Scaling up ESNs with Heterogeneous Reservoirs

Chester Wringe

Doctor of Philosophy

University of York
Department of Computer Science
September 2024

Abstract

Reservoir Computing (RC) is an unconventional computing model first designed for training Recurrent Neural Networks (RNNs). The technique involves randomly initiating the weights of the inner neural network (called a reservoir), then training the observed output of this reservoir using a single-layer readout.

The design of the reservoir computing model allows us to treat the reservoir as a black box. This makes RC ideal for computing with unconventional physical materials, as we can treat the material as a black box reservoir. This physical or "in materio" reservoir computing allows us to exploit the dynamics of physical systems, and make strides towards smaller-scale, low power computing.

One challenge in the field of in materio RC is that the materials with the most interesting dynamics are difficult or impossible to work with at large scale. In this thesis, we study how we might scale up reservoir computers by combining multiple smaller reservoirs together.

Our approach of combining reservoirs allows us to have reservoirs with different properties, such as distinct materials or timescales. This approach allows us to tackle more complex tasks than are typically possible with classical RC, such as the sleep apnea benchmark.

Our work is completed in simulation. In service of this, we work to bring our simulation closer to the constraints of physical RC with "mock materials" we design. We design a technique for building heterogeneous reservoirs for complex tasks.

We find that heterogeneous reservoirs are not suited to all RC tasks, such as a variation we design of the Multiple Superimposed Oscillators (MSO*) benchmark. We propose alternative reservoir designs which may be found to be more effective in future work.

Data and code related to this thesis is available from https://github.com/FromAnkyra.

Declaration and Related Publications

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

This thesis contains material from papers that are published or under review. These papers are listed below, as are the chapters that are based on the material.

Related Publications:

1. Chester Wringe, Susan Stepney, and Martin Albrecht Trefzer. “Modelling and Evaluating Restricted ESNs”. In: *Lecture Notes in Computer Science*. Vol. 14003. UCNC 2023, Jacksonville FL, USA, 2023.

2. Chester Wringe, Susan Stepney, and Martin A Trefzer. “Restricted reservoirs on heterogeneous timescales”. en. In: *Artificial Neural Networks and Machine Learning*. Vol. 15025. LNCS. ICANN 2024, Lugano, Switzerland: Springer 2024, Sept. 2024, pp. 168–183.

3. Chester Wringe, Susan Stepney, and Martin Trefzer. “Modelling and Evaluating Restricted ESNs on Single-and Multi-Timescale Problems”. In: *Natural Computing* (2023). DOI: https://doi.org/10.21203/rs.3.rs-3758288/v1.

4. Martin Trefzer Chester Wringe and Susan Stepney. “Reservoir computing benchmarks: a tutorial review and critique”. In: *International Journal of Parallel, Emergent and Distributed Systems* 0.0 (2025), pp. 1–39. DOI: 10.1080/17445760.2025.2472211. eprint: https://doi.org/10.1080/17445760.2025.2472211. URL: https://doi.org/10.1080/17445760.2025.2472211.

Acknowledgements

In completing this thesis, there are a number of people whose help was invaluable, and to whom I must extend my thanks:

To my supervisors, Susan Stepney and Martin Trefzer, for striking the appropriate balance between guiding my research and letting me assert my own independance.

To my parents, Sandrine Bergès and Bill Wringe, for raising me in such a way that this endeavour was far less challenging than it could have been, and for supporting me throughout the writing of this thesis.

To my brother Max Bergès, for his daily texts.

To Ash Holland, who is the coolest person I know, and whose company never ceases to delight me, for consistently being there whenever I needed support.

To Ethan Holwill, for the weekly phonecalls that helped me keep track of time even when I was drowning in work, to the blonds, for the almost-weekly boardgames and takeaways, and to Abi Sutherland, for the parallel working sessions that kept me sane.

To Tian Gan, Andrew Walter, Rinku Sebastian, Penn Rainford, David Griffin, and Riv Waldegrave, for being excellent company to share an office with.

Chester Wringe,
York,
September 2024.

