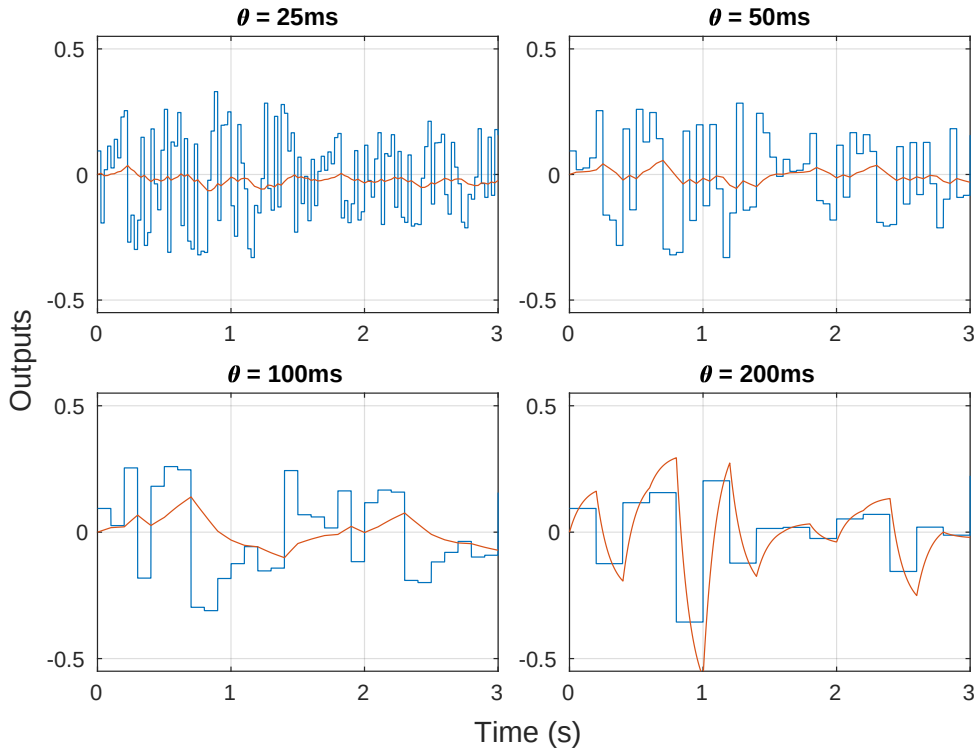


Figure 3.6: Input traces for different θ and their corresponding signals.

input signal, which can be used to perform various benchmark tasks.

On the other hand, using a stable regime of the Mackey-Glass equation as the substrate for the reservoir may not be as effective, as it produces less complex and less dynamic temporal patterns.

Distance between Virtual Nodes : θ

There are three crucial time scales in Delay-Feedback Reservoir Computing: the distance between virtual nodes θ, the delay time τ and the timescale of nonlinear node T. Previous work by McDonnell *et al.* [165] found that, the relation between timescale and θ has a significant effect on computational

performance of DFRC – good performance obtained when the time scales are related by $\theta \leq T$.

In our work, the timescale of the nonlinear node is set to 100ms (real time) in Simulink model 3.4, where the results of different θ and their corresponding signals are shown in Figure. 3.6. When $\theta = 200$ms and greater than timescale T, the system rapidly reaches a steady-state that is independent of previous inputs. Empirically we found for 64 nodes, the $\theta = 50$ms is the best choice, where we use $\theta = 25$ms as comparison (determined using NARMA and Santa Fe task in Chapter 4).

Nonlinear Value n

The transfer function of Mackey-Glass equation is

$$X_{out} = \frac{\beta \cdot X_{in}}{1 + (\delta \cdot X_{in})^n} \tag{3.7}$$

where the input and feedback term from Equation 3.6 is removed. This allows to change the operational point as the reservoir changes from strongly nonlinear to a weak nonlinear. The shape of the nonlinearity is illustrated in Figure 3.7. The nonlinear values n chosen between 1 and 12 are presented. In [4], the nonlinear parameter n is chosen to be 9.65 to provide chaotic behaviour. In that model, state values are always positive (they represent blood cell concentrations). Here state values are voltages, and are modulated by inputs; with some masking procedures the state value plus input term $x_\tau + \delta I_t$ can become negative. In order to avoid problems with raising negative values to fractional powers, we restrict n to integer values. These values straddle the chaotic Mackey-Glass value, and provide both an odd power

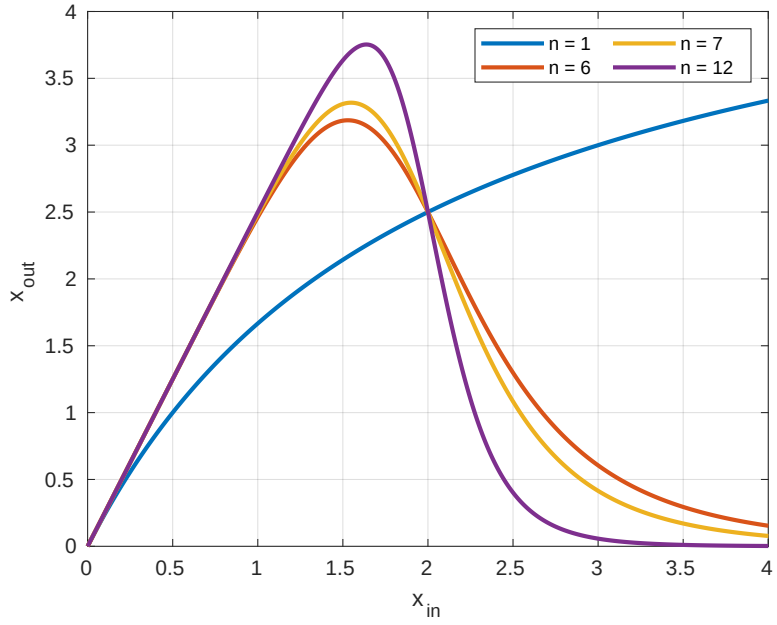


Figure 3.7: Mackey-Glass mathematics model transfer function with different nonlinear values n, given by Eq.3.7, and $\beta = 2$, $\delta = 0.5$.

(so negative values stay negative) and an even power (so negative values are transformed to positive values). In this thesis, our primary focus is not on achieving cutting-edge outcomes but rather on showcasing the techniques involved in designing delay-line reservoirs. Therefore, we predominantly use a mild n value (6 or 7) throughout our research.

3.3 Benchmark Task Definition

In this section we illustrate the benchmark tasks used in this work (Chapter 4 and Chapter 5). We have chosen two different tasks to evaluating the performance of our delay-feedback reservoir:

- An imitation task, in which the system mapps the input sequence to

an output sequence based on a specific relationship between the inputs and outputs.

- A time series prediction task, where the reservoir uses its own memory to prediction one-step data in the future.

3.3.1 Dynamical System : NARMA

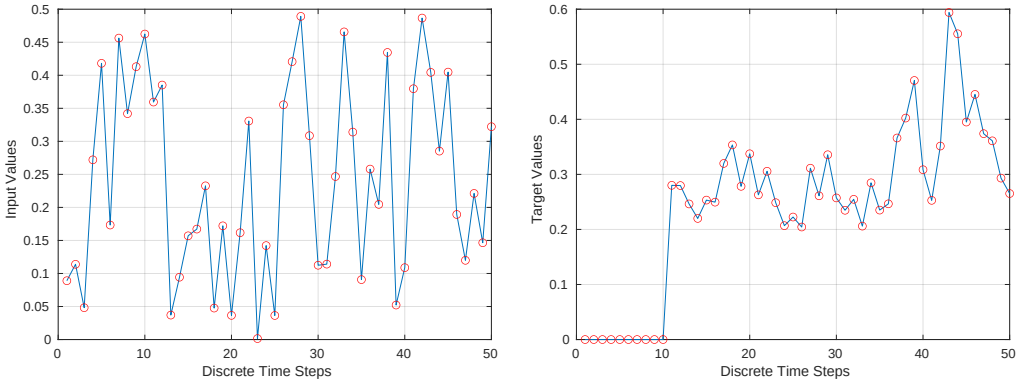
We use the discrete-time n-th order NARMA benchmarks to analyse the computational abilities of the delay-based reservoir in various scenarios [166]. 'NARMA' is an acronym for Non-Linear Auto-Regressive Moving Average and it is one of the most widely used benchmark in reservoir computing. In the literature, two instances of the NARMA model occur frequently: NARMA-10 and NARMA-30, where the number at the end represents the order of the model. The NARMA-10 model is given by:

$$y_{k+1} = 0.3y_k + 0.05y_k \left(\sum_{i=0}^{9} y_{k-i} \right) + 1.5u_k u_{k-9} + 0.1 \tag{3.8}$$

The NARMA-30 model is defined as:

$$y_{k+1} = 0.2y_k + 0.004y_k \left(\sum_{i=0}^{29} y_{k-i} \right) + 1.5u_k u_{k-29} + 0.001 \tag{3.9}$$

where k is the time-step, and u_k is an input stream randomly generated from a uniform distribution over the interval $[0, 0.5]$. The parameters of NARMA-10 and NARMA-30 comes from [166]. An example of input sequence and target output for NARMA-10 is given in Fig.3.8, where the first 10 time steps of the target sequence are empty. The calculation of state $k+1$



(a) Input sequence (b) Target sequence

Figure 3.8: NARMA-10 input and target sequences. 3.8a) Discrete points drawn from a uniform distribution within the interval $[0, 0.5]$. 3.8b) Target points calculated from input points using NARMA-10 equation (3.8), with the first 10 steps of the target equal to 0.

involves y_{k-i}, u_{k-i}, and nonlinear terms, indicating n-th NARMA benchmark requires both memory and non-linear processing.

The task is to train the reservoir to reproduce the NARMA-10 / NARMA-30 dynamics as closely as possible when driven with the same input stream. The process is presented in Fig. 3.9. The reservoir input signal I_t is obtained from u_k according to the procedure defined in (Masking section).

We use normalised root mean square error (NRMSE) to quantify and compare the performance of approaches between different experiments:

$$\text{NRMSE} = \sqrt{\frac{1}{m}\frac{\sum_{k=1}^{m}(\hat{y_k} - y_k)^2}{\sigma^2(y_k)}} \tag{3.10}$$

where y is the NARMA target function, $\hat{y}$ is the reservoir output, m is the number of data samples in the run, and σ is the standard deviation. An NRMSE of 0 indicates perfect agreement between system output and target

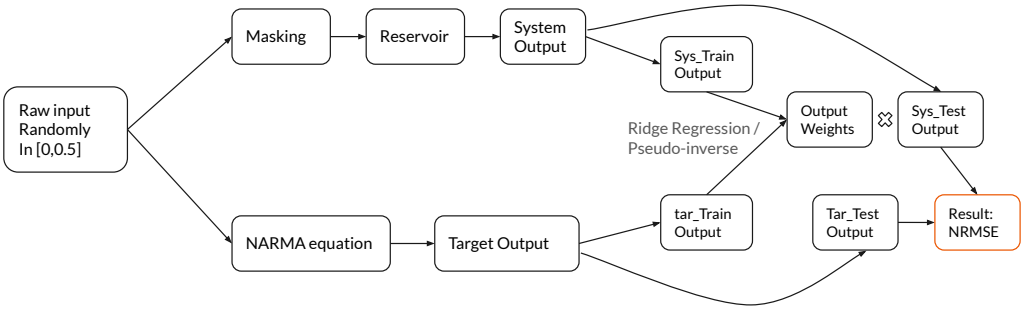


Figure 3.9: NARMA benchmark task process flow diagram.

output; an NRMSE of 1 can be achieved by setting the system output equal to the mean target output.

3.3.2 Time-Series Prediction : Santa Fe Laser Task

The Santa Fe laser benchmark task is a commonly used benchmark for evaluating the performance of reservoir computing algorithms. This task involves the prediction of the chaotic behavior of a laser system that using a LeCroy oscilloscope, measurements were taken on an 81.5-micron $14NH_3cw(FIR)$ laser. The setup is described in [167].

In this task, the laser system is modeled as a set of coupled nonlinear differential equations, and the goal is to predict its future behavior based on a time series of its past states (Figure 3.10). The error between the target output y and reservoir output $\hat{y}$ is represented as a Normalised Mean Square Error, which is defined as follows:

$$\text{NMSE} = \frac{1}{m} \frac{\sum_{k=1}^{m} (\hat{y}_k - y_k)^2}{\sigma^2(y_k)} \tag{3.11}$$

Since the Santa Fe Laser Task is a time-series prediction problem, and in

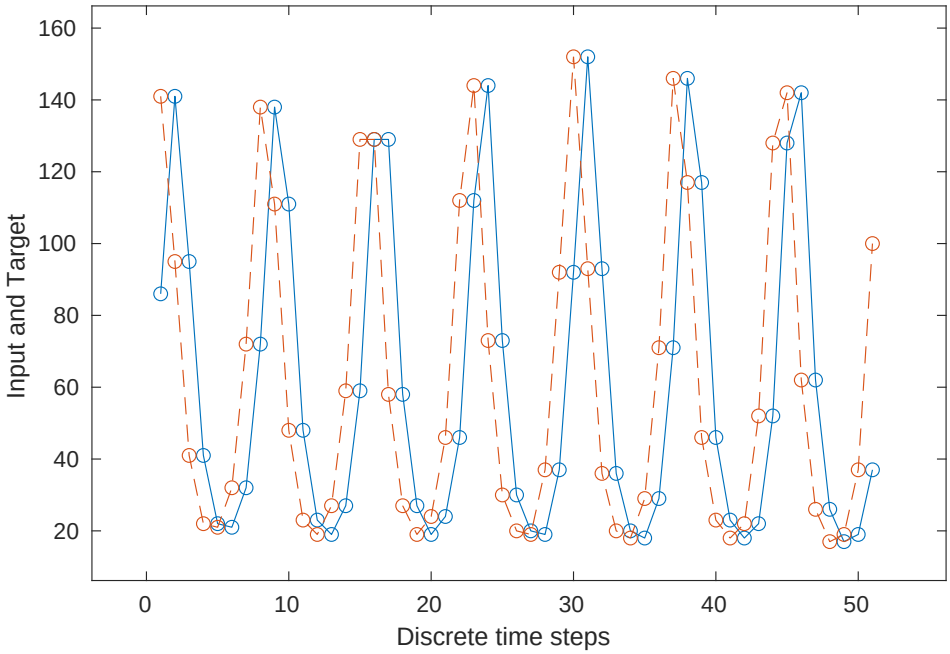


Figure 3.10: Santa Fe input vs target. Input: blue solid line. Target: red dotted line.

such problems, larger errors can have a greater effect on the accuracy of predictions. NMSE emphasises greater mistakes more so than other measures since it squares the discrepancies between projected and actual values. This makes it a suitable metric for measuring the performance of models on this task, where it is essential to minimise larger errors. The choice of NMSE for the Santa Fe Laser Task is also influenced by consistency with existing literature and research.

3.4 Summary

This chapter presents a detailed explaination of Delay-feedback Reservoir Computing (DFRC), outlines the experimental methodology employed and the benchmark tasks used in this thesis.

Section 3.1 delves into the fundamental principles of DFRC, focusing on three crucial layers and the training procedures. The first layer discussed is the input driving layer, which encompasses the basic setups and the pre-processing procedure "Masking" applied to the input signal. The second layer is about the dynamics of the reservoir and its interconnection structure. Understanding the behavior and architecture of the reservoir is highlighted as crucial in the overall computation process. The third layer is the output layer, which encompasses the state matrix read-out and the employed training strategies. This section highlights the methods used to extract valuable information from the reservoir and effectively train the system.

Following the detailed exploration of the three layers, Section 3.2 presents the nonlinear dynamical system used in the experimental work. Various modeling frameworks and parameter configurations are outlined, providing

a comprehensive understanding of the investigated system.

Finally, this chapter presents a comprehensive description of the benchmark tasks used to evaluate the computational performance of the reservoir. These carefully selected tasks offer meaningful insights into the reservoir's capabilities and performance in different scenarios.

4 Devirtualisation

In this chapter, we introduce an approach termed **Devirtualisation**. Devirtualisation describes the procedure of directly tapping into the delay line at the position of a 'virtual' node, rather than at the delay line's end. This approach allows for a trade-off between the read-out frequency and the number of output lines. It's noteworthy to mention that physical delay lines are subject to attenuation, which diminishes the signal strength. Consequently, we explore how devirtualisation, by harnessing a less-attenuated output, may mitigate the impacts of attenuation in the physical delay line.

Section 4.1 introduces the concept and rationale behind 'Devirtualisation'. This chapter uses both ideal and attenuation delay-line environments, with four distinct configurations, by varying the length of the entire delay-line and time lag between each virtual node. The experimental setups for these configurations are detailed in Section 4.2. The first experiment, presented in Section 4.3, uses an ideal delay-line to test the proper functioning of each block and demonstrate the feasibility of 'devirtualisation' without the influence of other factors. In Section 4.4, the second experiment compares the ideal delay-line and damping delay-line with fully virtualised systems, meaning that only one output line is used. Lastly, in Section 4.5, we investigate the devirtualisation approach in a damping environment, focusing on

two distinct scenarios: *a*) Constant damping rate over the delay-line and *b*) Constant damping rate per 'length'.

The benchmark tasks including NARMA-10, NARMA-30 and Santa Fe Laser Task are used to demonstrate the effectiveness of this approach under different configurations in all experiments.

4.1 'Devirtualised' Systems

In recent years, there has been a significant growth in the use of photonic/optical fibre for physical delay-line reservoir computing [168–172]. This paradigm shift is driven by the numerous advantages offered by these optical computing systems, including their ultrafast processing speeds, high bandwidth capacities, and low energy consumption.

However, in the real world, several practical challenges can significantly impact the performance of physical systems employed for delay-line reservoir computing, including **attenuation (damping)** and **dispersion**. The attenuation or damping in a photonic system could occur when some portion of the signal is absorbed or scattered, so the signal's amplitude is gradually decreased as propagates through the system. This can result in weaker signals and a reduced signal-to-noise ratio (SNR), making it harder for the delay-line reservoir to process and interpret the input signals accurately.

Our concept of 'devirtualisation' offers a way to trade-off the number of output lines and the real-time readout frequencies in physical delay-feedback reservoir systems. This approach is aimed at addressing practical challenges that affect the system along the delay-line from data-processing aspect (see Section. 2.5).

Figure 4.1 shows the illustration of 'devirtualisation' strategy. If we set the sampling time equal to τ, which is the delay time along the physical tapped delay, we need to use N output lines to collect data simultaneously from each of the N 'devirtualised nodes'. We have increased the number of outputs to N, and reduced the output frequency by N (from $1/\theta$ to $1/\tau$). If we sub-sample the system with time $\tau/2$, only half of the virtual nodes are

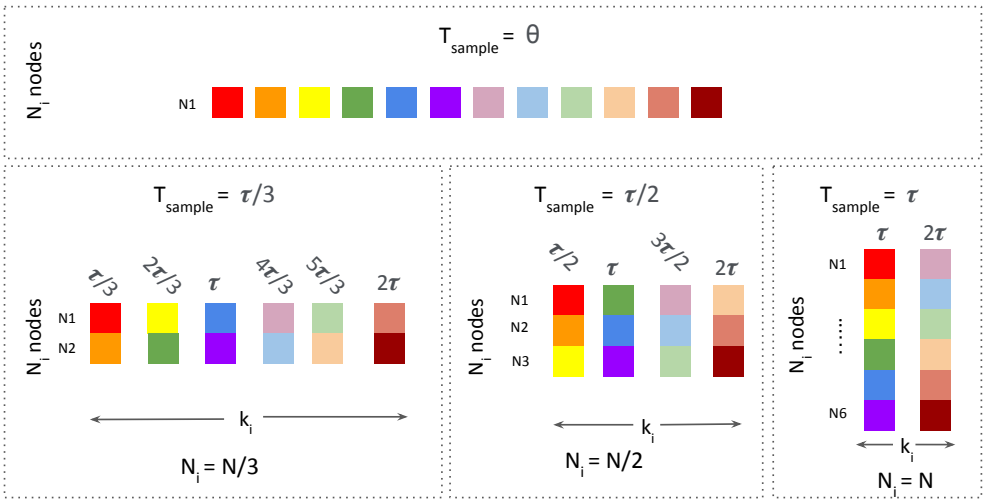


Figure 4.1: 'Devirtualisation' example with $N = 6$ nodes. Each coloured square block represents one output signal from one virtual node. N_i indicates the number of connected nodes (output lines). T_{sample} is the sampling time, i.e, every connected output line collects data simultaneously at each T_{sample} time. When $N_i = N$, every virtual node $N1..N6$ has an output line, and the output is sampled every τ. Here, $N_i = 1$ where $T_{sample} = \tau$ is the base case. When $N_i = N/2$, nodes $N1..N3$ have output lines: at $\tau/2$ they output the first three blocks, at time τ they output the last three blocks. Similarly, when $N_i = N/3$, nodes $N1..N2$ have output lines, outputting each $\tau/3$.

required to output data simultaneously.

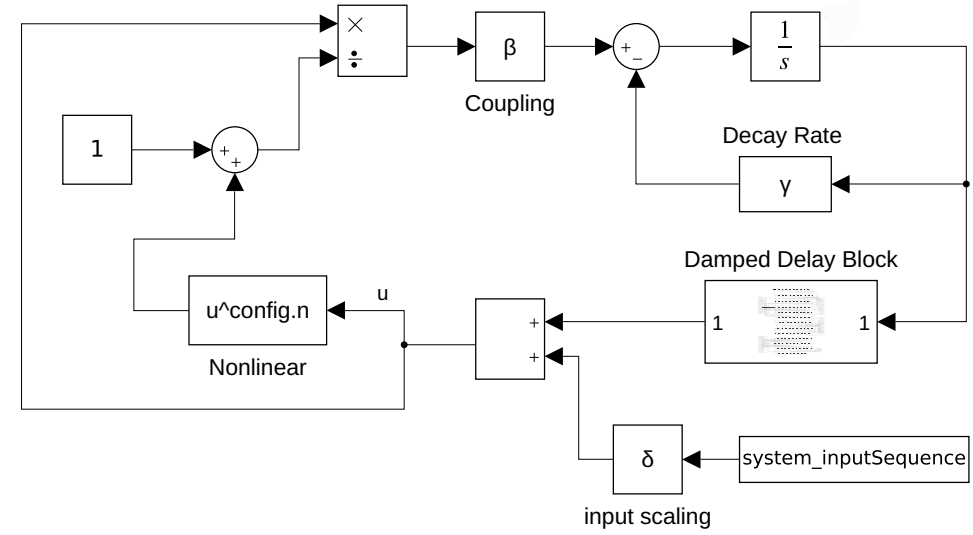
No data is lost in this devirtualisation process: the same data is sampled in each case. Before the standard procedure of training or testing is applied, the read-out matrix from the devirtualised nodes is reshaped appropriately to the dimension of $[N \times k]$.

4.2 Experimental setups

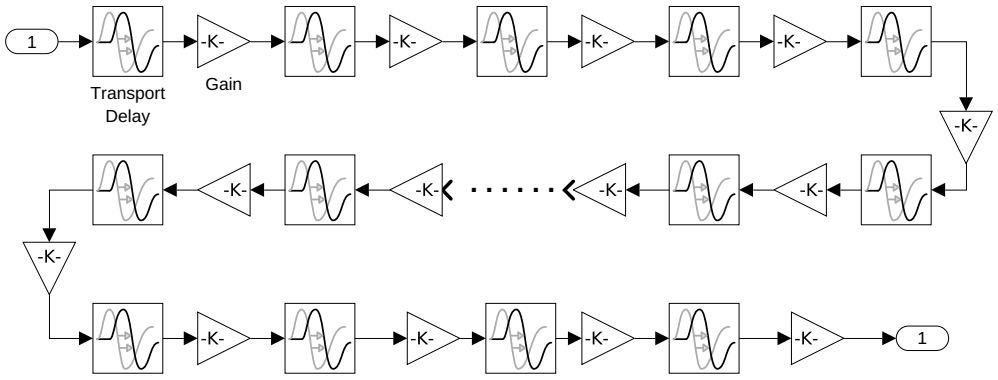
4.2.1 Physical implementation

The experimental simulation framework, presented in Section 3.2.1, is carried out using MATLAB and SIMULINK as the primary tools. In this section, we simulate information loss in an optical fiber, which is widely used as the delay line in delay-feedback reservoir computing.