Contents

List of Tables

List of Figures

1

Introduction and Motivation

Reservoir Computing (RC) is an unconventional model of computation, originally based in Artificial Neural Networks (ANNs). RC is a method that allows us to train Recurrent Neural Networks (RNNs) cheaply and efficiently.

RNNs cannot be trained using classical methods such as backpropagation, and alternative methods such as Backpropagation Through Time (BPTT) are slow and computationally expensive. RC trains not the inner weight of the RNNs, but instead has them randomly initiated. Only the output nodes are trained, using methods such as linear regression.

Beyond its application in the field of ANNs, RC has also become a popular model in the fields of physical and in materio computing. The random initiation of the inner weights allows the internal ("Reservoir") state of the computer to be treated as a black box. This internal state can thus be replaced with any physical material, so long as the material has sufficiently rich dynamics, can receive inputs, and produce outputs.

Physical reservoir computers can lead to a number of advantages, such as low power usage or exploiting the dynamics that are unique to a given system. However, the practical uses of these physical reservoirs are currently limited, as the same properties that make a system suited to reservoir computers make them difficult to scale up to tackle more complex problems.

Materials that make for good reservoir computers are often difficult to work with at large scales, and the computational capacity of the computer may not increase linearly with the size of the material substrate used in the reservoir. Additionally, the properties that may make a certain material ideal for solving one part of a complex task may also lead to poor results when trying to solve another part of it.

In this thesis, we work towards scaling up reservoir computers by combining

multiple small reservoirs together. We focus on combining reservoirs with heterogeneous properties to solve more complex problems.

In chapter 2, we provide a background on different Reservoir Computing models. We focus on Physical RC and the limitations we find with them, as well as the existing literature on combining reservoir computers.

In chapter 3, we provide further background in the form of a review and discussion reservoir computing benchmarks, as well as the evaluation of Reservoir Computing in general.

In chapter 4, we establish our research questions, and we discuss the methods that we employ to answer them. We proceed to address these questions in the following four chapters.

In chapter 5, we introduce a framework for describing combinations of reservoirs. We use this framework to establish a baseline performance on which to base future experiments for a number of popular benchmarks.

In chapter 6, we explore combining reservoirs that work on heterogeneous timescales. In order to do this, we introduce a model for simulating multi-timescale reservoirs. In this chapter, we also introduce “mock materials”, which allow us to simulate the constraints of physical reservoirs while still working in simulation.

In chapter 7, we study combinations of different mock materials, through the lens of a complex benchmark with multiple inputs. This benchmark has three inputs which, while related, operate on three distinct timescales.

In chapter 8, we combine the work of the previous chapters to create a multi-timescale, multi-material reservoir, and compare it to every other model studied in this thesis. Through this, we develop a process to tailor these combined reservoirs to the task being studied.

Finally, in chapter 9, we review the findings of the previous chapters, and look to the future to how these findings further the field of reservoir computing.

2 Background and Context

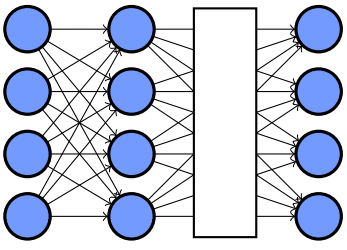
In this chapter, we introduce a number of concepts that will be used in used in this thesis going forward: Reservoir Computing, and different models thereof: Physical Reservoir Computers and Echo State Networks. We then introduce the ways that multiple Reservoir Computers have been combined in the literature.

2.1 Reservoir Computing

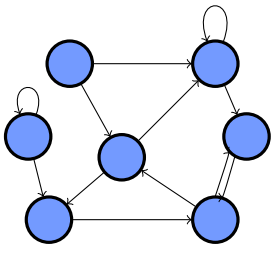
Recurrent Neural Networks (RNNs) are a type of Artificial Neural Network (ANN) distinguished by their recurrent connections. Unlike feed-forward neural networks, where each node only has outgoing edges to nodes in the next layer, a node in a RNN can share an edge with any other (fig. 2.1).

RNNs are computationally interesting for several reasons:

- RNNs are well-suited to integrating nonlinearity, a necessity for a number of computational functions, such as **XOR**. Details on how to implement this nonlinearity can be found in Williams and Zipser [166, sec.2].