Typically, the information loss in an optical fiber is characterised by its attenuation, which is the reduction in the optical power of the signal as it travels through the fiber. Attenuation is usually expressed in decibels per kilometer (dB/km) and is dependent on the wavelength of light used. For a 10-kilometer-long optical fiber, the information loss can be calculated by multiplying the attenuation by the cable length. For example, if the attenuation is 0.3 dB/km, the power loss in a 10-kilometer-long optical fiber would be approximately 3 dB, or 50%. The ideal delay line is modified with the 'Gain' blocks, used to model the damping in the reservoir's delay line by setting the gain value of the blocks to the damping coefficient of the physical system. The output lines are simulated with the implementation of Mackey–Glass system in Simulink (Figure 3.4 for ideal delay-line and Figure 4.2 for



(a) Overall schematic



(b) Damped delay block

Figure 4.2: Simulink circuit schematic of Mackey–Glass delay-feedback system with damping property. 4.2b) 'Gain' blocks are used to model the damping property by modifying the gain value of the block in order to set the damping coefficient of the system.

damping delay-line).

4.2.2 Parameter settings

In this chapter's experiments, we employ the *Real Weight Mask* as the time-multiplexing function (refer to Section 3.1.1). For each of the 30 runs in a single sampling scheme, we use different random seeds to regulate both the mask and input sequence. Moreover, we maintain the same random seeds across all sampling schemes for the given configurations.

NARMA-10 and NARMA-30. The NARMA-10 and NARMA-30 equations are presented in Section 3.3.1, with Eq.3.8 corresponding to NARMA-10 and Eq.3.9 representing NARMA-30. The input sequence length generated by NARMA is $L = 3000$. This sequence is divided into *training* and *testing* datasets, with $L_{train} = 1700$ and $L_{test} = 1200$ respectively. The initial $L_{washout} = 100$ results from each set are disregarded. Each experiment consists of 30 runs.

The performance of the reservoir is evaluated using the normalised root mean square error (NRMSE), which compares the predicted values against those obtained from the NARMA model (details of NRMSE are given in 3.3.1).

Santa Fe laser task. The Santa Fe laser data prediction task serves as an instance of one-step time series forecasting. The dataset used in this case comprises 4000 data points, which are split into four distinct samples, each containing 1000 points. As the normalised mean square error (NMSE) is the standard metric used in the literature to evaluate performance on this task, we employ NMSE to compare predicted values with their actual value in our

parameter	value	description
β	0.8	coupling factor
γ	1	decay rate
τ	See Figure 4.4	delay in feedback loop
n (NARMA)	6	nonlinearity
n (Santa Fe laser)	2	nonlinearity
δ	0.5	input weighting
N	32 and 64	number of virtual nodes

Figure 4.3: Parameter values for the Mackey–Glass delay-line reservoir devirtualisation experiments. The Mackey–Glass system parameter values are from [4].

experiments. The definition of NMSE is given in Section 3.11.

The experimental parameter settings can be found in Figure 4.3. The majority of these parameters are cited from [4], with the selection of nonlinear value, delay time, and the number of virtual nodes configurations being determined based on the objectives of this chapter.

Figure 4.4 illustrates four delay-lines under investigation in this section, each with varying lengths or a different quantity of virtual nodes in the system.

- Delay-line **A** has 32 virtual nodes defined along it, with time lag of 25ms between each consecutive virtual node, total delay time τ in feedback loop is 0.8s.
- Delay-line **B** has 32 virtual nodes defined along it, with time lag of 50ms between each consecutive virtual node, total delay time τ in feedback loop is 1.6s.
- Delay-line **C** has 64 virtual nodes defined along it, with time lag of 25ms

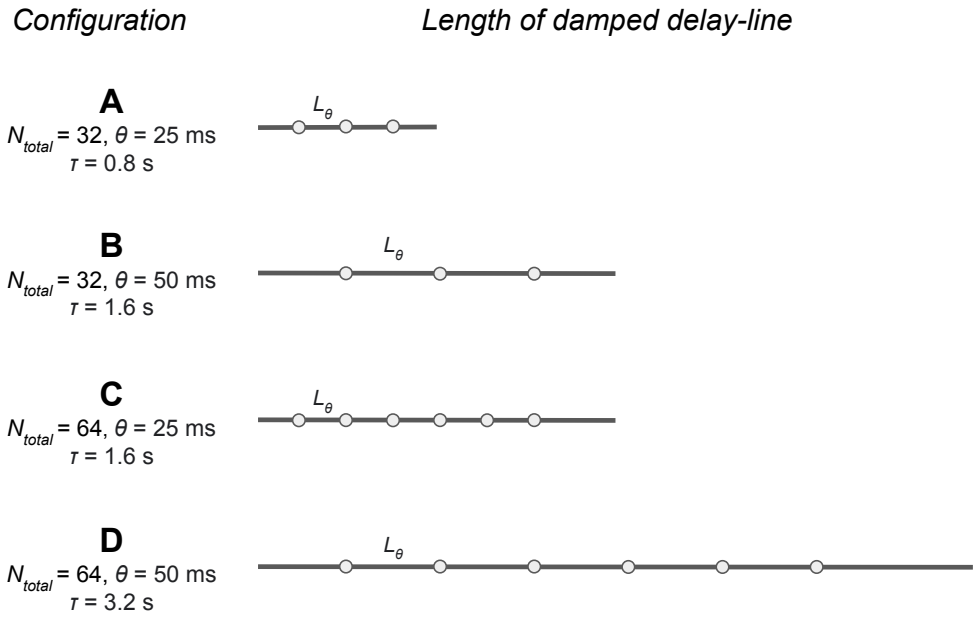


Figure 4.4: The length of damped delay-line of 4 different configurations, A: N_{total}=32, θ=25ms, τ=0.8s; B: N_{total}=32, θ=50ms, τ=1.6s; C: N_{total}=64, θ=25ms, τ=1.6s; D: N_{total} =64, θ=50ms, τ=3.2s.

between each consecutive virtual node, total delay time τ in feedback loop is 1.6s.

- Delay-line **D** has 64 virtual nodes defined along it, with time lag of 50ms between each consecutive virtual node, total delay time τ in feedback loop is 3.2s.

The setups illustrated in Figure 4.4 play a crucial role in examining the impact of various configurations that use damping in delay-line reservoir computing.

With four distinct configurations under ideal and attenuation environments, we explore the effectiveness of θ (time between each virtual node) and the length of delay-line. Moreover, we investigate this sub-sampling de-

virtualisation approach, that allows this tradeoff to be altered: reducing the readout frequency by using multiple output lines while the number of virtual nodes in the reservoir layer is constant.

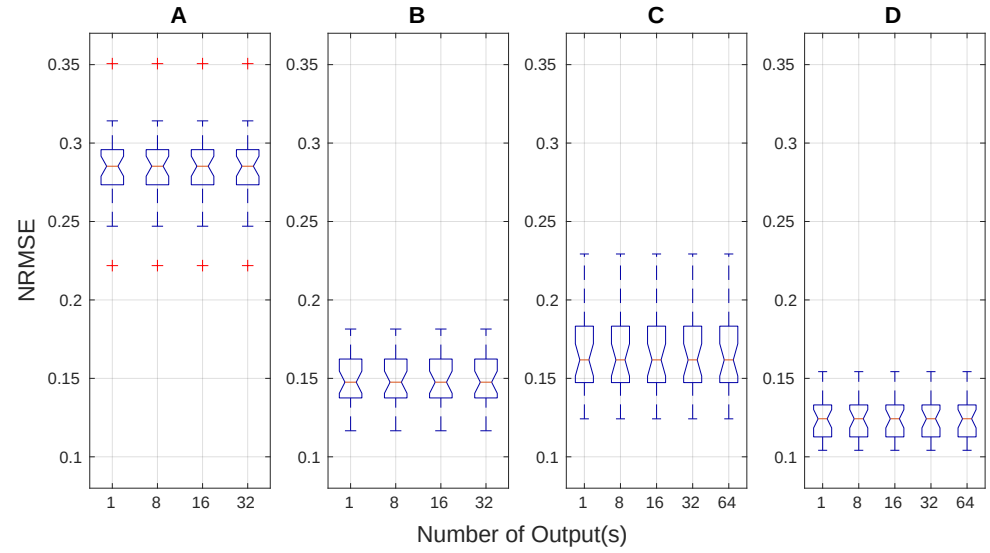
4.3 Devirtualisation with ideal delay-line

In this section, the results from ideal environment simulations are presented, demonstrating the proper functioning of each block shown in Figure 3.4. This experiment investigates the tradeoffs between node number, and distance between nodes, in terms of Mackey-Glass timescales. The NARMA-10, NARMA-30, and Santa FE laser tasks are used as benchmark tasks for experimental purposes (refer Section 3.3 for details of benchmark tasks).

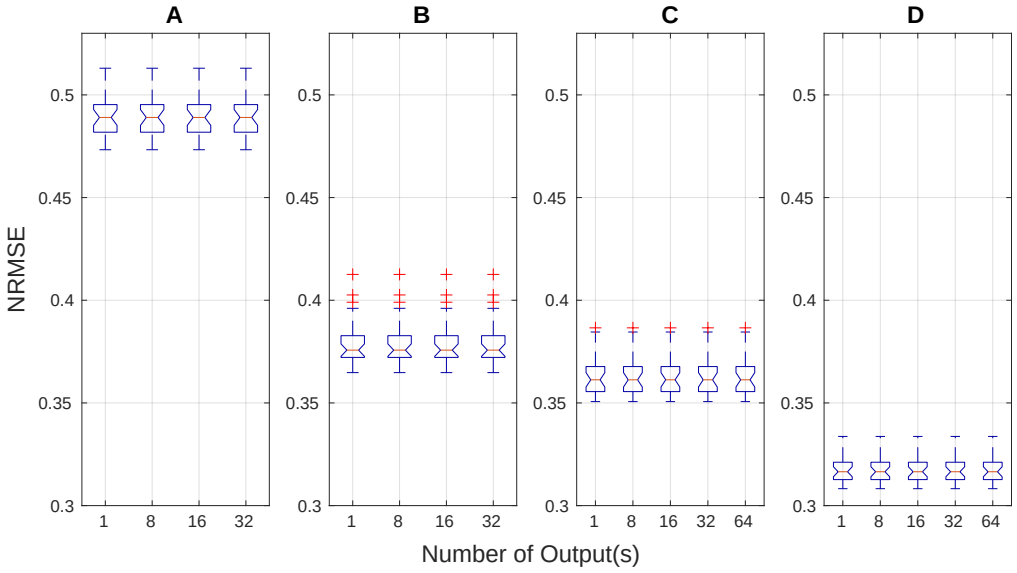
4.3.1 NARMA

Figure 4.5 shows the results of NARMA-10 (Eq. 3.8) and NARMA-30 (Eq. 3.9) benchmark tasks with four configurations mentioned in the last section. In the NARMA-10 and NARMA-30 tasks, delay-line A, with 32 virtual nodes and a 25ms time delay between each node, exhibits the poorest performance; NRMSE = 0.285 for NARMA-10, NRMSE = 0.4923 for NARMA-30. In contrast, delay-line D, consisting of 64 virtual nodes and a 50ms time delay, demonstrates the most favorable results; NRMSE = 0.1242 for NARMA-10, NRMSE = 0.321 for NARMA-30.

Devirtualisation in ideal environment. In the NARMA-10 and NARMA-30 tasks, when comparing varying numbers of output from the delay-line within the same configuration, the results are the same for the lossless SIMULINK



(a) NARMA-10



(b) NARMA-30

Figure 4.5: (a) NARMA-10 and (b) NARMA-30 experimental results for devirtualised Mackey-Glass system with ideal environment. The x–axis 'Number of Outputs(s)' indicates the actual output lines connected to the delay-line. A: N_{total}=32, θ=25ms; B: N_{total}=32, θ=50ms; C: N_{total}=64, θ=25ms; D: N_{total} =64, θ=50ms.

implementation employed in this case. While this may appear somewhat simplistic, it aligns with our expectations and would not typically occur in a real-world system with attenuation.

Varying time delay θ. When comparing delay-line A with delay-line B, or delay-line C with delay-line D as shown in Figure 4.5a and Figure 4.5b, it becomes clear that the latter in each pair yields better results. Although they all share the same number of virtual nodes (A and B possess 32 virtual nodes, while C and D have 64), the enhancement in performance can be linked to the distinct spacing between virtual nodes (for further details, see Section 3.2.2).

Varying number of virtual nodes. Conversely, when comparing pairs with the same time delay θ but varying numbers of virtual nodes (A and C or B and D), it becomes apparent that configurations with more virtual nodes possess greater computational capacity and more trainable read-outs, leading to improved performance.

Same length of delay-line. In the NARMA-10 and NARMA-30 experiments, delay-line B and delay-line C have the same length, with signals taking 1.6 seconds to travel through the delay-line. Despite this, delay-line B has twice the spacing between nodes and half the number of virtual nodes compared to delay-line C, resulting in differing outcomes for NARMA-10 and NARMA-30. For NARMA-10, delay-line B performs better, while for NARMA-30, delay-line C shows superior performance. This can be explained by the fact that delay-line B, with only 32 nodes, may lack the memory ca-

pacity needed to handle NARMA-30 as effectively as a system with 64 nodes. Comparable findings are observed in the damping delay-line environment and is further analysed.

4.3.2 Santa Fe Laser Task

We now showcase the simulation results for the Santa Fe laser task, as introduced earlier in Section 3.3.2. In Figure 4.6, NMSE values are displayed in boxplots. Here, similar to NARMA benchmark results, delay-line A exhibits the weakest performance (NMSE = 0.0025), while delay-line D achieves the most remarkable results (NMSE = 0.0012).

Similar to NARMA results, for the Santa Fe laser task, when comparing different numbers of outputs within the same delay-line configuration, the results remain consistent in the ideal environment implementation used in this study.

It is again demonstrated that for systems with an equal number of virtual nodes but different time intervals between nodes, the one with larger spacing achieves superior performance (refer to Figure 4.6 for delay-lines A and B or delay-lines C and D). Additionally, when systems have the same spacing between virtual nodes but vary in the number of nodes, the one with more nodes attains better results (as seen in Figure 4.6).

4.3.3 Summary

The findings presented in this section underscore the importance of considering multiple aspect at one and the same time when designing physical Delay-Feedback Reservoir Computing (DFRC) systems. It becomes evident

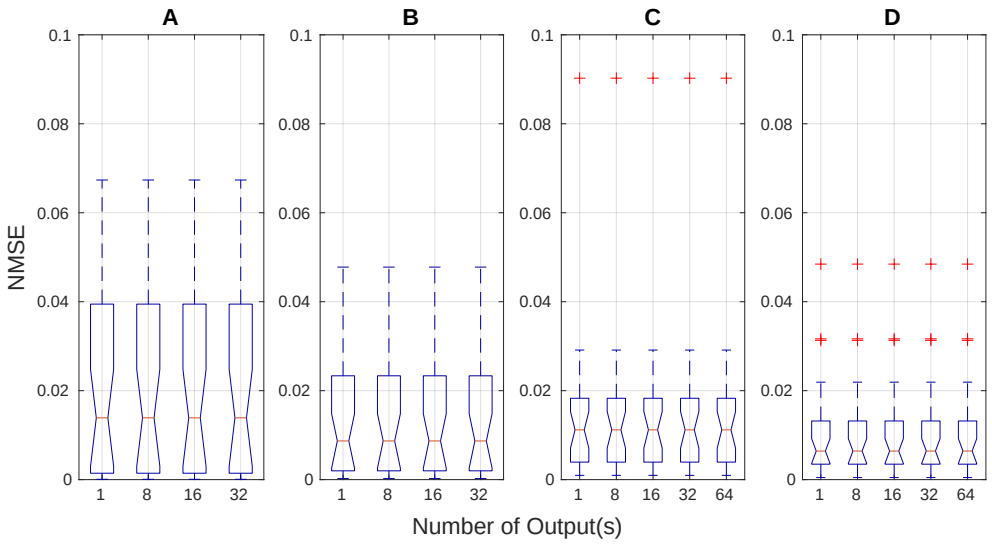


Figure 4.6: Santa Fe Laser task results for devirtualised Mackey-Glass system with ideal environment. The x–axis 'Number of Outputs(s)' indicates the actual output lines connected to the delay-line. A: N_{total}=32, θ=25ms; B: N_{total}=32, θ=50ms; C: N_{total}=64, θ=25ms; D: N_{total} =64, θ=50ms.

that various elements, such as the length of the delay-line, the count of virtual nodes along the delay-line, and the timescales [165] discussed in Section 3.2.2, are not isolated factors, even in the absence of damping. Instead, they are interdependent and influence one another. This interconnectedness is particularly crucial given that the non-linear Mackey–Glass node inherently possesses its own timing characteristics.

4.4 Ideal vs Physical implementation

In this section, the simulation platform of Mackey–Glass delay-feedback reservoir computing, incorporating a damping delay-line as detailed in Section 4.2, is used and compared with an ideal delay-line for each benchmark

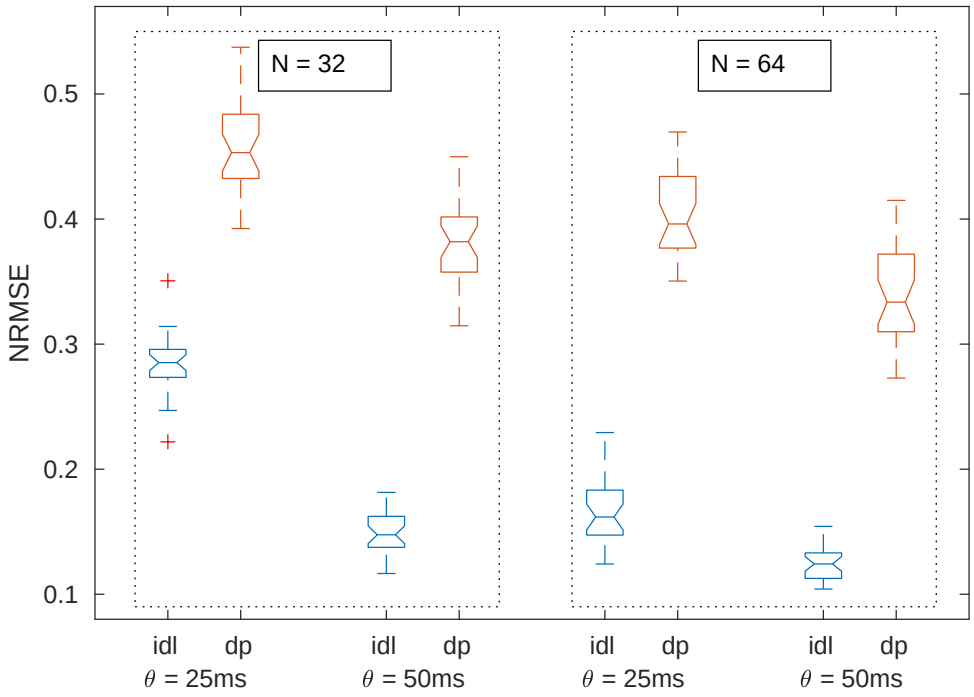
task. The objective of this comparison is to highlight the differences between the ideal and damping delay-line implementations. The results of NARMA and Santa Fe laser task are presented and discussed accordingly.

4.4.1 Results

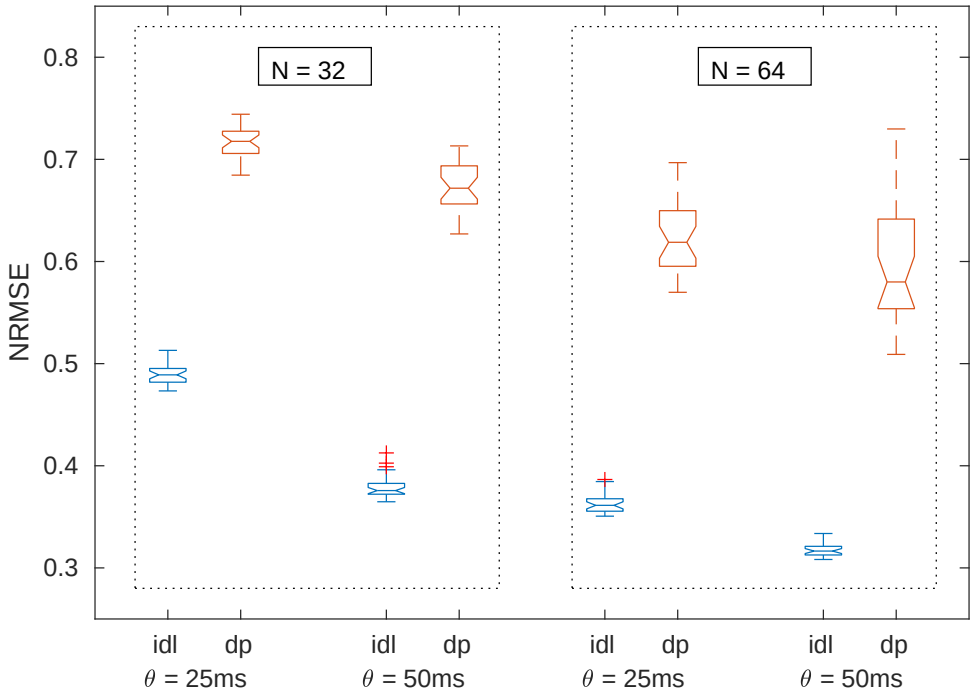
Figure 4.7 and Figure 4.8 display the experimental results for the NARMA and Santa Fe laser benchmarks, respectively, using both an ideal delay-line (represented in blue and marked as 'idl') and a damping delay-line (shown in orange and denoted as 'dp'). The displayed results represent the 'fully virtualised' scenario, in which only one single output line is needed, and the sampling rate is set to θ.

NARMA. The trends observed in the NARMA-10 (Figure 4.7a) and NARMA-30 (Figure 4.7b) tasks resemble those of the ideal delay-line (see Section 4.3). When comparing pairs with different numbers of virtual nodes but the same spacing between them, the delay-line with more nodes achieves superior results. Additionally, when comparing pairs with the same number of virtual nodes but varying spacing, the delay-line with a greater distance between virtual nodes exhibits improved performance.

Santa Fe laser task. The experimental results for the Santa Fe laser task can be found in Figure 4.8. It is important to observe that the overall performance of the damping delay-line exhibits similarities across the various cases. However, distinctions in the results among different damping delay-lines are evident in the third quartile and whiskers of each boxplot.



(a) NARMA-10



(b) NARMA-30

Figure 4.7: Experimental results to compare the ideal environment ('idl' in blue) and damping delay-line ('dp' in orange) with (a) NARMA-10 and (b) NARMA-30.