(a) A feed-forward neural network: This network is composed of multiple layers of nodes. In each layer, a node is connected to all the nodes in the next layer. There are typically many such layers, refered to as “hidden” layers.



(b) A recurrent neural network. Here, instead of layers of nodes, each node may be connected to any other, including itself. In the ESN model, these connections are randomly initiated to a set sparsity.

Figure 2.1: Two types of Artificial Neural Networks.

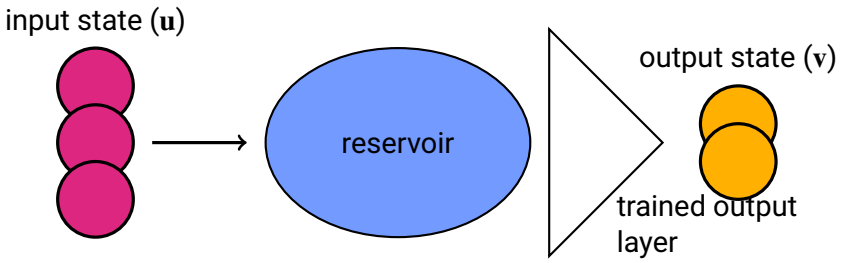


Figure 2.2: An abstract diagram of a reservoir computer. It shows a reservoir as a "black box", of which the content is undefined (and may be unknown). The input is fed to this reservoir, and then the output of the reservoir is trained to be the system's output.

- RNNs are partially driven by their own past state, allowing for "dynamical memory" [61, 88].

Unlike feed-forward neural networks, RNNs cannot be trained with traditional backpropagation methods. While methods such as backpropagation through time (BPTT) [5] and evolved networks [128] have been proposed to train the internal weights in service of a given task, this training is computationally expensive.

Reservoir Computing (RC, fig. 2.2) is an approach to training RNNs that seeks to address this issue. The technique consists of leaving the RNN's reservoir weights untouched, and instead producing the desired output through a trained linear combination of the internal state.

This approach was proposed independently by Herbert Jaeger for discrete networks as the Echo State Network [63] and by Wolfgang Maass for spiking networks as the Liquid State Machine (LSM) [94]. In a 2009 Special Issue on the topic, which included a survey of the RC field as it stood at the time [88], the Reservoir Computing model was made more precise.

Reservoir Computers can be trained using one-shot training, typically using linear regression or ridge regression[1]. This allows for much faster training than BPTT and evolved networks. Another advantage of this training method is that if two tasks can be performed on a reservoir driven by the same input signal, the reservoir only needs to be run once, and it can then be trained for both tasks ([85], discussed in more detail in sec. 3.8.3).

One of the strengths of Reservoir Computing is that any system that can be driven

[1] A guide to using these methods can be found at Lukoševičius [87]. Some instances of unsupervised training can also be found in the literature [7].

by an input signal, has sufficiently rich dynamics to transform it, and can produce a readable output signal can act as a reservoir. This means that we can effectively treat the reservoir as a black box, which has led to the field of Physical RC.

With this faster training requiring low power consumption, as well as strength in tasks requiring fading memory, Reservoir Computing from traditional machine learning methods. While it is occasionally used for the same applications (see section 3.7), it can be used in very different contexts, such as embedded, low-power devices[33].

2.2 Dynamical Systems

A dynamical system is described by its state's time evolution, within a given state space. A continuous time dynamical system can be defined with an ordinary differential equation, as $\dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t), \mathbf{b})$, where $t \in \mathbb{R}$ is the time, $\mathbf{x}$ is the state, $\mathbf{u}$ is the input, and $\mathbf{b}$ is a time independent parameter. A discrete time dynamical system can be defined with a difference equation, as $\mathbf{x}(t+1) = f(\mathbf{x}(t), \mathbf{u}(t), \mathbf{b})$, where $t \in \mathbb{N}$ is the time. More sophisticated dynamical systems can be defined; for example, using delay differential equations to incorporate memory of previous states, using partial differential equations to incorporate spatial properties.