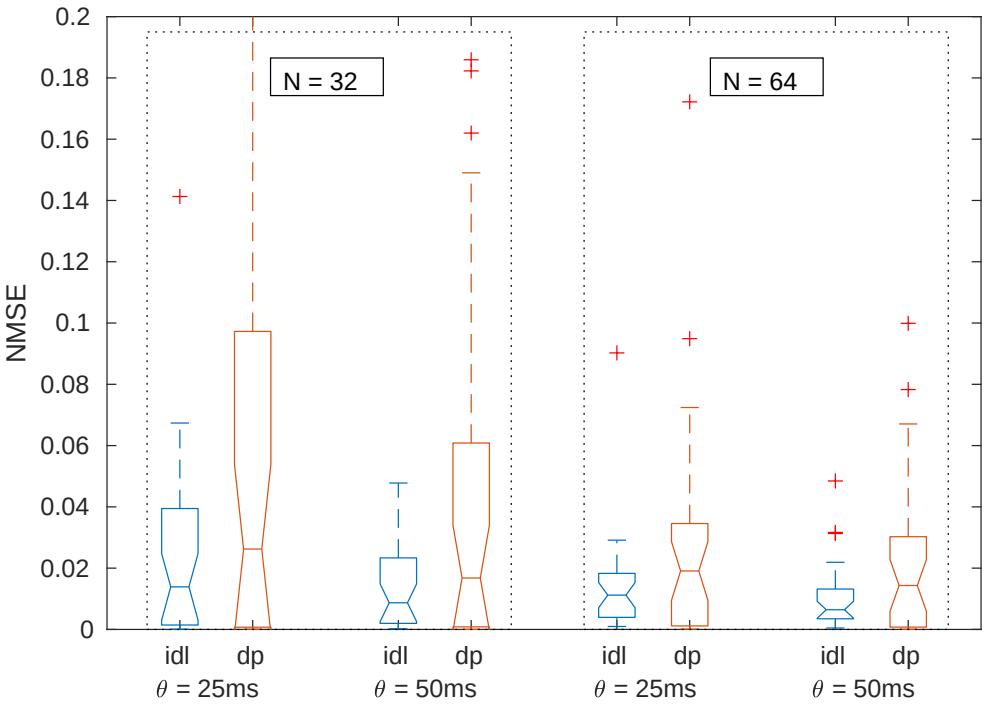


Figure 4.8: Experimental results to compare the ideal environment ('idl')and damping delay-line ('dp') with Santa Fe laser task. Other parameters are described in

Summary. For all the benchmark tasks, it is evident that the performance with an ideal delay-line surpasses that of a damping delay-line. This can be attributed to the fact that when signals travel through the damping delay-line, their amplitude diminishes, resulting in a loss of information.

4.5 Devirtualisation with Damping Delay-line

The previous sections have shown the fesibility of devirtualisation in both ideal and damping delay-line environments. In this section, we will delve deeper into various configurations of damping delay-lines to gain a more comprehensive understanding of devirtualisation approach.

As depicted in Figure 4.4, our experiment uses four damping delay-lines that vary either in the number of virtual nodes or in the length of the line. To explore the performance of the devirtualisation approach in various situations, we assess the effectiveness of the four delay-lines in two distinct scenarios: *a*) Constant damping rate over delay-line and *b*) Constant damping rate per 'length' with NARMA-10 and NARMA-30 tasks. The results for Santa Fe laser task are shown in Appendix.

Constant damping rate over delay-line. In this scenario, we keep the damping rate consistent across all four delay-lines by modifying the 'Gain' values in the simulation platform. Preserving a constant damping rate enables more accurate comparisons between different delay-line configurations. This leads to a better understanding of how varying lengths and virtual nodes influence the system's performance. Furthermore, sustaining a constant damping rate ensures that the system's stability is not compromised

Delay-line	N_v	θ (ms)	τ (ms)	$A_{low}(\%)$	$A_{mid}(\%)$	$A_{high}(\%)$
A	32	25	800	0.4	2	8
B	32	50	1600	0.4	2	8
C	64	25	1600	0.2	1	4
D	64	50	3200	0.2	1	4

Figure 4.9: Parameter values of four delay-lines for the scenario of 'Constant damping rate over delay-line' experiments. N_v: Number of virtual nodes; θ: time delay between each virtual node; τ: time delay over the delay-line; $A_{low}(\%)$, $A_{mid}(\%)$ and $A_{high}(\%)$ here indicates the parameter settings for 'Gain' block in the simulation (see Section 4.2 for details). 'Low', 'Mid' and 'High' here indicates the overall damping level of delay-line, where the value approximately equals to 10%, 50% and 90% respectively.

by alterations in delay-line length, which is essential when investigating the effects of various configurations on system performance.

The parameters depicted in Figure 4.9 include:

- N_v: The number of virtual nodes within a delay-line.
- θ: The time delay occurring between each successive virtual node.
- τ: The total time delay experienced across the entire delay-line.
- $G_{low}(\%)$, $G_{mid}(\%)$, and $G_{high}(\%)$: These values represent the parameter settings for the 'Gain' block in our simulation. For more information on these settings, refer to Section 4.2.

The terms 'Low', 'Mid', and 'High' correspond to the overall damping level of a given delay-line. These levels are approximately equal to 10%, 50%, and 90% damping, respectively.

Delay-line	N_v	θ (ms)	τ (ms)	$G_{low}(\%)$	$G_{mid}(\%)$	$G_{high}(\%)$
A	32	25	800	0.5	1	2
B	32	50	1600	1	2	4
C	64	25	1600	0.5	1	2
D	64	50	3200	1	2	4

Figure 4.10: Parameter values of four delay-lines for the scenario of 'Constant damping rate per length' experiments. N_v: Number of virtual nodes; θ: time delay between each virtual node; τ: time delay over the delay-line; $G_{low}(\%)$, $G_{mid}(\%)$ and $G_{high}(\%)$ here indicates the parameter settings for 'Gain' block in the simulation (see Section 4.2 for details). 'Low', 'Mid' and 'High' here indicates the damping rate of 1%, 2% and 4% per 'Length' (see text for definition of 'Length').

Constant damping rate per 'length'. In this scenario, we maintain a consistent damping rate for every unit 'length' across the four delay-lines. In our experiment, the term 'length' denotes the distance between virtual nodes, which corresponds to the spatial extent covered by the signal as it travels for 50ms to reach the adjacent virtual node.

Figure 4.10 depicted the details of delay-lines A–D, where delay-lines B and D represent the idea of standard unit 'length'. For each unit 'length', we investigate damping rates labeled as 'low', 'mid', and 'high', which correspond to 1%, 2%, and 4% damping, respectively.

This scenario aims to explore the impact of overall damping on the idea of the devirtualisation approach and provide insights into choosing the appropriate length for the physical delay-line.

4.5.1 Results

In this paragraph, we investigate the devirtualisation approach for both 'constant damping rate over delay-line' and 'constant damping rate per unit length' scenarios using the NARMA-10 and NARMA-30 benchmarks. The 10th– and 30th–order NARMA equations can be found in Section 3.3.1, while additional information on implementing the experiments is provided in Section 4.2.2.

Constant damping rate over Delay-line

By maintaining a consistent damping rate across all delay-lines, we obtain the experimental results for NARMA-10 and NARMA-30, as shown in Figure 4.11 and Figure 4.12, respectively. The parameters of system such as nonlinearity, feedback strength and input scaling are presented in Figure 4.3.

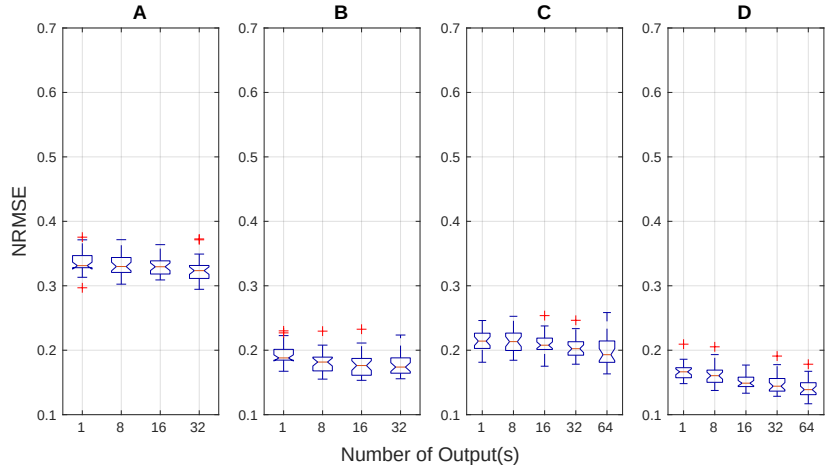
When comparing the results for one delay-line under various damping rates, such as delay-line A in Figure 4.11b or delay-line D in Figure 4.12c, it becomes clear that the system's performance improves with more output lines, albeit at different levels of enhancement. Optimal performance is consistently attained when the number of output lines equals the number of virtual nodes in the system. This observation is consistent for both NARMA-10 and NARMA-30 tasks, which strengthens our confidence that the performance of physical delay-feedback reservoir computing can benefit from the devirtualisation approach along the damping delay-line.

The impact of varying levels of enhancement becomes more apparent when comparing results across different damping rates. The trend in performance results becomes steeper when transitioning from a 'low' damping rate

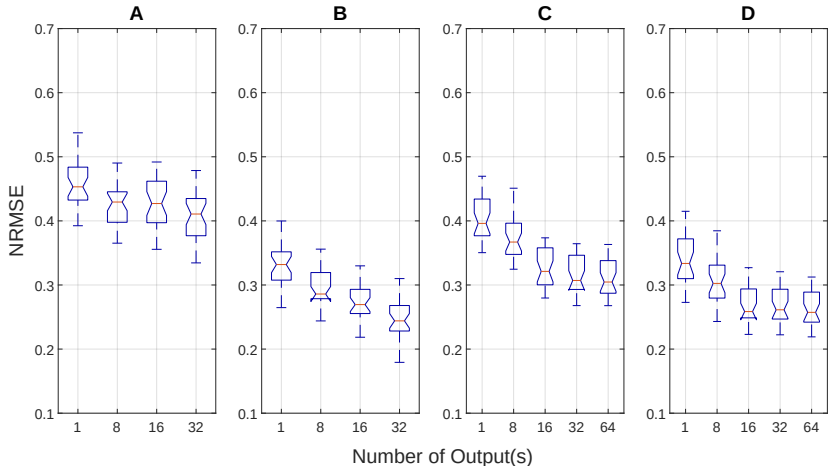
scenario to a ‘high’ damping rate scenario.

When comparing the results from the same delay-line under different damping rates, such as delay-line B in the ‘Low’, ‘Mid’, and ‘High’ cases, it is apparent that the overall performance is influenced by the damping rate level. The results tend to deteriorate when the delay-line exhibits a higher damping rate. For instance, the NARMA-10 results as shown in Figure 4.11, when delay-line A is fully devirtualised, the NRMSE values are as follows: 0.3231 in the ‘low’ damping case, 0.4198 in the ‘mid’ damping case, and 0.5503 in the ’high’ damping case. When the delay-line exhibits a low damping rate, the results more closely align with those of an ideal delay-line, as shown in Figure 4.5a and Figure 4.5b for NARMA-10 and NARMA-30 tasks, respectively.

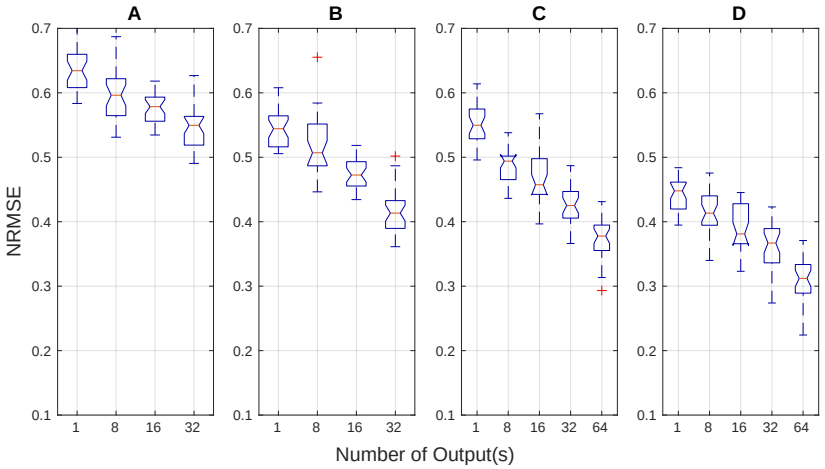
The reason for the above observation is that, higher damping rates can be interpreted as causing more signal attenuation, i.e., a greater reduction in the signal’s amplitude as it travels along the delay-line. This may impact the readout from virtual nodes located near the end of the delay-line and lead to the consequence of poor performance.



(a) Low damping 10%

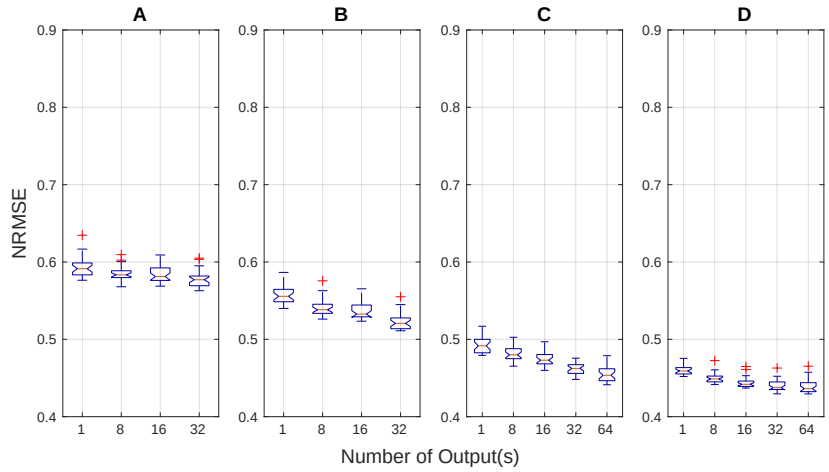


(b) Mid damping 50%

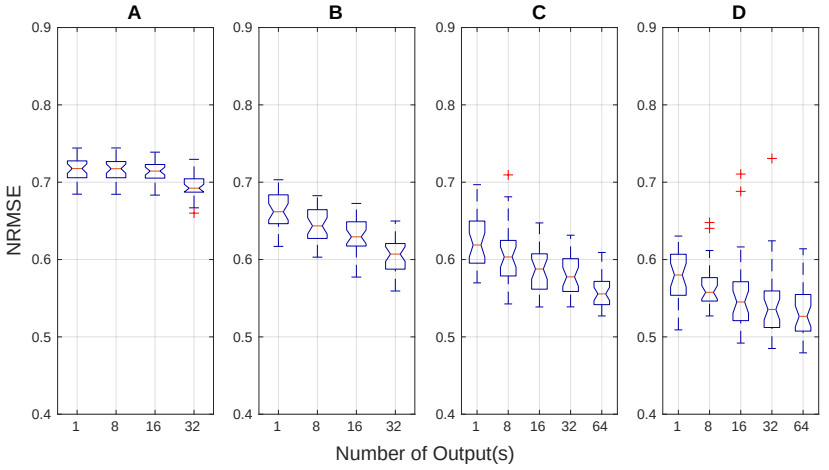


(c) High damping 90%

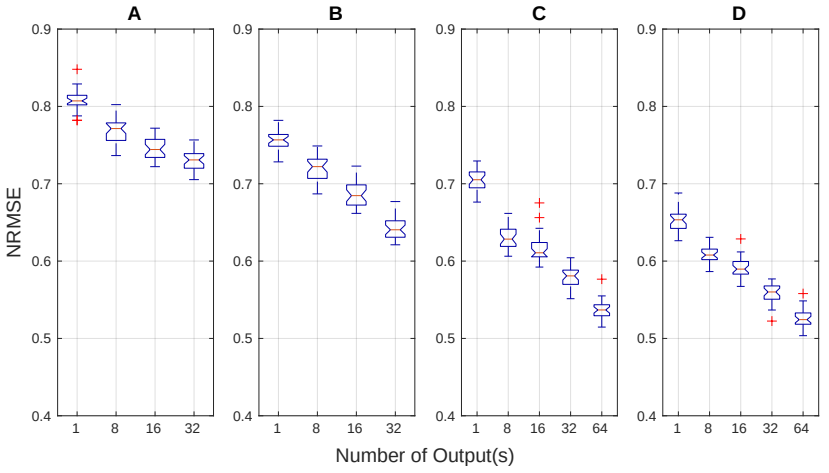
Figure 4.11: NARMA-10 results for constant damping rate over delay-line. The x–axis 'Number of Outputs(s)' indicates the actual output lines connected to the delay-line.



(a) Low damping 10%



(b) Mid damping 50%



(c) High damping 90%

Figure 4.12: NARMA-30 results for constant damping rate over delay-line. A: N_{total}=32, θ=25ms; B: N_{total}=32, θ=50ms; C: N_{total}=64, θ=25ms; D: N_{total} =64, θ=50ms.

Delay-line	$Overall_{low}(\%)$	$Overall_{mid}(\%)$	$Overall_{high}(\%)$
A	14.8	27.5	47.6
B	27.5	47.6	72.9
C	27.5	47.6	72.9
D	47.6	72.9	92.7

Figure 4.13: Overall damping of delay-lines based on different damping rate per length. The values are calculated based on Eq. 4.1.

Constant damping rate per Length

In this experiment, we maintain a constant damping rate per unit length of the delay-line, which means that the overall damping for each delay-line may differ. This leads to difference in overall damping of each delay-line as shown in Figure 4.13. The values are calculated based on Eq.4.1, where G stands for gains of damping blocks (see Figure 4.10).

Each value within the table are calculated based on the damping rate in Figure 4.10 as:

$$Overall_{low/mid/high} = 1 - (1 - G_{low/mid/high})^{N_v} \tag{4.1}$$

To facilitate easier comparison and analysis of the results, the overall damping values are provided in Figure 4.13.

Figure 4.14 and Figure 4.15 show the experimental results of NARMA-10 and NARMA-30 tasks respectively. The results in each box of both figures show a trend of performance improvement with an increasing number of output lines, further validating the effectiveness of the devirtualisation approach.

The variations in the trend of performance observed in this scenario high-

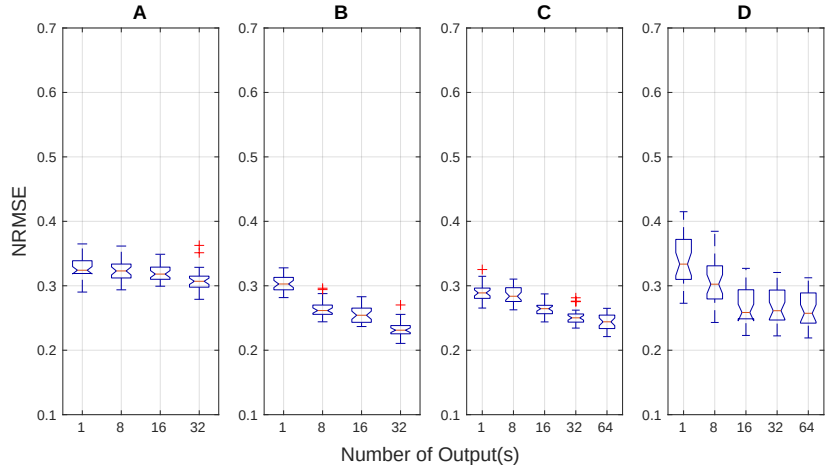
light the dependence on the overall damping of the delay-line. For example, delay-line A and delay-line D in Figure 4.14a exhibit differing trends, as delay-line A has an overall damping of 14.8%, while delay-line D has an overall damping of 47.6%.

Differ from the previous experiment in Section 4.5.1, when comparing the results horizontally in Figure 4.14 and Figure 4.15, with constant damping per unit length, the results are roughly within the same range. In other words, the performance difference between each delay-line under the same per unit length damping rate is not as significant as in last experiment. This is because the overall damping rates of each delay-line are now different, resulting in delay-line A, with a lower overall damping rate, achieving performance that compensates for its limited number of virtual nodes and time lag. On the other hand, delay-line D, with a higher overall damping rate, exhibits worse performance compared to the results of the experiment with constant damping rate over the delay-line.

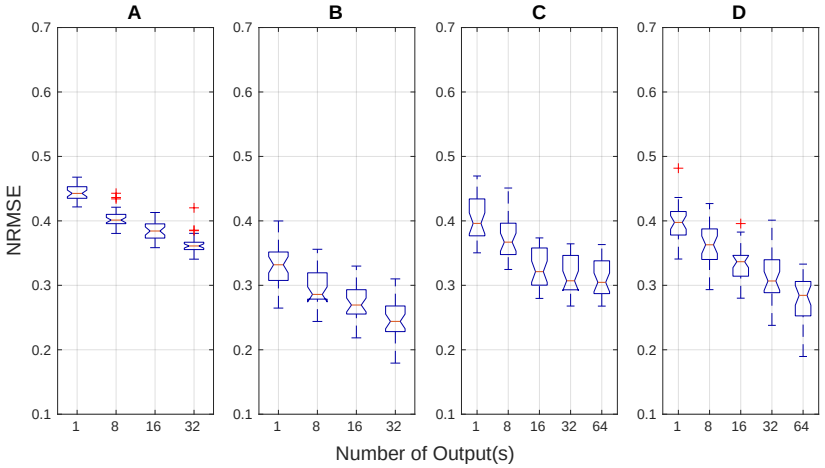
4.5.2 Discussion

In this section, we discuss the results obtained from our investigation of the devirtualisation approach for both 'constant damping rate over delay-line' and 'constant damping rate per unit length' scenarios using the NARMA-10 and NARMA-30 benchmarks. The main goal is to assess the effectiveness of the devirtualisation approach in physical delay-feedback reservoir computing when considering damping delay-lines.

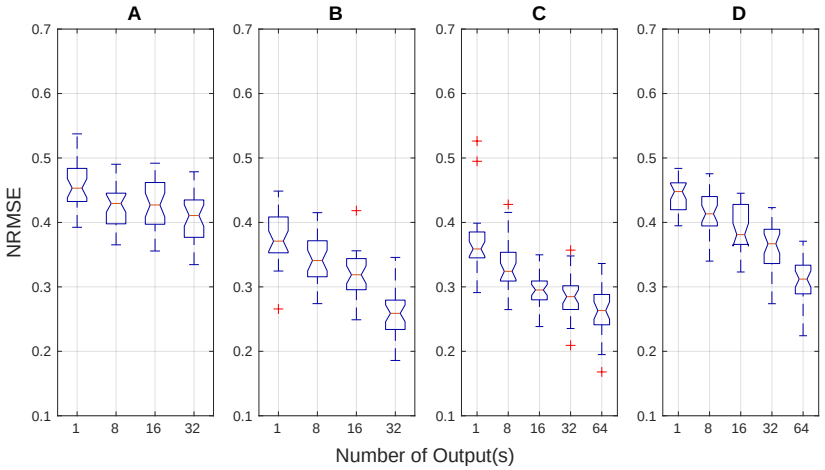
For the 'constant damping rate over delay-line' scenario, the results clearly show that the system's performance improves with an increasing number of



(a) Low damping 1% per length

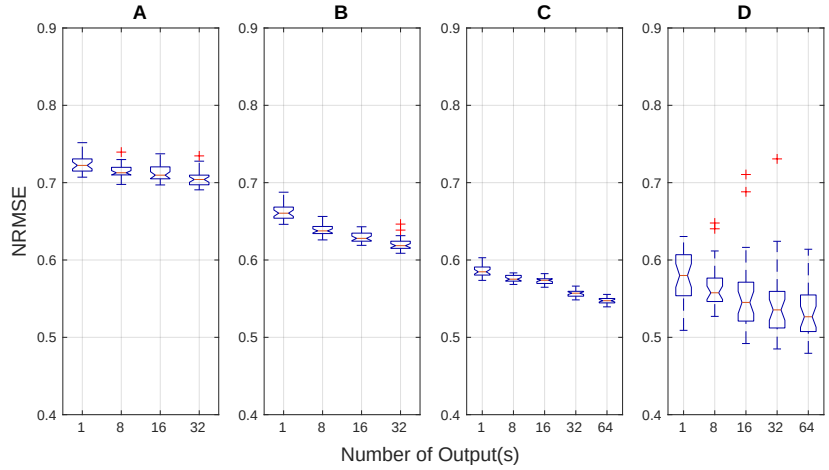


(b) Mid damping 2% per length

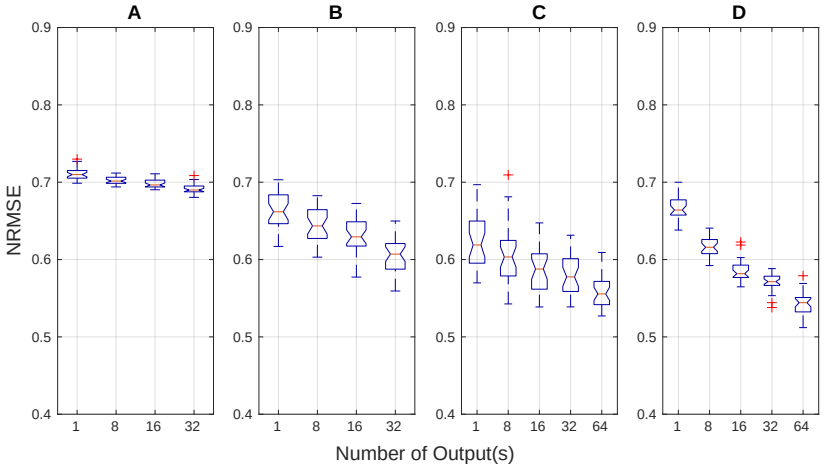


(c) High damping 4% per length

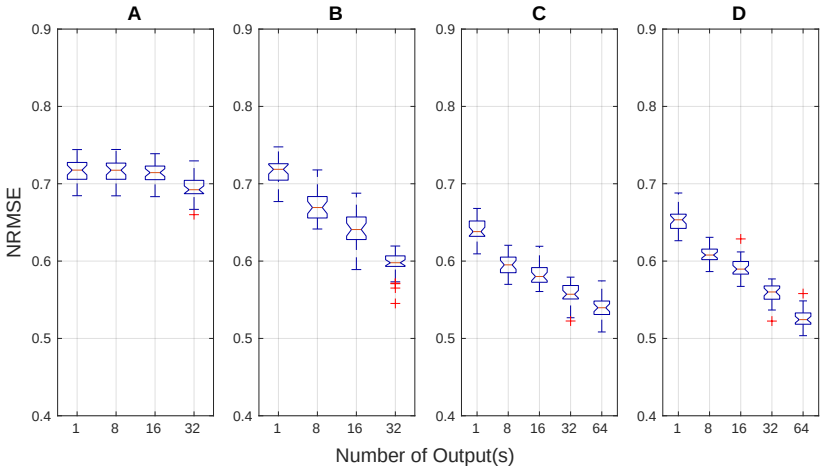
Figure 4.14: NARMA-10 results for constant damping rate per length. A: N_{total}=32, θ=25ms; B: N_{total}=32, θ=50ms; C: N_{total}=64, θ=25ms; D: N_{total} =64, θ=50ms.



(a) Low damping 1% per length



(b) Mid damping 2% per length



(c) High damping 4% per length

Figure 4.15: NARMA-30 results for constant damping rate per length. A: N_{total}=32, θ=25ms; B: N_{total}=32, θ=50ms; C: N_{total}=64, θ=25ms; D: N_{total} =64, θ=50ms.

output lines. Optimal performance is consistently achieved when the number of output lines equals the number of virtual nodes in the system, which suggests that the devirtualisation approach can indeed benefit the performance of physical delay-feedback reservoir computing.

However, the level of enhancement differs across various damping rates. When comparing results across different damping rates, the trend in performance becomes steeper when transitioning from a 'low' damping rate scenario to a 'high' damping rate scenario. This observation can be attributed to the higher damping rates causing more signal attenuation, impacting the readout from virtual nodes located near the end of the delay-line, which in turn leads to poorer performance.

For the 'constant damping rate per unit length' scenario, the results continue to show a trend of performance improvement with an increasing number of output lines, further validating the effectiveness of the devirtualisation approach. However, the performance difference between each delay-line under the same per unit length damping rate is not as significant as in the previous experiment. This is because the overall damping rates of each delay-line are now different, resulting in varying performance levels depending on the specific delay-line.

In summary, the devirtualisation approach proves to be effective in improving the performance of physical delay-feedback reservoir computing, particularly for the NARMA-10 and NARMA-30 tasks. The results highlight the importance of considering the damping rates, both over the entire delay-line and per unit length, when optimizing system performance. By carefully selecting the appropriate damping rates and number of output lines, it is

possible to achieve optimal performance in physical delay-feedback reservoir computing systems.

4.6 Summary

In this chapter, we presented a comprehensive investigation of the devirtualisation approach in physical delay-feedback reservoir computing systems, mainly focusing on the impact of damping delay-lines. We aimed to understand how different damping rates and number of output lines affect the system's performance with the approach. We assessed the performance of these systems using the NARMA-10, NARMA-30 and Santa Fe laser benchmarks.

We began by introducing the concept of 'devirtualisation'. Physical delay-feedback reservoir computing systems use a single nonlinear node and a delay-line to emulate a high-dimensional reservoir of virtual nodes. We then discussed the challenges and potential limitations of using delay-lines, including the impact of signal damping on system performance.

We proceeded to contrast the findings derived from both the ideal delay-line and the damping delay-line. Subsequently, we introduced the fully damped delay-line in two distinct scenarios: 'constant damping rate over delay-line' and 'constant damping rate per unit length.'

The results demonstrated that the devirtualisation approach effectively improved the performance of physical delay-feedback reservoir computing systems. For both damping scenarios, the performance consistently improved with an increasing number of output lines, reaching optimal performance when the number of output lines equaled the number of virtual nodes. How-

ever, the extent of the enhancement differed across various damping rates, with the performance becoming steeper when transitioning from a low to a high damping rate.

In conclusion, our study highlights the importance of carefully considering damping rates and the number of output lines to achieve optimal performance in physical delay-feedback reservoir computing systems. The devirtualisation approach can significantly enhance the performance of these systems, particularly for tasks such as NARMA-10 and NARMA-30. Further research may explore the impact of other system parameters and benchmark tasks to broaden our understanding of the devirtualisation approach's potential.

5 Masking

In this chapter, we explore the assumptions regarding input masking, a crucial procedure of delay-feedback reservoir computing. The function of masking is to define virtual nodes along the delay-line, rendering it an integral element in this computational paradigm. The quality of virtual nodes is contingent upon the process of masking.

This chapter discusses two aspects of the masking procedure:

1. We conduct a rigorous experimental comparison between two types of masking signals, *Binary Weight Mask (BWM)* and *Random Weight Mask (RWM)*, which are predominantly referenced in scholarly literature. The performance of these masking signals is evaluated using benchmark tasks *NARMA-10*, *NARMA-30*, and the *Santa Fe laser task*. Owing to the ubiquitous application of these tasks in relevant studies, we anticipate that our findings will present a universal understanding of the performance of the different masking signals.

2. We experimentally investigate how the distribution (**randomness**) of *Binary Weight Mask* signal impacts the performance of Delay-Feedback Reservoir Computing, under two circumstances of *No-offset* and *Offset*.

The principles of masking and virtual nodes are introduced in Section 5.1.

In Section 5.2, a comprehensive account of the experimental configurations employed in both the simulation environment and benchmark tasks for each experiment is provided. Section 5.3 provides a comparative analysis of the experimental outcomes between *Binary Weight Mask* and *Random Weight Mask*, using benchmarks NARMA-10, NARMA-30, and the Santa Fe laser task. The findings from the second experiment, which probes into the masking distribution of randomness, are given in Section 5.4. Concluding the chapter, Section 5.5 offers a discussion of the topics covered throughout.

5.1 Concept

In delay-feedback reservoir computing, "masking" refers to a technique used to transform the input data before it is fed into the system, which is analogue to the input weight matrix in a standard reservoir. This transformation is typically done by multiplying the input data by a certain mask, which can be a binary or random sequence. The purpose of this process is to increase the dimensionality and complexity of the input data, which can help improve the performance of the reservoir computing system.

The concept of "virtual nodes" is closely related to masking in this context. In a delay-feedback reservoir, the "reservoir" is essentially a single nonlinear node with a delayed feedback loop. The delay line can be thought of as being divided into several segments or "virtual nodes", each of which represents the state of the system at a different point in time. The input data, after being transformed by the mask, is added to the state of the system at each of these virtual nodes.

The model's input stream is composed of values signifying discrete time measurements. In contrast, a physical reservoir operates in continuous real time. The input signals undergo a pre-processing phase, involving a *Sample and Hold* operation for input discretisation and a mask-multiplexing procedure to define the virtual nodes, before being fed into the reservoir layer.

As depicted in Fig.5.1, the discrete input stream, denoted as $u(k)$, is treated as a continuous time input $u(t)$. This is then transformed into a piece-wise signal function $I_0(t)$ for $\tau k \leq t < \tau(k+1)$, which remains constant throughout one delay period τ.

$M(t)$ here represents the masking, which is applied to $I(t)$ at each time

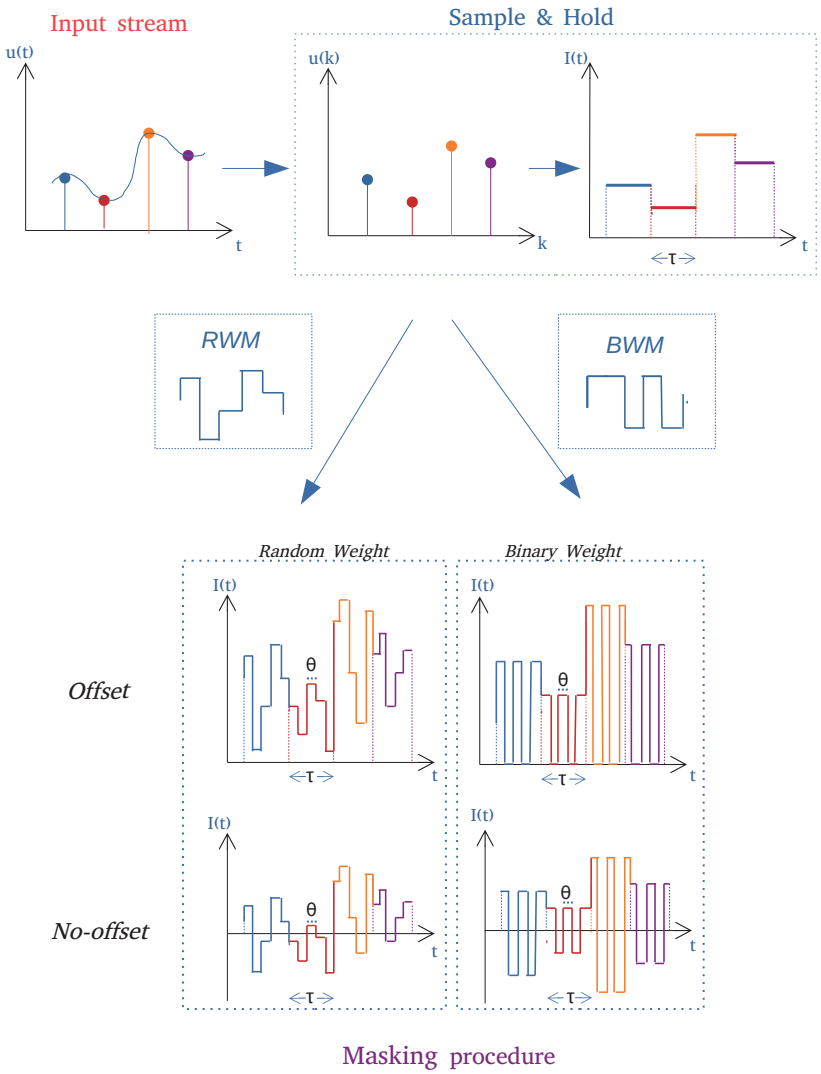


Figure 5.1: Time-multiplexing procedure. Top left: the discrete inputs of the delayed feedback reservoir model. Bottom left: physical continuous inputs. Middle: Sample and Hold discretising the physical inputs. Right: four masking procedures applied to the discretised inputs.

interval τ, thereby supplying the input to the reservoir. The mask $M(t)$ comprises a sequence of N randomly generated values, each of time period θ, with the same mask being used for each interval τ, i.e., $\tau = \theta * N$.

5.2 Experiment Setups

The quality and type of the mask can have a significant impact on the performance of the system. Different tasks may require different types of masks, and part of the challenge in this field is finding the most effective mask for a given task.

5.2.1 Experimental scenarios

In this section, different tasks and experimental scenarios for Experiment A: Comparison between BWM and RWM and Experiment B: Distribution of "Randomness" are addressed, to demonstrate a universal results for masking. The value of u is chosen at 0.5 in the following sections and experiment.

Experiment A: Comparison between BWM and RWM

For the first experiment, we evaluate the computational performances of Delay-Feedback Reservoir Computing between *Binary Weight Mask (BWM)* and *Random Weight Mask (RWM)*; among different masking procedures under three experimental configurations for benchmark tasks NARMA-10, NARMA-30 and Santa Fe laser task:

1) Mask-multiplexing with *No-offset*:

$$I(t) = I_0(t)M(t) \tag{5.1}$$

2) Mask-multiplexing with offset by u:

$$I(t) = I_0(t)(M(t) + 1) \tag{5.2}$$

3) Mask-multiplexing with offset by $2u$:

$$I(t) = I_0(t)(M(t) + 2) \tag{5.3}$$

where the original input values $I_0(t)$ are drawn from the interval $[0, u]$, the ranges of the masked input signal $I(t)$ are $[-u, u]$, $[0, 2u]$ and $[u, 3u]$ for the offsets of Eq.5.1, Eq.5.2 and Eq.5.3 respectively.

Experiment B: Distribution of "Randomness"

In this experiment, we investigate the affect of *No-offset* and *Offset*, and the distribution of "randomness" when generate the masking signals. NARMA-10 as a baseline benchmark is used in this experiment to demonstrate the performance of Delay-Feedback Reservoir Computing (DFRC) with offset varying between $[-u, u]$ or $[0, 2u]$ (Eq.5.1 and Eq.5.2, respectively) for *Binary Weight Mask*. The choice of scenarios/values in this experiment is affected by the results of experiment A.

5.2.2 Simulation Environment

Both experiments A and B are implemented with MATLAB/SIMULINK model of Mackey–Glass equation (Eq. 3.6 with an ideal delay-line (Figure. 3.4). The parameter configurations used in our experimental setup are given in Figure 5.2.

In Glass et al. [4], the nonlinear parameter n was given a value of 9.65, to induce chaos in the system. In that model the state value corresponds to blood cell concentration, and so is always positive. Here, the state value represents voltage, and is combined with various inputs and subjected to masking procedures, resulting in the possibility for the term $x_\tau + \delta I_t$ – the sum of the state value and input – to be negative.

To avoid the complications associated with raising negative numbers to fractional powers, we restrict the value of n to be an integer. For our investigations, we consider $n = 6$ and $n = 7$ for NARMA task; these values span the appropriate region determined by the Mackey–Glass mathematical model (Eq. 3.6) and present a study of the system's behaviour when raising to an odd power – where negative values remain negative; and to an even power – where negative values become positive. For the Santa Fe laser task, we selected values of n to be 2 and 4. This choice is informed by preliminary experiment. When n is set to 3, with the specified parameters for the Mackey-Glass model (see Table. 5.2), the No-offset scenario yields NaN values. This suggests that they have reached the error bar's limits. The plot is shown in Appendix. In earlier research by Appeltant et al. [173], n is set to 1 with different parameter configurations. However, as shown in Figure.3.7, the nonlinear curve for $n = 1$ differs from what we use in this thesis.

5.2.3 Tasks

NARMA

For the NARMA experiments, the input sequence length generated by NARMA-10 and NARMA-30 (see Eq. 3.8 and Eq. 3.9, respectively) is $L = 3000$. This

sequence is divided into separate datasets for training and testing purposes, with $L_{train} = 1700$ and $L_{test} = 1200$. The initial $L_{washout} = 100$ results of each sequence are discarded.

Each experiment is repeated 30 times to ensure reliability and robustness of results. We use three different configurations to investigate whether the randomness of input signals or of the masking is dominating the computational performance of the system: i) SIDM has the same random input sequence but different random masks for each run; ii) DISM has different random input sequences but the same random masks for each run; iii) DIDM has different input sequences and different masks for each run.

Santa Fe Laser task

The Santa Fe laser data prediction task is employed as an illustrative example of one-step time series forecasting. The dataset used for this task consists of 4000 data points, which are divided into four separate samples, with each sample containing 1000 data points. In accordance with the existing literature, the normalised mean square error (NMSE) is employed as the standard metric for evaluating performance on this task. In our experiments, we use the NMSE to compare the predicted values with their corresponding actual values. The detailed definition of NMSE can be found in Section 3.11.

Based on the objectives elucidated in this chapter, the nonlinear value, delay time, and the configurations of the virtual nodes were selected. These parameters are primarily derived from the study by [4]. Further details of the experimental parameter settings can be consulted in Figure 5.2.

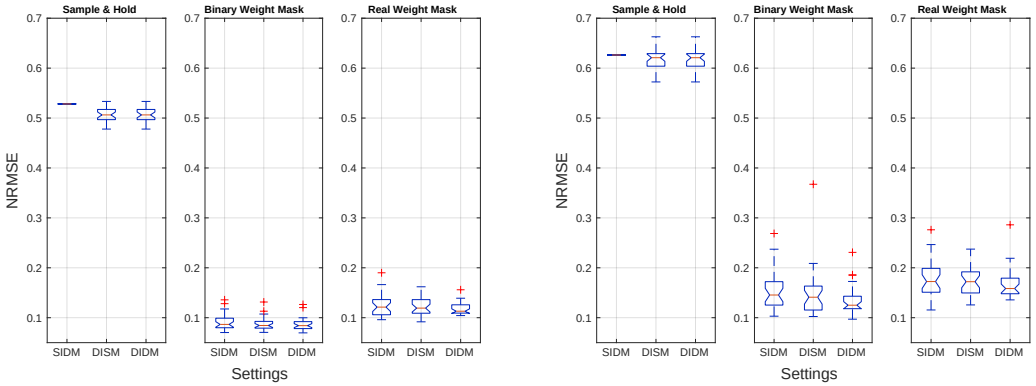
parameter	value	description
β	0.8	coupling factor
γ	1	decay rate
τ (NARMA)	3.2 (sec)	delay in feedback loop
τ (Santa Fe)	0.64 (sec)	delay in feedback loop
n (NARMA)	6 and 7	nonlinearity
n (Santa Fe laser)	2 and 4	nonlinearity
δ	0.1	input scaling
N	64	number of virtual nodes

Figure 5.2: Parameter values for the Mackey–Glass delay-line reservoir experiment. The Mackey–Glass system parameter values are from [4]; n is required to be an integer (see text for details).

5.3 Experiment A: Comparison between BWM and RWM

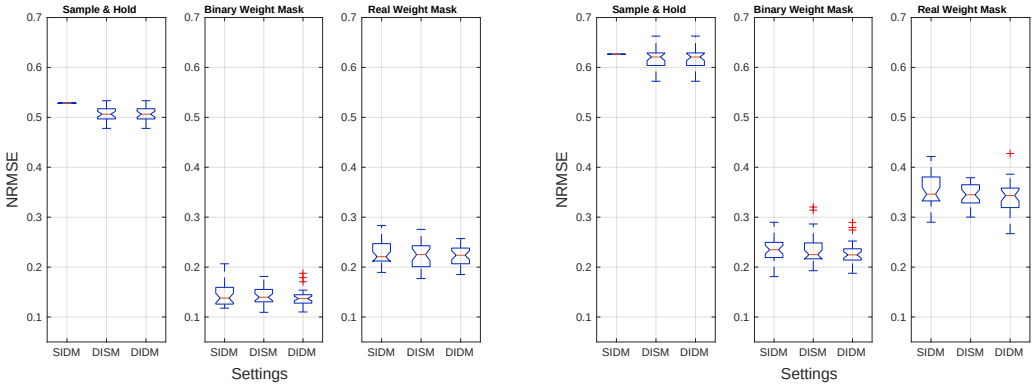
In this section, we experimentally compare the most commonly used masking types: *Binary Weight Mask* and *Random Weight Mask*. The experimental results for benchmarks NARMA-10, NARMA-30 and Santa Fe Laser task are shown in Figure. 5.3, Figure. 5.4 and Figure 5.5respectively.

We use Matlab's non-parametric statistical ranksum test to calculate the p value (probability that the effect appears by chance) and Vargha–Delaney's A [174] to calculate the effect size (calculated only when $p < 0.05$, a statistically significant difference at the 95% confidence level). An effect size $A > 0.64$ is a medium effect; $A > 0.71$ is a large effect.



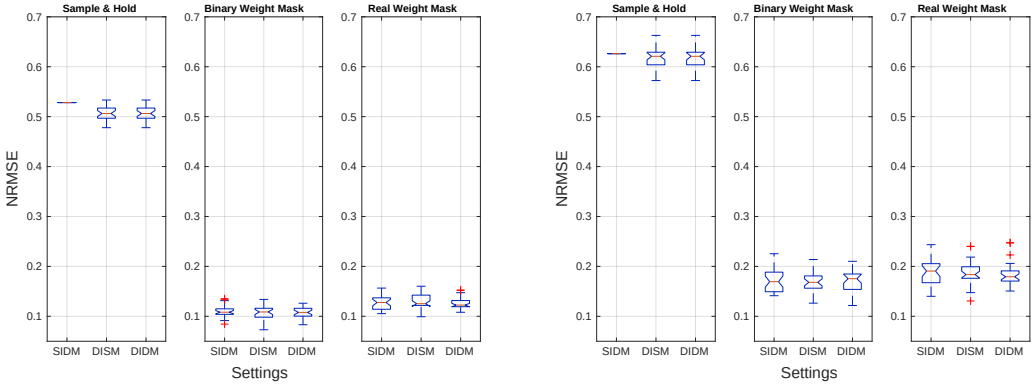
(a) $n = 6$ – No offset $[-u, u]$

(b) $n = 7$ – No offset $[-u, u]$



(c) $n = 6$ – Offset $[0, 2u]$

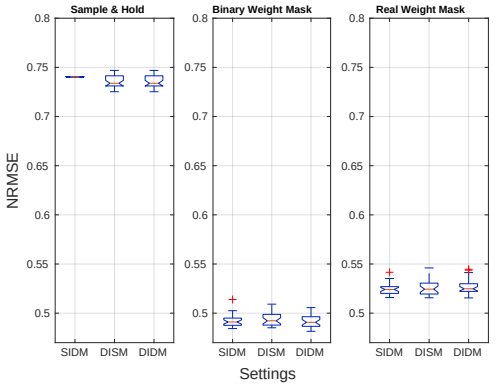
(d) $n = 7$ – Offset $[0, 2u]$



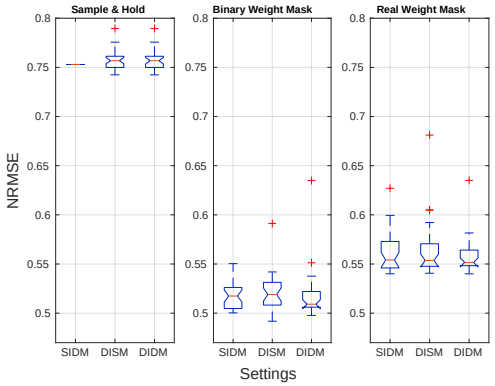
(e) $n = 6$ – Offset $[u, 3u]$

(f) $n = 7$ – Offset $[u, 3u]$

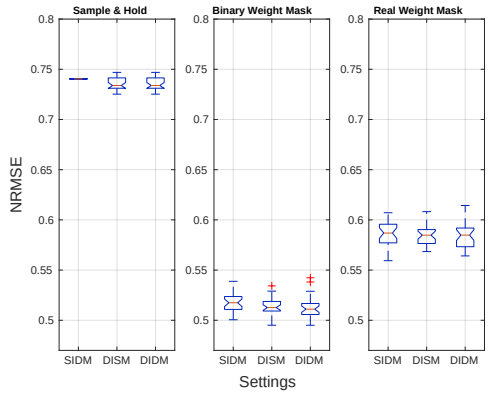
Figure 5.3: Simulation results for NARMA-10 benchmark task with two nonlinearity parameters: $n = 6$ (fig.5.3a, fig.5.3c and fig.5.3e), $n = 7$ (fig.5.3b, fig.5.3d and fig.5.3f) base on different time-multiplexing functions: Eq.5.1, Eq.5.2 and Eq.5.3. Each experiment is based on 30 runs.