The term $\mathbf{u}(t)$ captures the inputs to an *open* (non-autonomous) dynamical system. If $\mathbf{u}(t) = 0$, the system has no inputs, and is *closed* (autonomous), with its behaviour depending on only its initial state and other fixed parameters. A given system's internal dynamics f and inputs $\mathbf{u}$ may be understood and known, or may be unknown.

2.3 Physical Reservoir Computing

In materio Computing [55] is the term for computation performed using a direct mapping of a computational model to a physical material[2]. Early instantiations of in materio computing involved evolving physical configurations of a given substrate to perform a given task, such as signal classification [146].

[2] See Stepney [143] for a discussion of the difference between in materio and classical computing (section 1.1) and what it means for a physical system to compute (section 2).

Reservoir Computing is a powerful model for in materio computing, as it can be mapped to any physical substrate with sufficiently rich dynamics. One of the earliest works in the field involved using a bucket of water as a reservoir [39].

Physical RC has been reviewed extensively by Tanaka et al. [145]; Zhang and Vargas [175] review a number of physical RC systems grouped by application. Tutorials on how to implement physical RCs have been written by Stepney [143] and Cucchi et al. [22].

Physical Reservoir Computing presents us with several challenges. While any material can be used to make a reservoir, even a rusty nail [144], not every material will make a *good* reservoir[3].

A truism of reservoir computing is that an RNN-based Reservoir Computer improves as nodes are added, but this does not necessarily translate to physical reservoirs. This problem may arise for a number of reasons:

- the computational capacity of the material stems from a property that does not scale up linearly when increasing the quantity of material. One example of this is the phenomenon of "edge effects", wherein the computation of some magnetics–based reservoir computers take place on the edges of the material [29].

- the material is only available in very small quantities, making them equivalent to very small simulated reservoirs [25].

As such, it is frequently necessary to compare these physical reservoirs to simulations with a very small number of nodes, while the state-of-the-art in simulated reservoirs performs tasks on much larger reservoirs.

2.4 Echo State Networks

Echo State Networks [63, 68] (fig. 2.3) are modelled as random recurrent neural networks, which have the form of a discrete time dynamical system. The vanilla

[3] What makes a reservoir "good" is discussed in chapter 3.

form of an ESN is:

$$\mathbf{x}(t+1) = f(\mathbf{W_u}\mathbf{u}(t+1)+\mathbf{W}\mathbf{x}(t)) \quad (2.1)$$

$$\mathbf{v}(t+1) = \mathbf{W_v}\mathbf{x}(t+1) \quad (2.2)$$

Equation 2.1 defines the dynamics of the reservoir, where $\mathbf{x}$ is the N-d reservoir state; $\mathbf{W}$ is a random $N \times N$ weight matrix; $\mathbf{u}$ is the N_u-d input; $\mathbf{W}_u$ is a random input $N \times N_u$ weight matrix; and f is a non-linear function (typically the tanh or other sigmoid function).

Certain different transfer equations have been proposed: Leaky-integrator neurons are introduced into reservoir computing by Jaeger et al. [69], which helps model the loss of information, particularly when modelling physical materials which decay over time. This model also allows for better performance on certain problems, such as Multiple Superimposed Oscillators (MSO) over several thousand timesteps (see section 3.5.2). The leaky-integrator model introduces α, where $0 < \alpha < 1$ and where $1-\alpha$ is the rate of decay. The transfer equation becomes:

$$\mathbf{x}(t+1) = (1-\alpha)\mathbf{x}(t) + \alpha(f(\mathbf{W_u}\mathbf{u}(t+1)+\mathbf{W})\mathbf{x}(t)) \quad (2.3)$$

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

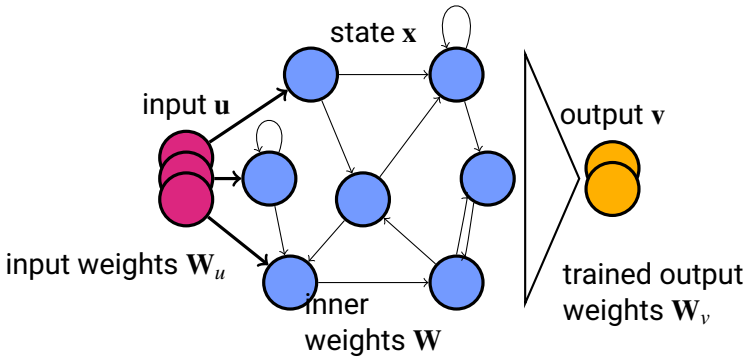
The ESN equation can further be modified to allow for feedback.