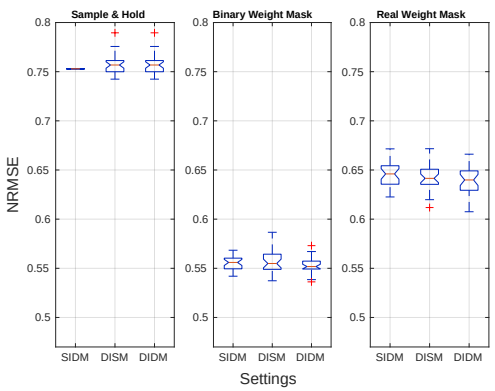
(a) $n = 6$ – No offset $[-u, u]$



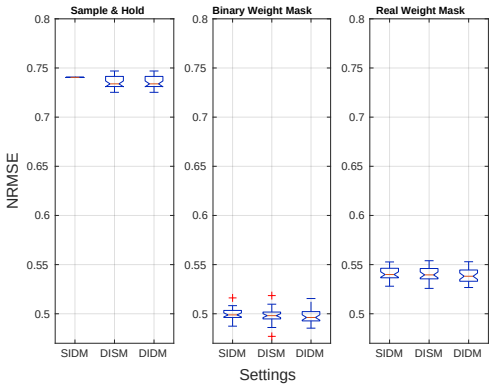
(b) $n = 7$ – No offset $[-u, u]$



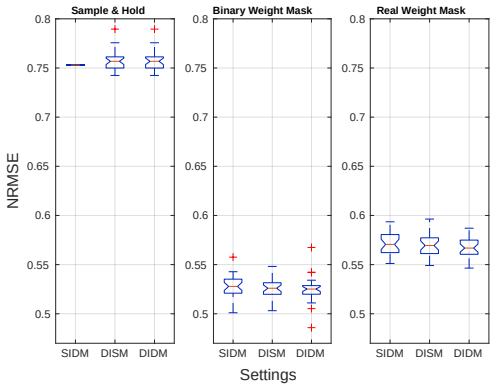
(c) $n = 6$ – Offset $[0, 2u]$



(d) $n = 7$ – Offset $[0, 2u]$

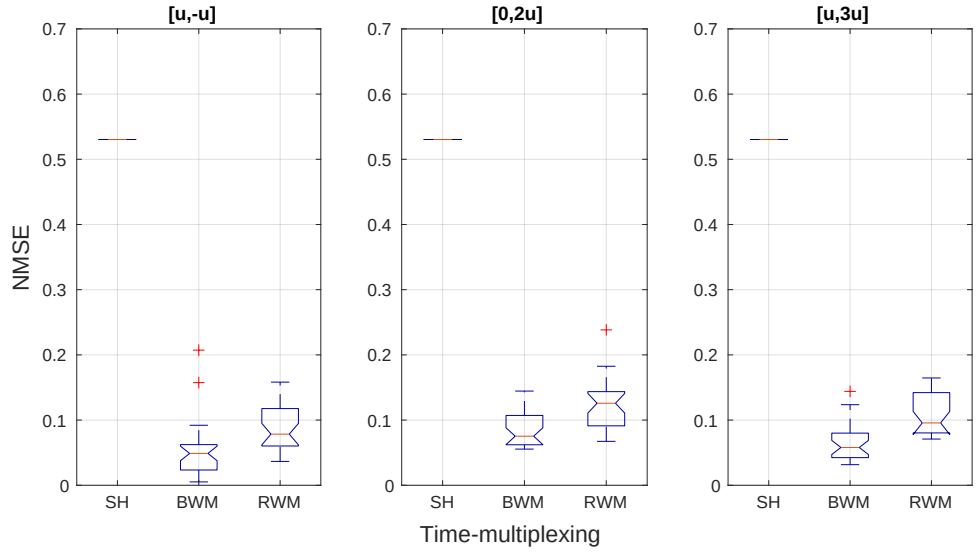


(e) $n = 6$ – Offset $[u, 3u]$

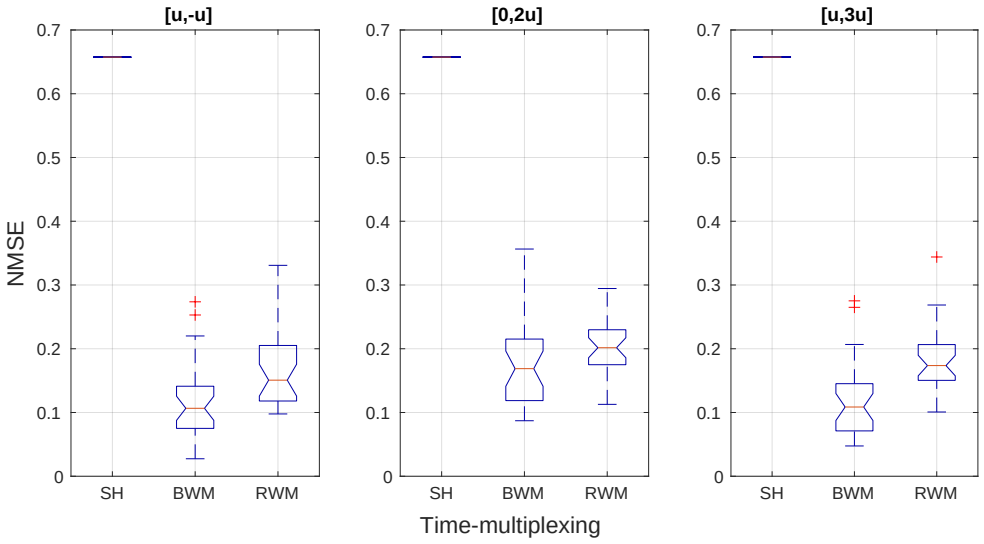


(f) $n = 7$ – Offset $[u, 3u]$

Figure 5.4: Simulation results for NARMA-30 benchmark task with two nonlinearity parameters: $n = 6$ (fig.5.4a, fig.5.4c and fig.5.4e), $n = 7$ (fig.5.4b, fig.5.4d and fig.5.4f) base on different time-multiplexing functions. Each experiment is based on 30 runs.



(a) $n = 2$



(b) $n = 4$

Figure 5.5: Simulation results for Santa Fe Laser Task with *No-offset*, offset by u and $2u$ scenarios.

5.3.1 The need for Mask

The *Sample and Hold* procedure has a constant input over the whole period τ (effectively, an $N = 1$ reservoir). Therefore, for SIDM, where we feed in the same input sequence for each run, the NRMSE error is constant. For DISM and DIDM we have the same sequence of random data, and (implicitly) the same mask, so these scenarios have identical NMRSE distributions. This control experiment also demonstrates that *Sample and Hold* does not provide enough complexity of the input data stream, and that masking is necessary to provide sufficient performance.

5.3.2 Binary Weight Mask *vs* Random Weight Mask

First, we compare the performance of Binary Weight Mask and Random Weight Mask.

NARMA

The experimental results for NARMA-10 and NARMA-30 are depicted in box-plots in Figures 5.3 and 5.4, with a summary of the NRMSE values in DIDM configuration (the right column of charts) provided in Tables 5.1 and 5.2, respectively.

The Binary Weight Mask exhibits the lowest median NRMSE values, which in *No-offset*, offset by u and $2u$ arrangements are 0.082, 0.132, and 0.114 ($n = 6$), 0.129, 0.221, and 0.172 ($n = 7$) in NARMA-10 result; 0.481, 0.522 and 0.497 ($n = 6$) and 0.524, 0.558 and 0.538 ($n = 7$) in NARMA-30, respectively.

Under the scenario of $n = 6$ with DIDM setting in NARMA-10, the null

hypothesis that there is no difference between BWM and RWM in the *No-offset* case has $p < 0.001$, $A = 0.67$ (medium effect); in the offset by u case has $p < 0.001$, $A = 0.81$ (large effect); and in the offset by $2u$ case has $p < 0.001$, $A = 0.63$ (medium effect). In NARMA-30, with the scenario of $n = 7$ with DIDM setting, the null hypothesis that there is no difference between BWM and RWM in the *No-offset* case has $p < 0.001$, $A = 0.79$ (large effect); in the offset by u case has $p < 0.001$, $A = 0.91$ (large effect); and in the offset by $2u$ case has $p < 0.001$, $A = 0.63$ (Medium effect).

Table 5.1: NARMA-10 experimental results (DIDM column NRMSE Values) for Binary Weight Mask (BWM) and Random Weight Mask (RWM). Hypothesis testing results p-value and A-value are used for data analysis between BWM and RWM.

n	Scenario	BWM	RWM	p-value	A-value	Effect Size
6	No-offset	0.082	0.118	< 0.001	0.67	Medium
6	Offset by u	0.132	0.220	< 0.001	0.81	Large
6	Offset by $2u$	0.114	0.131	< 0.001	0.62	Medium
7	No-offset	0.129	0.162	< 0.001	0.79	Large
7	Offset by u	0.221	0.341	< 0.001	0.91	Large
7	Offset by $2u$	0.172	0.186	< 0.001	0.63	Medium

Table 5.2: NARMA-30 experimental results (DIDM column NRMSE Values) for Binary Weight Mask (BWM) and Random Weight Mask (RWM). Hypothesis testing results p-value and A-value are used for data analysis between BWM and RWM.

n	Scenario	BWM	RWM	p-value	A-value	Effect Size
6	No-offset	0.481	0.525	< 0.001	0.67	Medium
6	Offset by u	0.522	0.583	< 0.001	0.81	Large
6	Offset by $2u$	0.497	0.546	< 0.001	0.63	Medium
7	No-offset	0.524	0.553	< 0.001	0.79	Large
7	Offset by u	0.558	0.642	< 0.001	0.91	Large
7	Offset by $2u$	0.538	0.569	< 0.001	0.78	Large

Santa Fe Laser Task

The experimental results in Figure. 5.5 demonstrate the Binary Weight Mask displays better performance of all the masking types, among all three arrangements, with the lowest median NMSEs of 0.054, 0.082 and 0.056 ($n = 2$), 0.102, 0.175 and 0.109 ($n = 4$) respectively. The summary of the lowest median NMSE values provided in Tables 5.3.

The null hypothesis states that there is no distinction between BWM and RWM in the *No-offset* case. The statistical analysis yields a significance level of $p < 0.001$ for this hypothesis, indicating strong evidence to reject it. The effect size for this comparison is $A = 0.65$, which falls within the range of a medium effect. Similarly, in the case of an offset by u, the null hypothesis is also rejected with a significance level of $p < 0.001$. The effect size for this comparison is $A = 0.66$, which again represents a medium effect. Likewise, in the case of an offset by $2u$, the null hypothesis is rejected with a significance level of $p < 0.001$. The effect size for this comparison is $A = 0.67$, indicating a medium effect size.

Table 5.3: Santa Fe laser task experimental results (DIDM column NMSE Values) for Binary Weight Mask (BWM) and Random Weight Mask (RWM). Hypothesis testing results p-value and A-value are used for data analysis between BWM and RWM.

n	Scenario	BWM	RWM	p-value	A-value	Effect Size
2	No-offset	0.054	0.086	< 0.001	0.65	Medium
2	Offset by u	0.082	0.129	< 0.001	0.66	Medium
2	Offset by $2u$	0.056	0.097	< 0.001	0.67	Medium
4	No-offset	0.102	0.152	< 0.001	0.79	Large
4	Offset by u	0.175	0.201	< 0.001	0.91	Large
4	Offset by $2u$	0.109	0.187	< 0.001	0.78	Large

Summary. Overall, these results evaluated by effect size suggest that the Binary Weight Mask is statistically significantly better than the Random Weight Mask in all three arrangements (*No-offset*, offset by u and $2u$) in both NARMA and Santa Fe Laser benchmark tasks.

5.3.3 Offset *vs* No-offset masking

In this section, the results of comparing the performance of Offset and No-offset masks are presented.

NARMA

For the NARMA-10 and NARMA-30 tasks, Table 5.4 presents a summary of experimental outcomes across configurations: No-offset, Offset by u, and Offset by $2u$. These results, derived from masking types BWM and RWM, are consolidated from DIDM configuration, the right column of the central charts of Figures 5.3 and 5.4. It is evident that with all types of masking, No-Offset scenario obtain the best performance.

For instance, the NARMA-30 experimental results show that, with n = 7, Random Weight Mask with DIDM configuration (the right column of the right chars of Figures. 5.4b, 5.4d and 5.4f), the median of box plots in NRMSE values are 0.553, 0.642 and 0.569 for No-offset ($[-u, u]$), offset by u ($[0, 2u]$) and offset by $2u$ ($[u, 3u]$), respectively.

Santa Fe Laser

The experimental results for the Santa Fe Laser task is illustrated in Figure. 5.5 and summarised in Table. 5.5. For both n = 2 and 4, the results

Table 5.4: NARMA task experimental results (DIDM column NRMSE Values) for No-offset, offset by u and offset by $2u$.

n	Task	Masking Type	No-offset	Offset by U	Offset by $2u$
6	NARMA-10	BWM	0.082	0.132	0.114
6	NARMA-10	RWM	0.118	0.220	0.131
6	NARMA-30	BWM	0.481	0.522	0.497
6	NARMA-30	RWM	0.525	0.583	0.546
7	NARMA-10	BWM	0.129	0.221	0.172
7	NARMA-10	RWM	0.162	0.341	0.186
7	NARMA-30	BWM	0.524	0.558	0.538
7	NARMA-30	RWM	0.553	0.642	0.569

derived from the Random Weight Mask (displayed in the right column of all figures) confirm that the best performance is achieved with no-offset, while the poorest performance is observed when offset by u. However, for this specific task, when a Binary Weight Mask is applied, there is no significant difference between no-offset and offset by $2u$.

Table 5.5: Santa Fe laser task experimental results (DIDM column NMSE Values) for No-offset, offset by u and offset by $2u$.

n	Masking Type	No-offset	Offset by U	Offset by $2u$
2	BWM	0.054	0.082	0.056
2	RWM	0.086	0.129	0.097
4	BWM	0.102	0.175	0.109
4	RWM	0.152	0.201	0.187

Summary. Both no-offset (Eq. 5.1) and offset by $2u$ (Eq. 5.3) cases have better results than the offset by u mask function (Eq. 5.2). This is due to the input signal $I(t)$ in the range over the interval $[0, 2u]$, which is the case of BWM means approximately half the input values are masked to 0, resulting in information loss, compared to the no offset (interval $[-u, u]$) and offset by

$2u$ (interval $[u, 3u]$) cases, where no such zero values are formed.

5.3.4 Random masks *vs* random input

There are two sources of randomness: the mask, and the input stream. In the NARMA-10 Random Weight Mask / no-offset experiments (Figure. 5.3a and Figure. 5.3b, center), comparing SIDM and DISM has $p = 0.272$ and $p = 0.325$, respectively, so there is no statistically significant difference between the randomness from the mask and from the input.

5.4 Experiment B: Random Bias of *BWM* Sequence

Prior studies in the area of Delay-feedback Reservoir Computing consistently mention the creation of virtual nodes along the delay-line, often involving a term like "randomly generating a mask within the interval $[-u, u]$". However, there is insufficient exploration of how the "randomness" of this distribution might affect the performance of the Delay-feedback Reservoir Computing.

In this section, we describe an experiment to ascertain the influence of the *Binary Weight Mask* signal's distribution (**randomness**) on the performance of Delay-Feedback Reservoir Computing, under two conditions: *No-offset* and *Offset*. This experiment provides an empirical "recipe" illustrating the **randomness** distribution in the generation of the mask signal. The term 'Uppers' defines the number of high values in the BWM. In a standard random case, there are on average typically 32 Uppers out of a total of 64 virtual nodes. However, random effects might cause slight deviations from

this 50/50 split, particularly as we are dealing with small numbers. In this experiment, we systematically select (parameter sweep) different numbers of Uppers, ensuring the bias of randomness is fully explored.

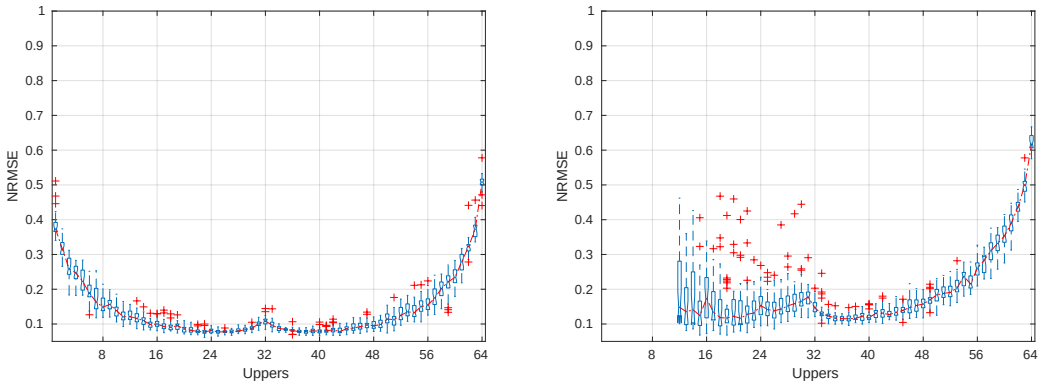
The experimental performances are presented by using the benchmark tasks NARMA-10 and NARMA-30. The parameters of Delay-Feedback Reservoir Computing based on Mackey–Glass system are defined in Table. 5.2.

5.4.1 No-Offset

The experimental results of NARMA-10 and NARMA-30 benchmark tasks under *No-Offset* (Eq. 5.1) scenario with nonlinearities of $n = 6$ and $n = 7$ are plotted in Figure. 5.6. The 'Uppers' noted in the figures refers to the number of positive u values in $[-u, u]$. In No-Offset scenario here, $u = 0.5$. The highest value for 'Uppers' is 64, as that is the number of virtual nodes we set along the delay-line in this study.

The experimental results for nonlinearity $n = 6$ in NARMA-10 and NARMA-30, depicted in Figures 5.6a and 5.6c respectively, both demonstrate a "symmetrical-like" trend centered around the midpoint of the x-axis, i.e., where the count of 'Uppers' is 32. The lowest NRMSE values are achieved at 'Uppers' = 27 (NRMSE = 0.077) and 'Uppers' = 37 (NRMSE = 0.078). When the system has all virtual nodes that generate from all masking values equal $-u$ or u in the system, i.e., 'Uppers' = 0 and 'Uppers' = 64 respectively, the system has the worst performance. However, the interquartile range (IQR) of the boxplots in Figures 5.6a and 5.6c show a relatively stable behaviour of the system with varying input sequences.

Figures 5.6b and 5.6d display the results of experiments carried out



(a) $n = 6$ – NARMA-10

(b) $n = 7$ – NARMA-10

(c) $n = 6$ – NARMA-30

(d) $n = 7$ – NARMA-30

Figure 5.6: Experimental results for investigating the 'Randomness' of distribution of Binary Weight Mask. No-offset $[-u, u]$ masking scenario is used. (a) and (b) show the NARMA-10 results for nonlineaities $n = 6$ and $n = 7$; (c) and (d) show the NARMA-30 results for nonlineaities $n = 6$ and $n = 7$ respectively. 'Uppers' stands for the number of positive u values in $[-u, u]$.

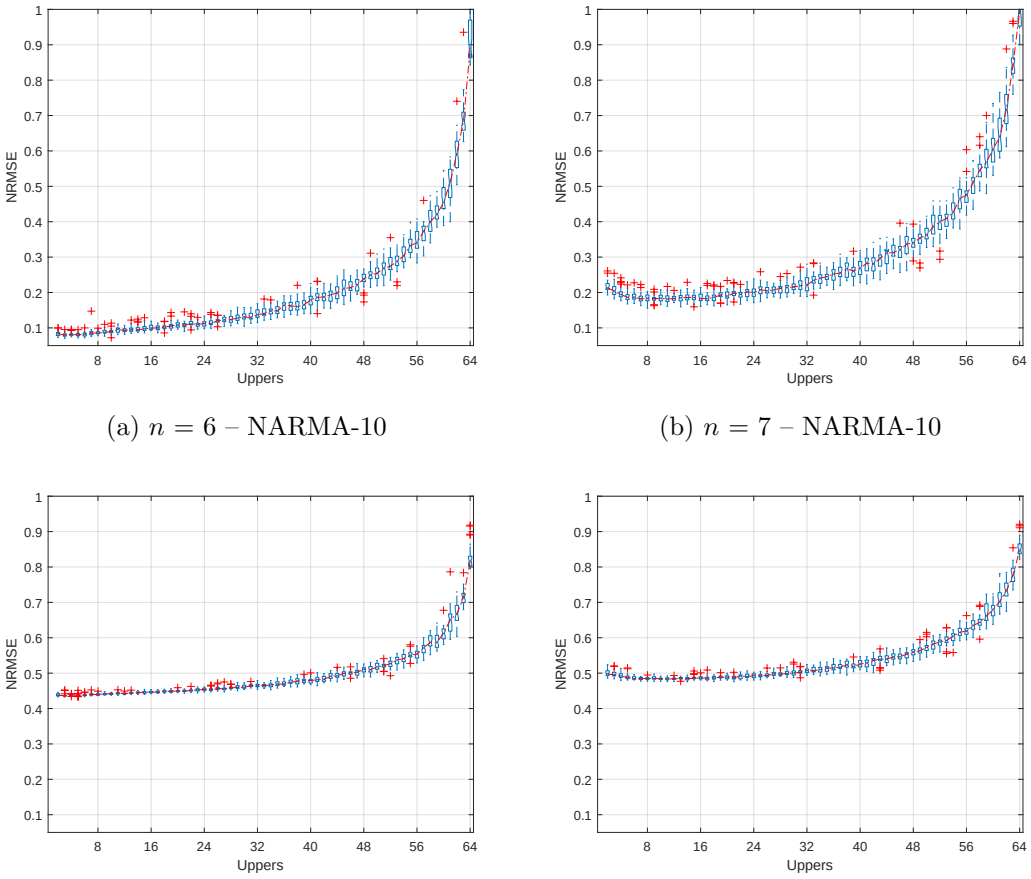
on NARMA-10 and NARMA-30 respectively, using a nonlinearity factor of $n = 7$. It is noteworthy that the first twelve 'Uppers' values on the x-axis (ranging from 'Uppers' = 1 to 'Uppers' = 11) correspond to NaN (Not a Number) values, which indicates that the Mackey-Glass DFRC system encountered errors/warnings while generating these test results. Furthermore, the interquartile range (IQR) of the boxplot results for 'Uppers' values from 12 to 31 appears considerably wider compared to the IQR for 'Uppers' values ranging from 32 to 64.

These results reveal that, careful consideration must be given when selecting the nonlinearity value in the Mackey-Glass model outlined in Eq. 3.6, whether it is an odd or even number under the No-Offset masking scenario. This is particularly crucial because the exponential term, which represents nonlinearity in the denominator, impacts the stability of the system. This effect is especially pronounced when the majority of mask values are negative.

5.4.2 Offset

Figure. 5.7 presents the experimental results of the NARMA-10 and NARMA-30 benchmark tasks, performed under the 'Offset' scenario (defined in Eq. 5.2), using nonlinearity parameters of $n = 6$ and $n = 7$. In this case, 'Uppers' as x-axis label in the figures, stands for the number of $2u$ values in $[0, 2u]$, i.e., $[0, 1]$ in this experiment as $u = 0.5$.

The figures indicate that the performance of the Mackey–Glass DFRC deteriorates as the system receives an increased number of 'Uppers' as input masking signals. However, it is observed that the points at which the trends begin to escalate exponentially vary across different figures. In Figures. 5.7a



(a) $n = 6$ – NARMA-10

(b) $n = 7$ – NARMA-10

(c) $n = 6$ – NARMA-30

(d) $n = 7$ – NARMA-30

Figure 5.7: Experimental results for investigating the 'Randomness' of distribution of Binary Weight Mask. Offset $[0, 2u]$ masking scenario is used. (a) and (b) show the NARMA-10 results for nonlineaities $n = 6$ and $n = 7$; (c) and (d) show the NARMA-30 results for nonlineaities $n = 6$ and $n = 7$ respectively.'Uppers' stands for the number of $2u$ values in $[0, 2u]$.

and 5.7b, the system's performance remains fairly stable until the mask contains more than 15 'Uppers' signals. Conversely, in Figures. 5.7c and 5.7d, the performance stability is maintained until the mask includes more than 40 'Uppers'.

The findings presented in Figure. 5.7 indicate that, under the Offset scenario ($[0, 2u]$), the system does not gain any advantages from increasing the amount of $2u$ values in the masking; however, it might gain by decreasing them from 32 $2u$ values.

5.5 Conclusion

In this chapter is undertaken a detailed exploration of the assumptions surrounding masking, an integral procedure within the framework of delay-feedback reservoir computing.

The examination of the masking procedure was approached from two distinct perspectives:

Firstly, an experimental comparison is made between Binary Weight Mask (BWM) and Random Weight Mask (RWM) using three benchmark tasks - NARMA-10, NARMA-30, and Santa Fe Laser task. The benchmarks use non-parametric statistical ranksum tests to calculate p-values and Vargha-Delaney's A to calculate the effect size, testing the masks' performance with different nonlinearity parameters, and offset and no-offset configurations.

The results consistently indicate BWM performs statistically significantly better than RWM across all scenarios. In terms of offset vs. no-offset masking, the results suggest that no-offset masking generally provides better per-

formance. Given the widespread use of these tasks in related research, we believe our results will facilitate a broader comprehension of how these different masking signals perform.

Secondly, the impact of the randomness of Binary Weight Mask (BWM) signal distribution on the performance of Delay-feedback Reservoir Computing was explored. The study assesses two conditions: No-offset and Offset, under which the BWM's randomness is investigated. Experiments are conducted using the NARMA-10 and NARMA-30 benchmark tasks, employing the Mackey-Glass model as the system's base. In the No-offset scenario, best performances occur at moderate "randomness", with worst results at the extremes. The Offset scenario, however, shows a decline in system performance with an increasing number of higher mask values. The section emphasizes the requirement to carefully select nonlinearity values according to different tasks, including controlling random bias when generating masks, particularly in the No-offset scenario.

6 DFRC with external Feedback

A distinctive feature of Delay-Feedback Reservoir Computing (DFRC) system is their single-input/single-output structure, which makes them efficient for physical implementation. However, this also presents a significant limitation to multi-input tasks, as how to multiplex the inputs in the already-time-multiplexed input stream is not obvious.

To investigate this, in this chapter, a novel task for DFRC is proposed, which intrinsically requires multiple inputs: the control of a forced Van der Pol oscillator system. Here, the trained reservoir serves as a controller, modulating the nonlinear dynamics of the Van der Pol system by constraining its trajectory to a circle. During this procedure, the DFRC is integrated into a closed system, where the Delay-Feedback RC operates with external feedback.

We introduce an enhanced input masking process to inject multiple inputs in the time domain, thereby expanding the application range of DFRCs. Two methodologies, 'Interleaved' and 'Sequential', are investigated for integrating multi-input signals into a delay-line reservoir, without necessitating a modification of its topology.

The chapter is structured as follows: Section 6.1 introduces the background of the Van der Pol oscillation system. Section 6.2 dives deeper into

our multi-input injection methodologies and provides a comprehensive explanation of the proposed task. The outcomes of the experimental application of the proposed task are explored in Section 6.3. Finally, Section 6.4 culminates with the conclusion of the chapter.

6.1 The Van der Pol Oscillator System

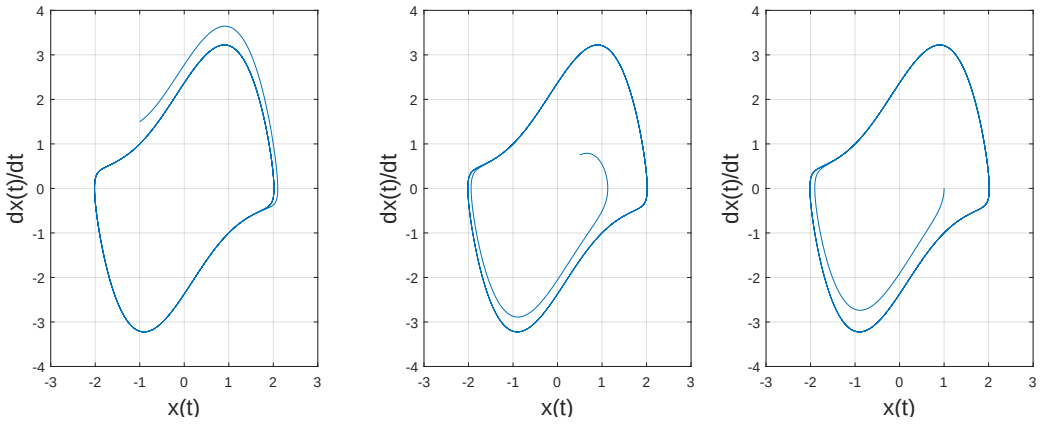
The Van der Pol oscillator was first described by the Dutch physicist Balthazar Van der Pol in the 1920s [175] and later employed to model the oscillations of the heart [176]. It is particularly useful for modeling systems that exhibit non-linear behavior, such as relaxation oscillations and self-sustained oscillations. Within the context of machine learning, the Van der Pol oscillator is commonly used to test the effectiveness of neural networks [177, 178]. In a recent study, Shougat et al. applied the Van der Pol oscillator as the substrate of physical reservoir computing [179]. Here, we use Delay-Feedback Reservoir Computing as the controller to constrain the trajectory of Van der Pol oscillator.

The Van der Pol oscillator is characterised by a nonlinear damping term. The forced Van der Pol, expressed as a nonlinear differential equation, is described as follow:

$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x = F(t) \tag{6.1}$$

where x is the state variable, t is time, and μ is a scalar parameter that dictates the nonlinearity and the strength of the damping. $F(t)$ is the external force applied to the VdP system.

When the external forcing term $F(t) = 0$, this is the Van der Pol oscillator (depicted in Figure. 6.1). The equation demonstrates oscillatory activity, but its amplitude is not constant; it represents an invariant set known as a 'limit cycle' attractor. Regardless of the initial conditions, all system trajectories converge to this attractor.



(a) x_0 = -1, $\dot{x_0}$ = 1.5 (b) x_0 = 0.5, $\dot{x_0}$ = 0.75 (c) x_0 = 1, $\dot{x_0}$ = 0

Figure 6.1: Phase plane of Van der Pol oscillator in Equation 6.1 with *F(t)* = 0 (no external force applied). Parameter settings: μ = 1.5, the initial condition of each plot is listed.

We seek to control the VdP system so that it produces a fixed-amplitude oscillation by defining a suitable control function for the external force $F(t)$ in Eq. 6.1. We use a PID (proportional integral derivative) controller configured to optimise the performance under a commanded trajectory to realize a controlled circular trajectory. The PID controller equation is given as:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{d}{dt} e(t) \tag{6.2}$$

where $u(t)$ is the controller output; K_p is the proportional gain; K_i is the integral gain; K_d is the derivative gain; $e(t)$ is the error signal, which is the difference between the desired set-point and the actual output. Cooper *et al.* highlights the challenges involved in regulating the nonlinear dynamics of the VdP Oscillator using a PID controller [180]. However, our aim in this work is not to find an optimal controller for the Van der Pol Oscillator but rather to illustrate the application of DFRC. And for this reason, we have

selected PID parameters that provide sufficiently good performance here.

6.2 Control of Van der Pol task

In this section, a novel multiple inputs task for Delay-Feedback Reservoir Computing is proposed: Control of a Forced Van der Pol Oscillator System. The task involves two steps:

1. *Imitation of PID controller behaviour.* The reservoir is trained to imitate the output control signal of the PID controller as closely as possible using the same inputs.

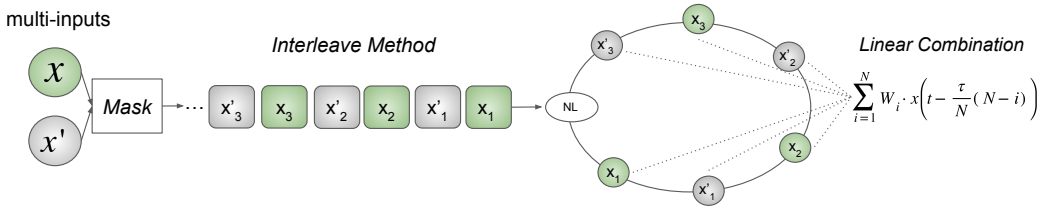
2. *Reservoir control Van der Pol Oscillator.* Replace the PID and drive the forced Van der Pol system with the trained reservoir as the controller.

The methodologies of injecting multiple inputs to DFRC and experimental setups used in this task are explained in the following sections.

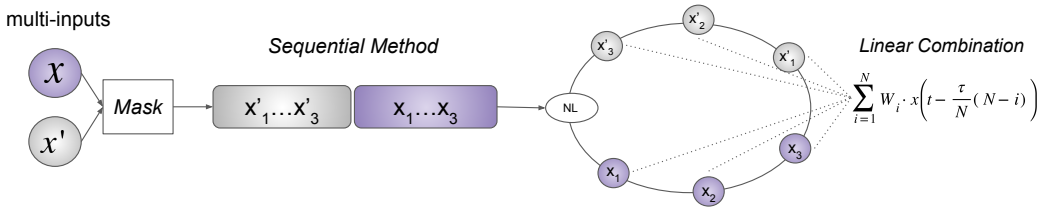
6.2.1 Multiple inputs

To effectively regulate the Van der Pol oscillator, the PID controller needs to know the oscillator's current state, including its position $x(t)$ and its velocity $\dot{x}(t)$.

The initial phase of the task involves training the reservoir to imitate the output control signal of the PID by injecting the same inputs of current position $x(t)$ and velocity $\dot{x}(t)$. However, there is inherent limitation when dealing with multi-input tasks with DFRCs due to their single input



(a)



(b)

Figure 6.2: (a) Interleave and (b) Sequential masking methods.

stream structure in the time domain. To address this, we introduce the masking approaches of injecting multiple inputs in the time domain of delay-line reservoir: **Interleave** and **Sequential**.

Interleave

The interleave technique schematic, shown in Figure. 6.2a, is a method that places masked dual- or multi-inputs in an alternating manner. For example, when two input streams undergo time-multiplexing procedure, they are completely mixed before injecting into the system, i.e. 'x_1,$\dot{x}_1$,x_2,$\dot{x}_2$,...' shown in the illustration with inputs being 'x and $\dot{x}$'.

Sequential

In contrast, the sequential approach leaves the inputs in sequence and concatenates them before undergoing the masking procedure (shown in Fig-

ure. 6.2b), i.e. 'x_1,x_2,x_3,..., $\dot{x_1}$,$\dot{x_2}$,$\dot{x_3}$,...'. The length of each input value's concatenation depends on the number of virtual nodes along the delay line. In this instance, each input value is concatenated to length of $N/2$, as N virtual nodes are defined in the reservoir.

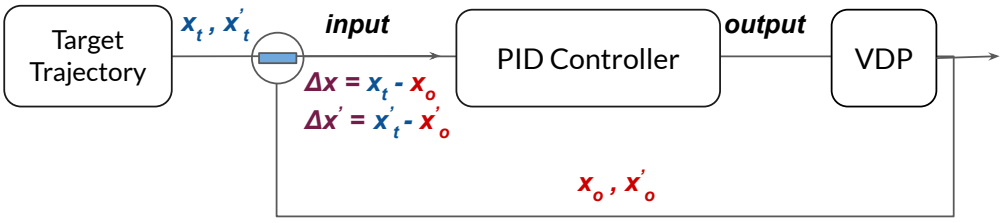
6.2.2 Training and Implementing

There are two phases involved in this task: a) *Imitation of PID controller* and b) *Using trained DFRC to drive the Van der Pol oscillator*. We refer to the initial phase as 'Training', as it involves training the DFRC to imitate the PID controller. The subsequent phase, 'Implementing', assesses the trained DFRC to determine if it has effectively gained control over the Van der Pol oscillator. The detail of each phase is given in the following sections.

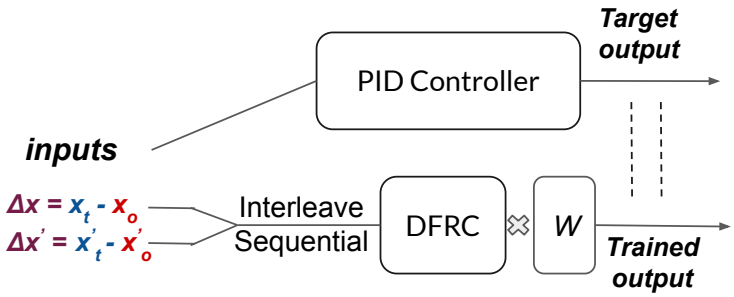
Training

Figure. 6.3 demonstrates the training procedure of the task – training DFRC to imitate the PID behaviour. The input sequences, which comprise of the error between the desired position and the current position (Δx), and the error between the desired velocity and current velocity ($\Delta \dot{x}$), along with the output control signals, are collected from the PID controller as depicted in Figure.6.3a. The Van der Pol oscillator is run to produce a sequence length of L, which is subsequently split into *training* and *testing* datasets, L_{train}, L_{test}, L_{unseen} and a washout of L_{washout} with the first L_{washout} samples discarded.

The input sequences (Δx and $\Delta \dot{x}$) in *train* dataset are combined using the proposed approaches: *Interleave* or *Sequential* to feed into the DFRC. The output control signal serves as the target output for DFRC. The output



(a) Van der Pol (VdP) oscillator driving by PID controller



(b) DFRC imitating PID controller behaviour

Figure 6.3: Training step of Forced Van der Pol task.

weight matrix W is trained using ridge regression (see Section 3.1.3 for detail on Determination of Weights). Normalised root mean square error (NRMSE) in Equation 3.10 is used to assess and compare the performance of different experimental techniques. Detail of NRMSE is given in Section 3.3.1.

Implementation

Figure. 6.4 depicts the implementing phase of Forced Van der Pol task by applying the trained DFRC as controller for Van der Pol oscillator. The L_{unseen} dataset from the training phase is interleaved/sequenced and used to evaluate DFRC as a controller. The weight matrix W used in this context are the sequences that have been optimized during the training phase.

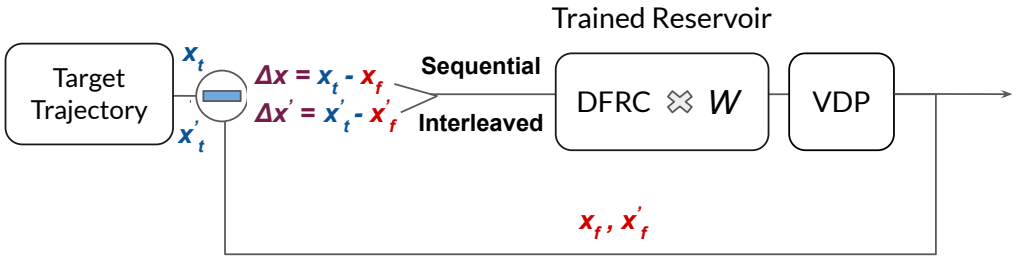


Figure 6.4: Testing phase of Forced Van der Pol ask. The DFRC with optimised weights W is used as controller to drive Van der Pol oscillator.

6.2.3 Experimental methods

Masking. Masking plays a crucial role in delay-feedback reservoir computing since it defines the virtual nodes along the delay line. In this task, we use *Binary Weight Mask* (a random sequence of -1 and $+1$ values) with no-offset and offset by u configurations in Equations 5.1 and 5.2 respectively.

Experimental environment. Van der Pol oscillator driven by PID controller is simulated in SIMULINK (Fig. 6.5), in which the nonlinear Van der Pol equation is propagated forward in time with different initial conditions. With the 'Feedback on/off Switch' set to 'OFF setting', the Van der Pol dynamic system exists on its own, without any external driving force, according to Eq. 6.1, and as shown in Fig. 6.6a. The single PID controller initial settings are: $k_{p0} = 1$; $k_{d0} = 50k_{p0}$; $k_{i0} = k_{p0} * 0.1$, which is optimized for an amplitude of 2. To compare the forced trajectory with this target trajectory, the phase plots are shown in Fig. 6.6. Note that here, the reservoir is trained using a PID controller tuned for a radius of 2 in all three cases. Since we are not seeking perfection but rather intend to demonstrate and compare the sensitivity of the PID and reservoir controller, optimised for radius 2

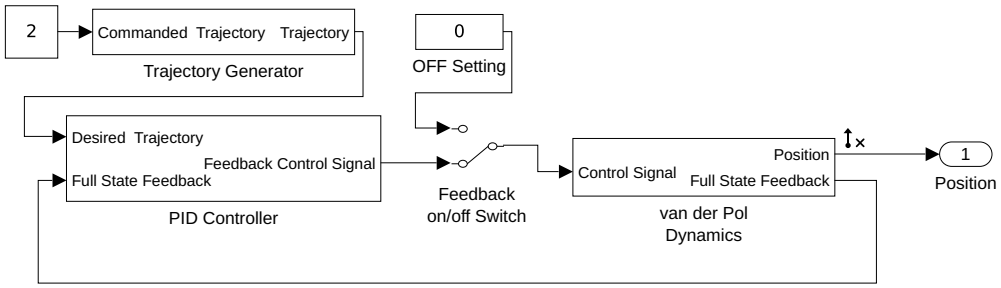


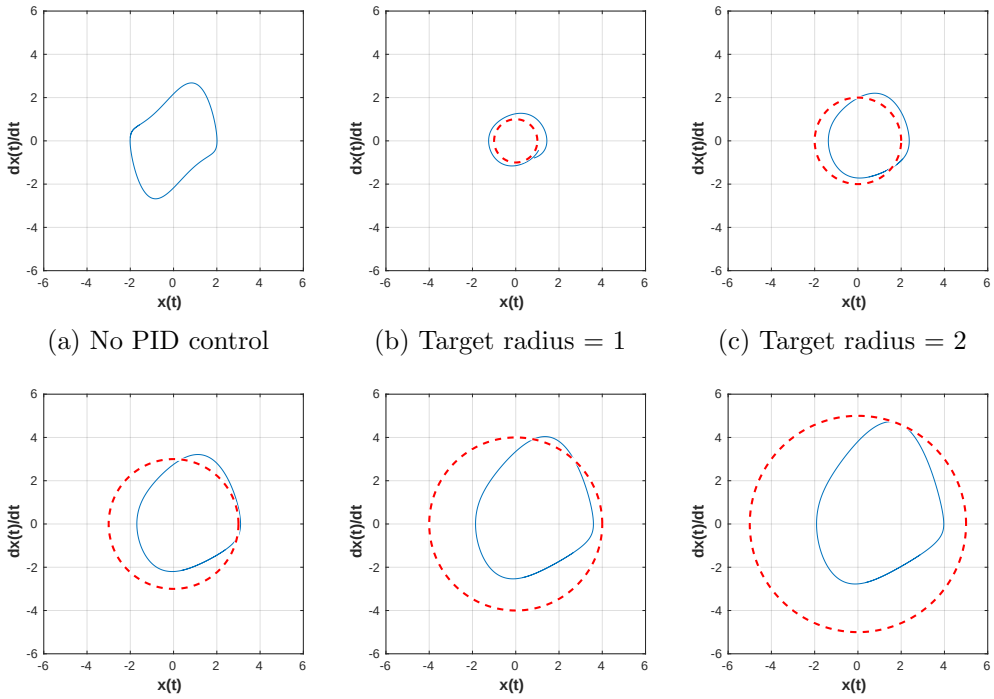
Figure 6.5: The block diagram of the Van der Pol Oscillator driven by PID controller. The single PID controller initial settings are: $k_{p0} = 1$; $k_{d0} = 50k_{p0}$; $k_{i0} = 0.1k_{p0}$.

respectively, when applied to larger or smaller radii.

The Mackey-Glass model depicted in Figure. 3.4 is used as DFRC in this experiment. Detail of the framework and implementation is presented in Section 3.2.1.

6.2.4 Parameter Setting

The experimental parameter settings of Mackey-Glass based DFRC are shown in Table 6.7. The Van der Pol oscillator is run to produce a sequence length of $L = 5000$, which is subsequently split into *training* and *testing* datasets, $L_{\text{train}} = 2280$, $L_{\text{test}} = 1140$, $L_{\text{unseen}} = 380$, and a washout of $L_{\text{washout}} = 1200$ with the first L_{washout} samples discarded.



(a) No PID control (b) Target radius = 1 (c) Target radius = 2

(d) Target radius = 3 (e) Target radius = 4 (f) Target radius = 5

Figure 6.6: Forced VdP oscillator trajectories (blue solid line) and forced target trajectories (red dashed line) are shown. (a) Initial baseline trajectory without PID controller (unforced Van der Pol trajectory). (b–f) Comparisons between desired forced trajectory and actual trajectory with amplitudes of 1–5 respectively. The single PID controller is optimized for an amplitude of 2 ($k_{p0} = 1$; $k_{d0} = 50k_{p0}$; $k_{io} = 0.1k_{p0}$).

parameter	value	description
β	2	coupling factor
γ	1	decay rate
τ	1.8 (sec)	delay in feedback loop
n	$1, 2, \ldots, 9$	nonlinearity
δ	0.25	input weighting
N	30	number of virtual nodes

Figure 6.7: Parameter values for the Mackey–Glass delay-line reservoir experiment. The Mackey–Glass system parameter values are from [4]; n is required to be an integer.

6.3 Experimental Results

6.3.1 Imitation of PID Control Behaviour

We systematically compare the computational performance of two multi-input techniques: 'Interleave' and 'Sequential', with no-offset (Eq. 5.1) and offset (Eq. 5.2) under different nonlinearity parameters: $n = 1$ to $n = 9$. In this experiment, the PID controller is optimized for a radius of 2 and these settings stay the same for all target radii used here, ranging from 1 to 5. Note that, therefore, the target sequences generated by the PID controller for radii of $1, 3, 4, 5$ are not optimal control solutions. This is deliberate, so that the capability to generalise over a range of radii $\neq 2$ can be evaluated independently for PID and reservoir controller.

Fig. 6.8 shows the results of these different experimental settings.

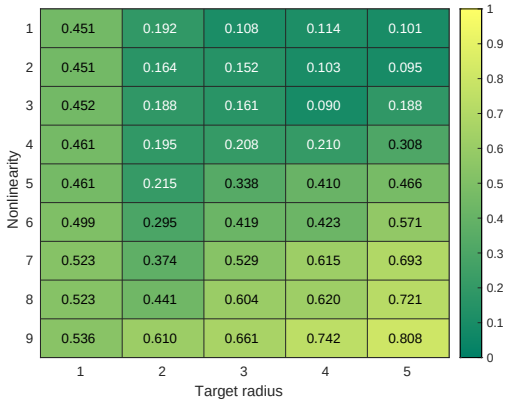
1. *Interleave v Sequential.* The Sequential technique displays better performance than Interleave, in the no-offset and offset arrangements, with lowest NRMSE of 0.090 and 0.069 ($n = 3$, $targetradius = 4$).