In this work, we use a modified transfer equation to allow us to model a more physically accurate system (see [141] for a discussion.):

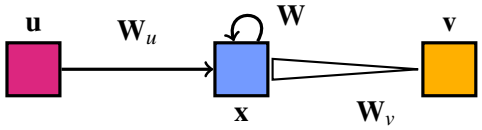
$$\mathbf{x}(t+1) = f(\mathbf{W_u}\mathbf{u}(t)+\mathbf{W}\,\mathbf{x}(t)) \quad (2.5)$$

$$\mathbf{v}(t+1) = \mathbf{W_v}\,\mathbf{x}(t) \quad (2.6)$$

Some best practices for designing ESNs are given by Lukoševičius [87]. These include scaling the input, ensuring that an initial transient (or "washout period") is of an adequate length, and scaling the spectral radius of the reservoir's weight matrix.



(a) An example classical ESN



(b) Abstracted representation of a classical ESN

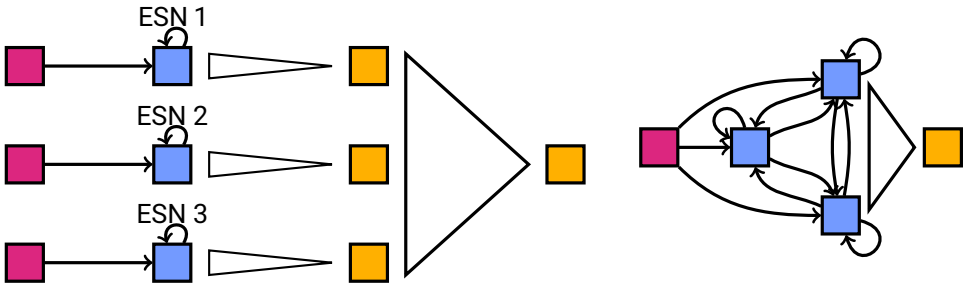
Figure 2.3: (a) An example classical ESN with 7 nodes. This ESN takes a 3-d vector of inputs $\mathbf{u}$, which are sent to the inner state $\mathbf{x}$ through weighted edges $\mathbf{W}_u$. The weights within the inner state, **W**, are recurrent and randomly set. The 2-d output vector $\mathbf{v}$ receives the inner state through trained edges $\mathbf{W}_v$. (b) An abstract representation of the different components of a general ESN.

2.4.1 Scaling ESN Weight Matrices

The spectral radius is a global parameter of the Echo State Network that can affect the performance: it has an effect on the "echo state property" of the ESN, the property that ensures the memory of the reservoir fades over enough timesteps, instead of being dependent on its initial state. The spectral radius ρ of an ESN's weight matrix $\mathbf{W}$ is the largest eigenvalue of the matrix, and it can be normalised to a value α by multiplying the weight matrix $\mathbf{W} \times \frac{\alpha}{\rho}$.

It is commonly held that a larger spectral radius will lead to violating the echo state property, while keeping the spectral radius smaller than 1 ensures the echo state property [104].

Yildiz, Jaeger, and Kiebel [173] examine this supposition and find it to be a misconception: simply scaling the weight matrix to a spectral radius below unity is neither sufficient nor necessary to ensure the echo state property. In this work, we perform preliminary experiments to determine whether scaling the weight matrix leads to better results, and proceed accordingly.



(a) A modular ESN. (b) A restricted ESN.

Figure 2.4: Two ways of combining reservoir computers. a modular ESN (a) where all the component reservoirs are trained independently of each other, and the final result is combined at the end, and a restricted ESN (b) where the reservoir layer is split up into regions.