2. *No-offset v Offset.* In this task, offset mask gives better results. Since we are using Mackey-Glass equation (Eq. 3.6) as the non-linear component of the reservoir, the negative input value may cause the denominator term crushed with worse NRMSE results = 1.000 (indicated yellow in Fig. 6.8a and Fig. 6.8c).

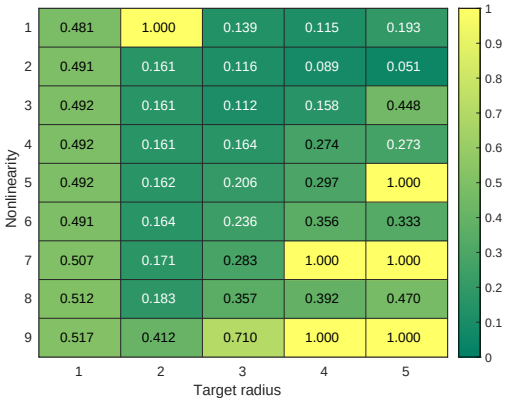
6.3.2 Trained DFRC controls Van der Pol Oscillator

Since the ultimate goal of this benchmark task is to use a reservoir as a controller to drive the trajectory of a Van der Pol system (see Figure 6.3b), we

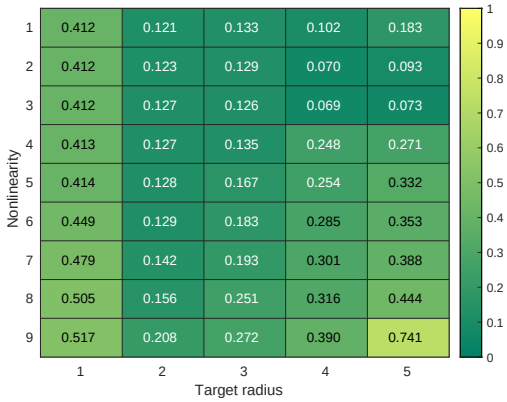
(a) Interleave with no-offset masking.



(b) Interleave with offset masking.

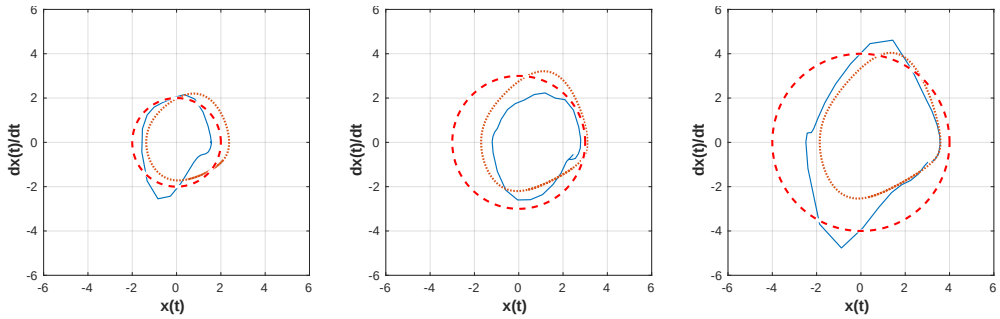


(c) Sequential with no-offset masking.



(d) Sequential with offset masking.

Figure 6.8: Simulation results in NRMSE for controlling benchmark task with two multi-input methodologies: *Interleave* (fig.6.8a and fig.6.8b) and *Sequential* (fig.6.8c and fig.6.8d) based on different masking schemes (Eq. 5.2 and Eq. 5.1).



(a) Target radius = 2. (b) Target radius = 3. (c) Target radius = 4.

Figure 6.9: Simulation results for Van der Pol nonlinear system driven by the trained reservoir. Comparisons between target circle (red dashed line), trajectory commanded by PID controller (orange dotted line), and trajectory controlled by reservoir computing (blue solid line).

use the weight matrix of best trained reservoir from the previous experiment, with $n = 2$, to drive the Van der Pol Oscillator.

Fig. 6.9 shows the comparison between the target circular trajectory (dashed line) representing the baseline, the trajectory commanded by the PID controller optimised for radius 2 (dotted line), and the trajectory controlled by the reservoir trained on the PID output from radius 2 (solid line).

6.4 Conclusion

In this chapter, a novel task for DFRC is proposed, that inherently necessitates multiple inputs: the control of a forced Van der Pol oscillator system. This task integrates the DFRC, which is trained on multiple-inputs (the oscillator's position and velocity), into a closed-loop system, i.e., with external feedback serves as a controller to modulate the Van der Pol Oscillator's nonlinear dynamics, constraining its trajectory to a circular path.

To facilitate multiple inputs, two masking approaches: 'Interleave' and

'Sequential' are investigated. Experimental results from comparing the two multi-input techniques showed that the 'Sequential' technique obtains better performance in both no-offset and offset arrangements, with the offset mask providing the best results.

The proposed task, divided into two stages – (i) DFRC imitating the PID controller, which showcases the adaptability of the DFRC in handling multiple inputs with proposed approaches; and (ii) implementing the trained DFRC as the controller – demonstrate the feasibility of DFRC controlling Van der Pol oscillators.

7 Conclusion and Future Work

7.1 Conclusion

This thesis begins with a simple question: is it possible to improve the computational performance of physical Delay-Feedback Reservoir Computing (DFRC) system by exploring its implicit assumptions? To answer this, we first list three implicit assumptions currently made about physical DFRC system:

1. The performance of DFRC is not affected by the attenuation that happens in physical delay-line since the nodes defined along it are 'virtual'

2. To define virtual nodes along the delay-line in DFRC, the mask used should comprise randomly generated numbers uniformly distributed between $[-u, u]$

3. DFRC is typically used to solve time series prediction tasks involving a single input and output with no external feedback

To investigate these assumptions, a simulation framework of DFRC based on a delay differential equation – the Mackey–Glass mathematical model – has been developed. The experimental results and discussions of assumptions 1,

2 and 3 are given in Chapter 4, 5 and 6 respectively, which are summarised in the following section.

7.1.1 Thesis Chapter Summary

This section contains overviews of each experimental chapter along with the technical details, which provides a reference to significant findings and outcomes presented in this thesis.

Regarding to the implicit assumption 1 on the read-out phase of physical DFRC, the experiments in **Chapter 4** show that the computation performance of DFRC is affected by the attenuation along the delay-line. To mitigate this impact, we propose a methodology referred to as 'Devirtualisation' which trades-off the DFRC system's read-out frequency and the number of output lines.

- Simulation environments: Mackey–Glass system with: a) ideal delay-line; b) damped delay-line
- Four configurations of delay-line: A: N_{total}=32, θ=25ms; B: N_{total}=32, θ=50ms; C: N_{total}=64, θ=25ms; D: N_{total} =64, θ=50ms.
- Benchmark tasks: NARMA-10, NARMA-30 and Santa Fe laser task.
- Concept of Devirtualisation: the procedure of directly tapping into the delay line at the position of a 'virtual' node, rather than at the delay line's end.
- The initial experiment uses an ideal delay-line to test the proper functioning of each block and demonstrate the feasibility of 'Devirtualisation' without the influence of other factors.

- The second experiment compares the ideal delay-line and damping delay-line with fully virtualised systems (only one output line is used), the result demonstrates the fact that the computation performance of DFRC is affected by the attenuation along the delay-line.

- In the third experiment, the fully damped delay-line is investigated in two distinct scenarios: 'constant damping rate over delay-line' and 'constant damping rate per unit length.' The results under both scenario show that the devirtualisation approach significantly enhances physical DFRC performance, consistently improving with more output lines and reaching peak effectiveness when equalling the virtual node count.

In **Chapter 5**, the assumptions about two perspectives of masking procedure in DFRC are explored: Comparison of Binary Weight Mask (BWM) vs. Random Weight Mask (RWM); The bias in randomness of Binary Weight Mask (BWM) signal distribution's impact due to small numbers/sequences of masks.

- Simulation environment: Mackey–Glass model with ideal delay.

- Benchmark tasks: NARMA-10, NARMA-30 and Santa Fe laser task.

- Assessed masks' performance:

  - With odd/even nonlinearity parameters.

  - Under different scenarios: No-offset ($[-u, u]$), Offset by u ($[0, 2u]$) and Offset by 2u ($[u, 3u]$) configurations.

- Experiment 1 – Comparison of Binary Weight Mask (BWM) vs. Random Weight Mask (RWM):
  - BWM is statistically superior to RWM across all scenarios.
  - No-offset masking generally yields better results.
- Experiment 2 – Randomness of Binary Weight Mask (BWM) signal distribution's impact:
  - Findings in No-offset: 1) Best performances are seen at moderate quantity of bias in randomness; poor results at extreme bias. 2) Importance of careful nonlinearity values selection, especially in No-offset scenario.
  - Findings in Offset: decline in system performance with an increasing number of high mask values.

In **Chapter 6**, a novel task for DFRC that inherently necessitates multiple inputs: the control of a forced Van der Pol oscillator system is proposed.

- Simulation environment: Mackey–Glass model with ideal delay-line.
- Two masking approaches to facilitate multiple inputs: *Interleave* and *Sequential*.
- Experimental finding: 'Sequential' outperforms 'Interleave' under the scenarios of No-offset and Offset by u.
- The control of a Forced Van der Pol Oscillator task: employ the trained DFRC as a controller, modulating the nonlinear dynamics of the Van der Pol oscillating system by constraining its trajectory to a circle.

- Two stages of proposed task:
  - DFRC imitates the PID controller: demonstrates DFRC's adaptability in handling multiple inputs using the proposed approaches.
  - Implementing the trained DFRC as the actual controller: validates the application of DFRC in controlling the Van der Pol oscillator.

7.2 Future Work

While the outcomes of this thesis break the implicit assumptions of physical DFRC and thereby enhance its performance, they also raise new open research questions that remain to be explored. Future work associated with specific parts of this thesis are as follow:

7.2.1 Topology of DFRC

As addressed in Chapter 6, a distinct characteristic of Delay-Feedback Reservoir Computing (DFRC) is its single-input configuration, which renders it efficient for physical deployments. Yet, this feature poses a notable constraint for tasks with multiple inputs, given that the order of information in the time-multiplexed input stream is not readily apparent.

In this thesis we propose masking procedures (*Interleave* and *Sequential*) to address this limitation, enabling multiple inputs to be channeled into a single nonlinear node. Here, the inherent topology of DFRC remains unchanged. Investigating the DFRC topology that incorporates multiple delay lines to improve its versatility for multi-input tasks, particularly those demanding varied input sample rates, could be advantageous, possibly expanding its

range of application.

7.2.2 Hyper-parameter Optimisation Framework

In Chapter 3, Section 3.2.2, we conducted a series of experiments to determine reasonable values for (we believe are) crucial parameters within the dynamical system being used, aiming to establish a relatively suitable regime for DFRC. When designing a physical DFRC, optimising each system parameter can be time-consuming. This is because the performance of DFRC is highly dependent on the choice of hyper-parameters, such as delay time and feedback strength, as well as other parameters that might not be immediately recognized as significant. Extensive tuning might be necessary to optimise the system for a specific problem. Therefore, for future advancements, developing a comprehensive framework to optimize hyperparameters of the dynamical system based on specific tasks could be highly beneficial for the design of physical DFRC.

7.2.3 Practical Real-World Applications

DFRC is primarily designed for time series prediction tasks. Previous work [181] uses DFRC successfully detect the VEB (Ventricular Ectopic Beat) in the continuous ECG (electrocardiogram) stream after fully optimizing the parameters.

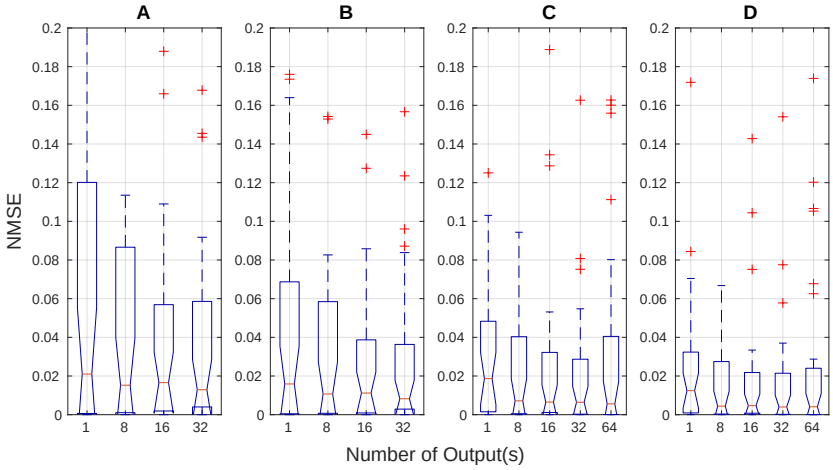
For future work, another medical application for physical DFRC is detecting and intervening of Parkinson's disease. Parkinson's disease detection often involves analysing biomedical data, such as voice recordings, gait analysis, or even brain signals. These biomedical data present time-series patterns

or temporal dependencies that are indicative of the disease's progression or symptoms, then DFRC, with its temporal processing capability, could be a potential tool for this purpose. In this thesis, we demonstrate the capability of DFRC to function as a controller. While DFRC cannot directly provide a cure, it might aid in optimising therapeutic interventions. For example, Deep Brain Stimulation (DBS) is a procedure sometimes used for Parkinson's. DFRC could potentially be used to optimise the parameters of DBS based on real-time feedback from the patient and alleviate the symptoms of Parkinson's disease to improve the quality of life for patients.

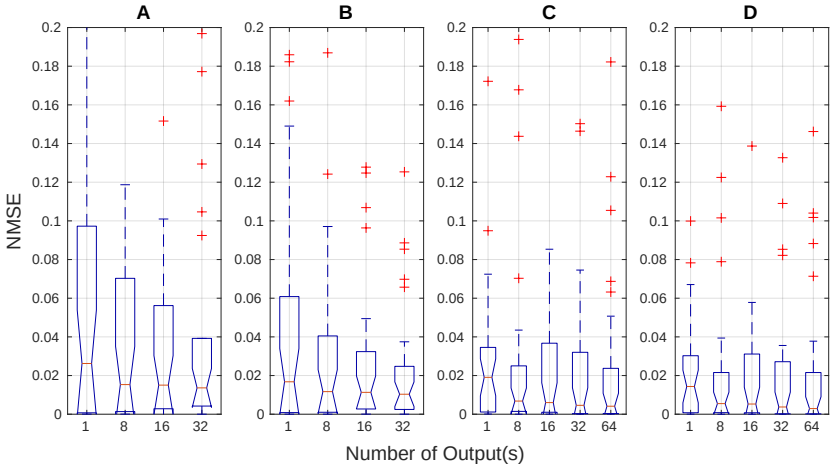
Appendices

In this appendix, the experiment results of Santa Fe laser task in Chapter 4 and Chapter 5 are given.

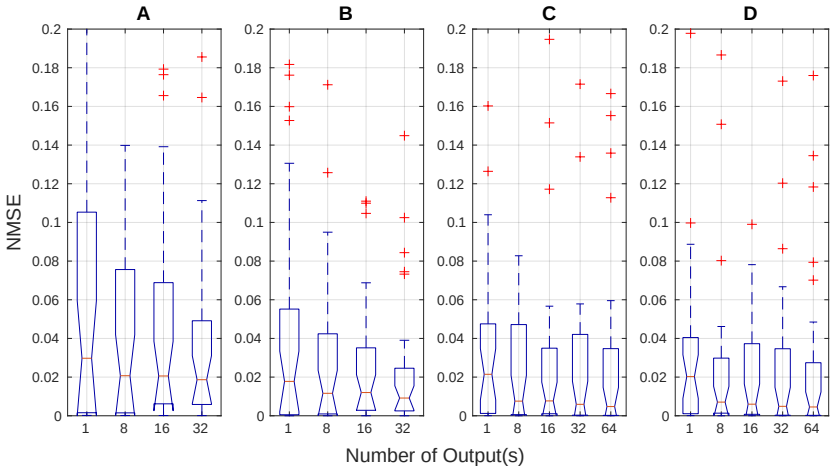
This experiment explores the performance of the devirtualisation approach in various situations, we assess the effectiveness of the four delay-lines in two distinct scenarios: *a*) Constant damping rate over delay-line and *b*) Constant damping rate per 'length'. This experiment uses four damping delay-lines that vary either in the number of virtual nodes or in the length of the line.



(a) Low damping 10%

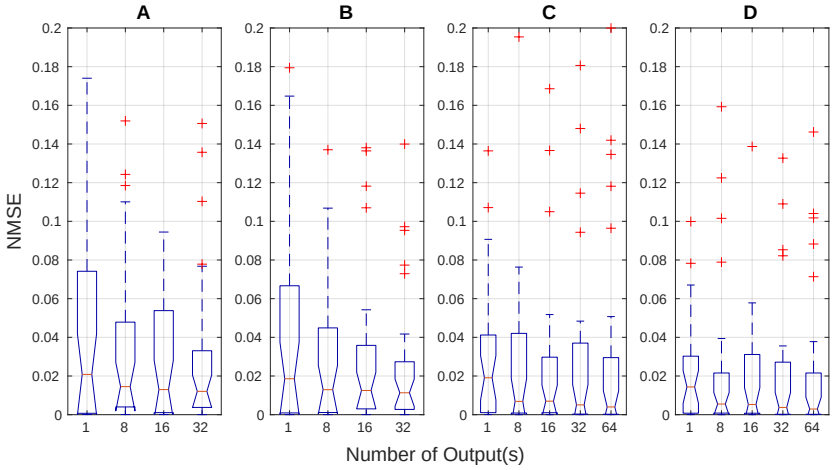


(b) Mid damping 50%

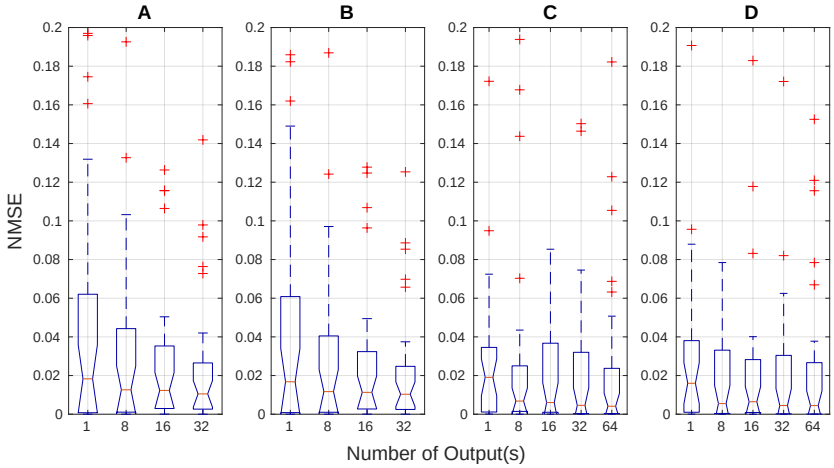


(c) High damping 90%

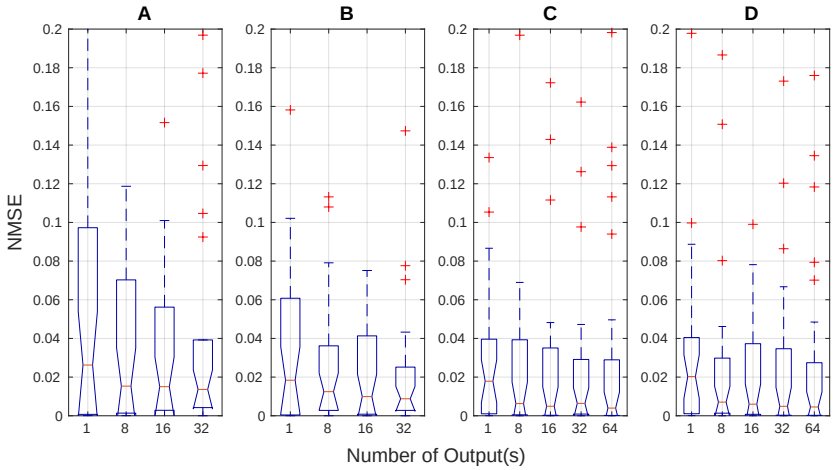
Figure 1: Santa Fe laser task results for constant damping rate over delay-line. A: N_{total}=32, θ=25ms; B: N_{total}=32, θ=50ms; C: N_{total}=64, θ=25ms; D: N_{total} =64, θ=50ms.



(a) Low damping 1% per length



(b) Mid damping 2% per length



(c) High damping 4% per length

Figure 2: Santa Fe laser task results for constant damping rate per length. A: N_{total}=32, θ=25ms; B: N_{total}=32, θ=50ms; C: N_{total}=64, θ=25ms; D: N_{total} =64, θ=50ms.

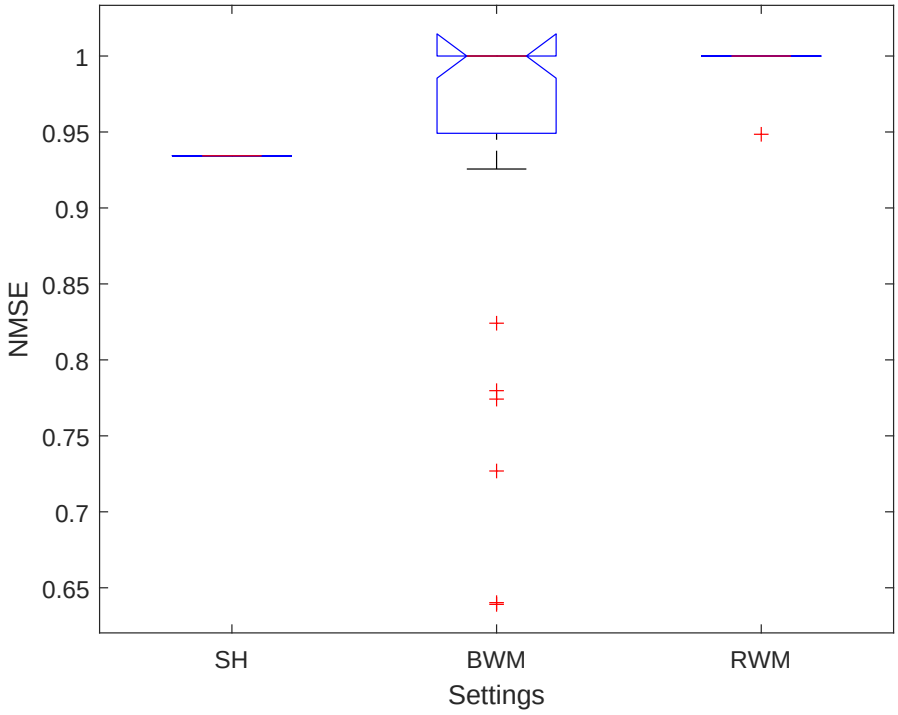


Figure 3: Santa Fe laser task results in NMSE values for $n = 3$ under No-Offset scenario. 'SH': Sample and Hold; 'BWM': Binary Weight Mask; 'RWM': Random Weight Mask.

This plot demonstrates the finding of Santa Fe laser task under nonlinearity of Mackey-Glass system equals to 3 under the configuration of No-Offset masking type. The parameters of experimental settings is listed in Table. 5.2.

Bibliography

[1] "Moore's law — Wikipedia, the free encyclopedia," [Online; accessed 11-April-2023]. [Online]. Available: http://en.wikipedia.org/wiki/Moore%27s_law

[2] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of the Atmospheric Sciences*, vol. 20, no. 2, pp. 130–141, 1963.

[3] O. E. Rössler, "An equation for continuous chaos," *Physics Letters A*, vol. 57, no. 5, pp. 397–398, 1976.

[4] L. Glass and M. Mackey, "Mackey-Glass equation," *Scholarpedia*, vol. 5, no. 3, p. 6908, 2010, revision #186443; doi:10.4249/scholarpedia.6908.

[5] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[6] R. R. Schaller, "Moore's law: past, present and future," *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[7] D. C. Brock and G. E. Moore, *Understanding Moore's law: four decades of innovation*. Chemical Heritage Foundation, 2006.

[8] P. Gelsinger, "Moore's law–the genius lives on," *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 18–20, 2006.

[9] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur, and H.-S. P. Wong, "Device scaling limits of si mosfets and their application dependencies," *Proceedings of the IEEE*, vol. 89, no. 3, pp. 259–288, 2001.

[10] L. Xiu, "Time moore: Exploiting moore's law from the perspective of time," *IEEE Solid-State Circuits Magazine*, vol. 11, no. 1, pp. 39–55, 2019.

[11] W. Olin-Ammentorp and N. Cady, "Biologically-inspired neuromorphic computing," *Science Progress*, vol. 102, no. 3, pp. 261–276, 2019.

[12] T. Chouard, "Legacy of a universal mind," *Nature*, vol. 482, no. 7386, pp. 455–455, 2012.

[13] H. Jaeger, "Towards a generalized theory comprising digital, neuromorphic and unconventional computing," *Neuromorphic Computing and Engineering*, vol. 1, no. 1, p. 012002, 2021.

[14] S. Lloyd, *Unconventional Quantum Computing Devices*. Springer, Singapore, 1998.

[15] A. Cifarelli, A. Dimonte, T. Berzina, and V. Erokhin, "On the loading of slime mold physarum polycephalum with microparticles for unconventional computing application," *BioNanoScience*, vol. 4, pp. 92–96, 2014.

[16] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[17] T. M. Mitchell, *Machine learning.* McGraw-Hill, 1997.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT Press, 2016.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 1097–1105.

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[21] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, *Recommender systems handbook.* Springer, 2011.

[22] R. J. Bolton and D. J. Hand, "Statistical fraud detection: A review," *Statistical Science*, vol. 17, no. 3, pp. 235–249, 2002.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[24] V. N. Vapnik, *The nature of statistical learning theory.* Springer, 1995.

[25] P. B. Jensen and P. Cohen, "Multiple comparisons in induction algorithms," *Machine Learning*, vol. 38, no. 3, pp. 309–338, 2000.

[26] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[27] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[28] T. Calders and S. Verwer, "Three naive bayes approaches for discrimination-free classification," *Data Mining and Knowledge Discovery*, vol. 21, no. 2, pp. 277–292, 2010.

[29] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[30] S. Barocas and A. D. Selbst, "Big data's disparate impact," *California Law Review*, vol. 104, p. 671, 2016.

[31] A. Ng, "The state of artificial intelligence," *Harvard Business Review*, vol. 95, no. 6, pp. 2–5, 2017.

[32] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*. Springer Series in Statistics, 2001.

[33] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media, 2009.

[34] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 20, no. 2, pp. 215–242, 1958.

[35] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[36] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[37] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[38] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[39] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[40] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, no. 3. Makuhari, 2010, pp. 1045–1048.

[41] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[42] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*. CRC Press, 2012.

[43] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[44] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[45] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," *Proceedings of the 20th international conference on machine learning (ICML-03)*, pp. 856–863, 2003.

[46] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[47] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. CRC press, 1984.

[48] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.

[49] G. Hinton and T. J. Sejnowski, *Unsupervised learning: foundations of neural computation*. MIT press, 1999.

[50] J. MacQueen, "Classification and analysis of multivariate observations," in *5th Berkeley Symp. Math. Statist. Probability*. University of California Los Angeles LA USA, 1967, pp. 281–297.

[51] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[52] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[53] K. Pearson, "On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[54] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. 11, pp. 2579–2605, 2008.

[55] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[56] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[57] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[58] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 285–295.

[59] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their composition-

ality," *Advances in neural information processing systems*, vol. 26, pp. 3111–3119, 2013.

[60] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[61] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Foundations and Trends in Information Retrieval*, vol. 2, no. 1–2, pp. 1–135, 2008.

[62] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of machine learning research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[63] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," in *Proceedings of the VLDB Endowment*, vol. 5, no. 7, 2012, pp. 622–633.

[64] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

[65] OpenAI, "Introducing gpt-4: An advanced language model by openai," 2021.

[66] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, ..., and F. Herrera, "Explainable artificial intelligence

(xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information Fusion*, vol. 58, pp. 82–115, 2020.

[67] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[68] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[69] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[70] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[71] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[72] J. T. Connor, R. D. Martin, and L. E. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 240–254, 1994.

[73] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[74] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn

encoder-decoder for statistical machine translation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014.

[75] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

[76] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[77] D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification," *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1422–1432, 2015.

[78] W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PLoS ONE*, vol. 12, no. 7, p. e0180944, 2017.

[79] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, and W.-k. Wong, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," *Advances in Neural Information Processing Systems*, pp. 802–810, 2015.

[80] Y. Yu, Z. Liu, and J. Sun, "Spatio-temporal LSTM for traffic prediction using Cnn to learn spatial dependencies," *2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 376–381, 2017.

[81] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," *International Conference on Machine Learning*, pp. 1310–1318, 2013.

[82] J. Tang, F. Yuan, X. Shen, Z. Wang, M. Rao, Y. He, Y. Sun, X. Li, W. Zhang, Y. Li *et al.*, "Bridging biological and artificial neural networks with emerging neuromorphic devices: fundamentals, progress, and challenges," *Advanced Materials*, vol. 31, no. 49, p. 1902761, 2019.

[83] W. Zhang, B. Gao, J. Tang, P. Yao, S. Yu, M.-F. Chang, H.-J. Yoo, H. Qian, and H. Wu, "Neuro-inspired computing chips," *Nature electronics*, vol. 3, no. 7, pp. 371–382, 2020.

[84] J. Grollier, D. Querlioz, and M. D. Stiles, "Spintronic nanodevices for bioinspired computing," *Proceedings of the IEEE*, vol. 104, no. 10, pp. 2024–2039, 2016.

[85] H. 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*, vol. 148, no. 34, p. 13, 2001.

[86] ——, "Short term memory in echo state networks. gmd-report 152," *GMD-German National Research Institute for Computer Science (2002)*, 2002.

[87] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based

on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[88] D. Verstraeten, B. Schrauwen, M. d'Haene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Networks*, vol. 20, no. 3, pp. 391–403, 2007.

[89] M. Lukovsevivcius and H. Jaeger, "A practical guide to applying echo state networks," *Neural networks: Tricks of the trade*, vol. 7700, pp. 659–686, 2009.

[90] M. Dale, J. F. Miller, S. Stepney, and M. A. Trefzer, "Evolving carbon nanotube reservoir computers," in *International Conference on Unconventional Computation and Natural Computation*. Springer, 2016, pp. 49–61.

[91] Z. Konkoli, S. Nichele, M. Dale, and S. Stepney, "Reservoir computing with computational matter," in *Computational Matter*, S. Stepney, S. Rasmussen, and M. Amos, Eds. Springer, 2018, pp. 269–293.

[92] L. Appeltant, M. C. Soriano, G. V. Der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer, "Information processing using a single dynamical node as complex system," *Nature Communications*, vol. 2, no. 1, pp. 468–468, 2011.

[93] F. Palumbo, C. Gallicchio, R. Pucci, and A. Micheli, "Human activity recognition using multisensor data fusion based on reservoir computing," *Journal of Ambient Intelligence and Smart Environments*, vol. 8, no. 2, pp. 87–107, 2016.

[94] W. Wang, X. Liang, M. Assaad, and H. Heidari, "Wearable wristworn gesture recognition using echo state network," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2019, pp. 875–878.

[95] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, "Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach," *Physical Review Letters*, vol. 120, no. 2, p. 024102, 2018.

[96] P. Antoniou and V. Tzagarakis, "Deep echo state networks for mobile-robot control," *Robotics and Autonomous Systems*, vol. 104, pp. 1–13, 2018.

[97] A. Rodan and P. Tino, "Minimum complexity echo state network," *IEEE transactions on neural networks*, vol. 22, no. 1, pp. 131–144, 2011.

[98] S. Stepney, "Non-instantaneous information transfer in physical reservoir computing," in *Unconventional Computation and Natural Computation: 19th International Conference, UCNC 2021, Espoo, Finland, October 18–22, 2021, Proceedings 19*. Springer, 2021, pp. 164–176.

[99] H. Manjunath and R. Narayanan, "Echo state networks for modelling and control of morphing aircraft," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 227, no. 1, pp. 160–171, 2013.

[100] S. Zhang, Y. Zhao, and L. Zhang, "Optimized echo state networks for time series prediction," *2018 37th Chinese Control Conference (CCC)*, pp. 6315–6320, 2018.

[101] C. Gallicchio, A. Micheli, and L. Pedrelli, "Deep reservoir computing: A critical experimental analysis," *Neurocomputing*, vol. 268, pp. 87–99, 2017.

[102] H. Jaeger, "Discovering multiscale dynamical features with hierarchical echo state networks," *Technical Report of Jacobs University Bremen*, 2007.

[103] Z. Wang, S. Joshi, S. E. Savel'ev, H. Jiang, R. Midya, P. Lin, M. Hu, N. Ge, J. P. Strachan, Z. Li *et al.*, "Reservoir computing with dynamic memristors for pattern recognition," *Nature Communications*, vol. 9, no. 1, pp. 1–11, 2018.

[104] F. Xue and J. C. Príncipe, "Echo state networks: Stability, robustness, and generalization," *Neural Networks*, vol. 140, pp. 88–95, 2021.

[105] I. Boybat, M. Le Gallo, S. R. Nandakumar, T. Moraitis, T. Parnell, T. Tuma, B. Rajendran, Y. Leblebici, and A. Sebastian, "Neuromorphic computing with multi-memristive synapses," *Nature Communications*, vol. 9, no. 1, pp. 1–12, 2018.

[106] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[107] J. Zhao, H. Xu, D. Xu, and S. Tan, "A comprehensive review of echo state network and its variants," *Neural Networks*, vol. 131, pp. 91–101, 2020.

[108] T. Zhang, Y. Yang, and M. Chatterjee, "Multichannel weighted speech prediction for intelligibility improvement in cochlear implant coding," *The Journal of the Acoustical Society of America*, vol. 146, no. 5, pp. 3639–3649, 2019.

[109] J. Stelzer and H. Jaeger, "Deep echo state networks: A simulation study," *Neurocomputing*, vol. 397, pp. 287–296, 2020.

[110] G. Yu and Z. Tan, "Hierarchical echo state networks with group sparse regularization," *Neurocomputing*, vol. 434, pp. 214–223, 2021.

[111] J. Dambre, D. Verstraeten, B. Schrauwen, and S. Massar, "Information processing capacity of dynamical systems," *Scientific reports*, vol. 2, no. 1, pp. 1–6, 2012.

[112] W. Maass, "Noise as a resource for computation and learning in networks of spiking neurons," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 860–880, 2014.

[113] S. Schliebs and M. Fiasche, "Neuromorphic hardware learns to learn," *Frontiers in Neuroscience*, vol. 6, p. 72, 2012.

[114] M. Lukoševičius, N. Kasabov, and Z. Gholami Doborjeh, "Practical properties of a liquid state machine using neucube for spatio- and spectro-temporal data," *Neural Networks*, vol. 78, pp. 39–50, 2016.

[115] D. Bacciu, C. Colombo, C. Gallicchio, and A. Micheli, "Emerging trends in edge-ai," *Journal of Machine Learning Research*, vol. 20, no. 158, pp. 1–5, 2019.

[116] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2330–2343, 2014.

[117] A. Tavanaei and A. S. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, 2019.

[118] R. Brette, "Philosophy of the spike: rate-based vs. spike-based theories of the brain," *Frontiers in systems neuroscience*, p. 151, 2015.

[119] M. Lukoševičius, H. Jaeger, and B. Schrauwen, "Reservoir computing trends," *KI-Künstliche Intelligenz*, vol. 26, no. 4, pp. 365–371, 2012.

[120] S. Haykin, *Adaptive Filter Theory*, 4th ed. Prentice Hall, 2001.

[121] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2014.

[122] S. Dasgupta, F. Wörgötter, and P. Manoonpong, "Information theoretic self-organised adaptation in reservoirs for temporal memory tasks," in *Engineering Applications of Neural Networks: 13th International Conference, EANN 2012, London, UK, September 20-23, 2012. Proceedings 13*. Springer, 2012, pp. 31–40.

[123] R. Legenstein and W. Maass, "Edge of chaos and prediction of computational performance for neural circuit models," *Neural networks*, vol. 20, no. 3, pp. 323–334, 2007.

[124] L. Büsing, B. Schrauwen, and R. Legenstein, "Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons," *Neural computation*, vol. 22, no. 5, pp. 1272–1311, 2010.

[125] M. Dale, J. F. Miller, S. Stepney, and M. A. Trefzer, "A substrate-independent framework to characterize reservoir computers," *Proceedings of the Royal Society A*, vol. 475, no. 2226, p. 20180723, 2019.

[126] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Networks*, vol. 115, pp. 100–123, 2019.

[127] D. V. Christensen, R. Dittmann, B. Linares-Barranco, A. Sebastian, M. Le Gallo, A. Redaelli, S. Slesazeck, T. Mikolajick, S. Spiga, S. Menzel *et al.*, "2022 roadmap on neuromorphic computing and engineering," *Neuromorphic Computing and Engineering*, vol. 2, no. 2, p. 022501, 2022.

[128] M. Dale, J. F. Miller, S. Stepney, and M. A. Trefzer, "Reservoir computing in material substrates," *Reservoir Computing: Theory, Physical Implementations, and Applications*, pp. 141–166, 2021.

[129] K. Nakajima, "Physical reservoir computing—an introductory perspective," *Japanese Journal of Applied Physics*, vol. 59, no. 6, p. 060501, 2020.

[130] J. H. Jensen and G. Tufte, "Reservoir computing in artificial spin ice," in *Artificial Life Conference Proceedings 32*. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ..., 2020, pp. 376–383.

[131] J. H. Jensen, E. Folven, and G. Tufte, "Computation in artificial spin ice," in *Artificial Life Conference Proceedings*. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ..., 2018, pp. 15–22.

[132] Y. Kuriki, J. Nakayama, K. Takano, and A. Uchida, "Impact of input mask signals on delay-based photonic reservoir computing with semiconductor lasers," *Optics Express*, vol. 26, no. 5, pp. 5777–5788, 2018.

[133] L. Larger, M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutierrez, L. Pesquera, C. R. Mirasso, and I. Fischer, "Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing," *Optics Express*, vol. 20, no. 3, pp. 3241–3249, 2012.

[134] D. Brunner, M. C. Soriano, C. R. Mirasso, and I. Fischer, "Parallel photonic information processing at gigabyte per second data rates using transient states," *Nature Communications*, vol. 4, no. 1, pp. 1364–1364, 2013.

[135] M. C. Soriano, D. Brunner, M. Escalona-Morán, C. R. Mirasso, and I. Fischer, "Minimal approach to neuro-inspired information processing," *Frontiers in computational neuroscience*, vol. 9, p. 68, 2015.

[136] Y. Zhong, J. Tang, X. Li, B. Gao, H. Qian, and H. Wu, "Dynamic memristor-based reservoir computing for high-efficiency temporal signal processing," *Nature communications*, vol. 12, no. 1, p. 408, 2021.

[137] J. Moon, W. Ma, J. H. Shin, F. Cai, C. Du, S. H. Lee, and W. D. Lu, "Temporal data classification and forecasting using a memristor-based reservoir computing system," *Nature Electronics*, vol. 2, no. 10, pp. 480–487, 2019.

[138] C. Du, F. Cai, M. A. Zidan, W. Ma, S. H. Lee, and W. D. Lu, "Reservoir computing using dynamic memristors for temporal information processing," *Nature Communications*, vol. 8, no. 1, pp. 1–10, 2017.

[139] T. Zheng, W. Yang, J. Sun, X. Xiong, Z. Wang, Z. Li, and X. Zou, "Enhancing performance of reservoir computing system based on coupled mems resonators," *Sensors*, vol. 21, no. 9, p. 2961, 2021.

[140] T. Lymburn, S. D. Algar, M. Small, and T. Jüngling, "Reservoir computing with swarms," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 31, no. 3, p. 033121, 2021.

[141] K. Nakajima, H. Hauser, R. Kang, E. Guglielmino, D. G. Caldwell, and R. Pfeifer, "A soft body as a reservoir: case studies in a dynamic model of octopus-inspired soft robotic arm," *Frontiers in computational neuroscience*, vol. 7, p. 91, 2013.

[142] T. Hülser, F. Köster, L. Jaurigue, and K. Lüdge, "Role of delay-times in delay-based photonic reservoir computing," *Optical Materials Express*, vol. 12, no. 3, pp. 1214–1231, 2022.

[143] M. Le Berre, E. Ressayre, A. Tallet, H. Gibbs, D. Kaplan, and M. Rose, "Conjecture on the dimensions of chaotic attractors of delayed-feedback dynamical systems," *Physical Review A*, vol. 35, no. 9, p. 4020, 1987.

[144] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar, "Optoelectronic reservoir computing," *Scientific Reports*, vol. 2, no. 1, pp. 287–287, 2012.

[145] F. Duport, B. Schneider, A. Smerieri, M. Haelterman, and S. Massar, "All-optical reservoir computing," *Optics express*, vol. 20, no. 20, pp. 22 783–22 795, 2012.

[146] D. Brunner, B. Penkovsky, B. A. Marquez, M. Jacquot, I. Fischer, and L. Larger, "Tutorial: Photonic neural networks in delay systems," *Journal of Applied Physics*, vol. 124, no. 15, p. 152004, 2018.

[147] K. Hicke, M. A. Escalona-Morán, D. Brunner, M. C. Soriano, I. Fischer, and C. R. Mirasso, "Information processing using transient dynamics of semiconductor lasers subject to delayed feedback," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 19, no. 4, pp. 1 501 610–1 501 610, 2013.

[148] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science*, vol. 197, no. 4300, pp. 287–289, 1977.

[149] A. Dhooge, W. Govaerts, Y. A. Kuznetsov, H. G. E. Meijer, and B. Sautois, "New features of the software matcont for bifurcation analysis of dynamical systems," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 14, no. 2, pp. 147–175, 2008.

[150] S. Boccaletti, J. Kurths, G. V. Osipov, D. L. Valladares, and C. Zhou, *Synchronization of chaotic systems.* Wiley Online Library, 2000.

[151] T. Kapitaniak, *Controlling chaos.* Academic Press, 1996.

[152] Y. Kuramoto and I. Nishikawa, "Statistical macrodynamics of large dynamical systems. case of a phase transition in oscillator communities," *Journal of the Physical Society of Japan*, vol. 49, no. 7, pp. 2355–2360, 1987.

[153] R. Hegger, H. Kantz, and T. Schreiber, "Computation of lyapunov exponents from time series: Efficiency vs. accuracy," *CHAOS*, vol. 9, no. 3, pp. 636–643, 1999.

[154] J. C. Sprott, *Chaos and Time-Series Analysis.* Oxford University Press, 2003.

[155] T. Erneux, *Applied delay differential equations.* Springer Science & Business Media, 2009, vol. 3.

[156] J. Schumacher, H. Toutounji, and G. Pipa, "An introduction to delay-coupled reservoir computing," in *Artificial Neural Networks: Methods and Applications in Bio-/Neuroinformatics.* Springer, 2015, pp. 63–90.

[157] P. Antonik, F. Duport, M. Hermans, A. Smerieri, M. Haelterman, and S. Massar, "Online training of an opto-electronic reservoir computer ap-

plied to real-time channel equalization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 11, pp. 2686–2698, 2016.

[158] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[159] L. Junges and J. A. Gallas, "Intricate routes to chaos in the mackey–glass delayed feedback system," *Physics Letters A*, vol. 376, no. 30-31, pp. 2109–2116, 2012.

[160] E. Shahverdiev, R. Nuriev, R. Hashimov, and K. Shore, "Chaos synchronization between the mackey–glass systems with multiple time delays," *Chaos, Solitons and Fractals*, vol. 29, no. 4, pp. 854–861, 2006.

[161] S. Sano, A. Uchida, S. Yoshimori, and R. Roy, "Dual synchronization of chaos in mackey-glass electronic circuits with time-delayed feedback," *Physical Review E*, vol. 75, no. 1, p. 016207, 2007.

[162] L. Berezansky and E. Braverman, "Mackey-glass equation with variable coefficients," *Computers & Mathematics with Applications*, vol. 51, no. 1, pp. 1–16, 2006.

[163] S. Documentation, "Simulation and model-based design," 2020. [Online]. Available: https://www.mathworks.com/products/simulink.html

[164] *MATLAB version 9.10.0.1613233 (R2021a)*, The Mathworks, Inc., Natick, Massachusetts, 2021.

[165] A. C. McDonnell and M. A. Trefzer, "The effect of system timescale on virtual node connectivity within delay-feedback reservoirs," in *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2023, pp. 1–8.

[166] A. F. Atiya and A. G. Parlos, "New results on recurrent network training: unifying the algorithms and accelerating convergence," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 697–709, 2000.

[167] U. Hübner, N. B. Abraham, and C. O. Weiss, "Dimensions and entropies of chaotic intensity pulsations in a single-mode far-infrared nh 3 laser," *Physical Review A*, vol. 40, no. 11, p. 6354, 1989.

[168] K. Harkhoe, G. Verschaffelt, A. Katumba, P. Bienstman, and G. Van der Sande, "Demonstrating delay-based reservoir computing using a compact photonic integrated chip," *Optics express*, vol. 28, no. 3, pp. 3086–3096, 2020.

[169] Y. Yang, P. Zhou, P. Mu, and N. Li, "Time-delayed reservoir computing based on an optically pumped spin vcsel for high-speed processing," *Nonlinear Dynamics*, vol. 107, no. 3, pp. 2619–2632, 2022.

[170] Z. Liang, M. Zhang, C. Shi, and Z. R. Huang, "Real-time respiratory motion prediction using photonic reservoir computing," *Scientific Reports*, vol. 13, no. 1, p. 5718, 2023.

[171] S. Masaad, E. Gooskens, S. Sackesyn, J. Dambre, and P. Bienstman, "Photonic reservoir computing for nonlinear equalization of 64-qam signals with a kramers–kronig receiver," *Nanophotonics*, 2022.

[172] H. Dai and Y. K. Chembo, "Rf fingerprinting based on reservoir computing using narrowband optoelectronic oscillators," *Journal of Lightwave Technology*, vol. 40, no. 21, pp. 7060–7071, 2022.

[173] L. Appeltant *et al.*, "Reservoir computing based on delay-dynamical systems," *These de Doctorat, Vrije Universiteit Brussel/Universitat de les Illes Balears*, 2012.

[174] A. Vargha and H. D. Delaney, "A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[175] B. van der Pol, "Lxxxv. on oscillation hysteresis in a triode generator with two degrees of freedom," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 43, no. 256, pp. 700–719, 1922.

[176] B. Van Der Pol and J. Van Der Mark, "Lxxii. the heartbeat considered as a relaxation oscillation, and an electrical model of the heart," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 6, no. 38, pp. 763–775, 1928.

[177] W. Nicola and C. Clopath, "Supervised learning in spiking neural networks with force training," *Nature communications*, vol. 8, no. 1, p. 2208, 2017.

[178] K. Yeo, "Data-driven reconstruction of nonlinear dynamics from sparse observation," *Journal of Computational Physics*, vol. 395, pp. 671–689, 2019.

[179] M. R. E. U. Shougat and E. Perkins, "The van der pol physical reservoir computer," *Neuromorphic Computing and Engineering*, vol. 3, no. 2, p. 024004, 2023.

[180] M. Cooper, P. Heidlauf, and T. Sands, "Controlling chaos—forced Van der Pol equation," *Mathematics*, vol. 5, no. 4, p. 70, 2017.

[181] X. Liang, H. Li, A. Vuckovic, J. Mercer, and H. Heidari, "A neuromorphic model with delay-based reservoir for continuous ventricular heartbeat detection," *IEEE Transactions on Biomedical Engineering*, vol. 69, no. 6, pp. 1837–1849, 2021.