2.5 Reservoir Computing Paradigms

In this work, we primarily focus on Echo State Networks (ESNs) and on *in-materio reservoirs*. Some other Reservoir Computing paradigms are:

- *Liquid State Machines* [39] are a Reservoir Computing paradigm based on Spiking Neural Networks.
- *Delay-line* reservoir [4] uses a single dynamical node, using a mask on the input such that it is multiplexed in time, rather than in space.

2.6 Combining Echo State Networks

We identify two distinct ways of combining ESNs in the literature (fig. 2.4). We dub these techniques modular (fig. 2.4a) and restricted (fig. 2.4b) ESNs.

2.6.1 Modular ESNs

A *modular* ESN (fig. 2.4a) typically comprises multiple individual reservoirs, each with its own input layer and trained output weights, connected in a variety of ways.

The Dynamic Feature Discoverer (DFD) [65] is a modular reservoir based on Deep Belief Networks, with the ESNs being components of a larger system. The ESNs may be replaced by other components, such as Extreme Learning Machines. The ESNs are arranged hierarchically, with each ESN being fed the standard input as well as the outputs of all the ESNs lower in the hierarchy. This hierarchy also allows the DFD

to contain separate timescales, such that each ESN in one level of the hierarchy runs more slowly than those in the previous levels.

Modular ESNs are also used in acoustic modelling [149, 150]. This model is based on the Hidden Markov Model, with the different ESNs with different timescales arranged linearly and hierarchically, with each reservoir processing dynamics that are slower than the previous ones.

The ConvESN [93] is a modular reservoir model based on Convolutional Neural Networks. The reservoirs are arranged in parallel and analyse dynamics at different timescales. The trained outputs of the ESN are then joined together in a convolutional layer.

2.6.2 Restricted ESNs

A *restricted ESN* (fig. 2.4b) has the same overall structure as a single ESN, with one input layer and one output layer. Its internal reservoir (a random RNN in the ESN model) has its overall state partitioned into "subreservoirs" with typical RNN connections within a subreservoir, and *restricted* connections between the subreservoirs[4]. There are several models in the literature that follow this structure.

The dual-reservoir network (DRN) [92] connects two subreservoirs in the network with an "unsupervised encoder", for which the weights are chosen using Principal Component Analysis (PCA). Triefenbach et al. [150] have a bidirectional dual-reservoir model, which consists of two subreservoirs running in parallel, with one of the subreservoirs receiving the inputs in chronological order, and the other receiving its inputs in reverse chronological order.

The Reservoir of Reservoirs (RoR) [23] is a model with dense connections within each subreservoir, and sparse random connections between subreservoirs. Two models are investigated: RoR, where the inputs are sent to only one subreservoir, and RoR-IA, where the inputs are sent to all of the subreservoirs. The multilayered echo state machine (ML-ESM) [95] arranges the subreservoirs sequentially, with each subreservoir fully connected to its neighbouring subreservoirs with fixed weights.

The Reservoir with Random Static Projections (R^2SP) [13] and the ϕESN [42]

[4] These ESNs are often referred to as "hierarchical ESNs", due to the structure of having a graph formed of smaller, densely connected subgraphs. We do not use this phrasing here, to avoid confusion with a reservoir where there is a hierarchy between the subreservoirs, with some subreservoirs driving others.

are both models that combine ESNs with an Extreme Learning Machine. There are several deep-ESN models [15, 43, 44, 91] based on deep learning networks. In these, the subreservoirs are arranged sequentially, and the inputs are sent only to the first subreservoir. They are compared to the grouped-ESN, where the subreservoirs are arranged in parallel, and deep-ESN Input-to-All (deep-ESN IA), a deep-ESN with inputs sent to every subreservoir. A variation on the deep-ESN, the Deep Fuzzy ESN (DFESN)[176] adds a "fuzzy reinforcement layer" running in parallel to each subreservoir.

The Decoupled ESN (DESN) [171] is a restricted ESN that tackles multi-timescale tasks by decoupling certain sections of the inner state from each other using a lateral inhibition unit.

The scale-free highly clustered ESN (SHESN) [31] has each subreservoir connected to every other subreservoir by "backbone nodes", of which there is one in every subreservoir. The hierarchically clustered ESN (HESN) [70] builds on the SHESN by allowing several backbone nodes per subreservoir, and by making them randomly connected as opposed to fully connected.

2.7 Echo State Networks on Multiple timescales

A number of modular ESNs operate on multiple timescales, including the Dynamical Feature Discoverer[65], the HMM–based acoustic modelling reservoirs([149, 150]), and the Convolutional ESN (ConvESN)[93]. These machines all analyse the dynamics of their input at different timescales, although the first two order the subreservoirs sequentially,

Multiple timescales can also be found in restricted ESNs[98]. In this example, the multiple timescales are simulated by varying the leakage rates of the subreservoirs. As the leakage rate of an ESN is analogous to the rate of decay of a physical material, changing the leakage rate changes the speed at which the material state decays. If we assume a singular material, then a higher leakage rate would correspond to the material moving faster.

2.8 Combining Physical Reservoir Computers

Some work has been done in combining physical reservoirs, although typically of homogeneous materials. Shen et al. [134] make a linear network of four photonic reservoirs, where each subreservoir follows the “delay-line” architecture of a single nonlinear node multiplexed over time. Mallinson et al. [96] make a network of ten physical reservoirs of Percolating Networks of Neuroparticles (PNNs), with the reservoirs each running in parallel.

Some work has been done towards designing reservoirs with heterogeneous subreservoirs using digital twins[99]. These twins are created out of Stochastic Differential Equations, and are intended to find optimal configurations for different subreservoirs before the physical reservoir is implemented.

3

Reservoir Computing Benchmarks: a review, a taxonomy, some best practices

3.1 Introduction

When evaluating a given system, a common approach is to use benchmarks. Many authors present their results using benchmark tasks, often using an appeal to the literature to justify the use of any given benchmark.

The field of Reservoir Computing has a large number of these benchmarks. Little is written on what these benchmarks are, why they are used, how to use them, what makes them a useful measure of the performance of a Reservoir Computer, or what different benchmarks have in common.

When we discuss scaling up Reservoir Computing, and improving its performance, it is important to be able to define what we mean by this. This review aims to answer these questions, offers a general paradigm through which to view benchmarks, and proposes some best practices when using benchmarks.

Reservoir Computers are uniquely suited to temporal tasks, which means that most of the benchmarks used to evaluate them are of a similar temporal nature. For historical reasons pertaining to its origin in neural networks, Reservoir Computing is also frequently used in classification tasks. Classification is a less intuitive way of using Reservoir Computing, and the tasks typically require some processing to be adapted to suit Reservoir Computing. For this reason, we focus primarily on temporal benchmarks.

3.1.1 What Are Benchmarks?

We introduce some terminology. Any given thing that one uses a Reservoir Computer to do is a *task*. A *benchmark* is a task used to evaluate the Reservoir Computer itself. This is as opposed to a *problem*, a task where a Reservoir Computer is used to provide a unique, insightful, or specific answer. While any given task may sometimes be employed as a benchmark (evaluating the RC), and sometimes as a problem (using the RC), we believe that the distinction between them is helpful to clarify arguments being made.

Benchmarks and problems can be distinguished by examining what the author argues. If the work is about solving a task, and the Reservoir Computer merely a means to an end, then the task in question is a problem. Conversely, if the work is about Reservoir Computers, or a specific implementation of Reservoir Computing, and the task is used to illustrate or evaluate the argument posed, then the task in question is a benchmark. A problem is exploratory, a benchmark is comparative. Whether the comparison is to other works in the literature, other implementations produced in the work, or even an arbitrary standard with no other basis for existing, the comparative element remains there.

Frequent recurrence of a task in the literature is neither a necessary nor a sufficient condition for it being a benchmark: certain problems may recur because Reservoir Computing is well-suited to solving them; certain tasks may be used as benchmarks for specific reasons, or for the first time. However, if a problem is frequently approached through the lens of reservoir computing, it may become a benchmark; see, for example, the spoken digits benchmark (sec. 3.7.2).

3.1.2 Why Are Benchmarks Used?

Given benchmarks are tasks used in a specific way, are they indeed a good way of evaluating Reservoir Computing in general, or just for those specific tasks? This is not a commentary on the quality or effectiveness of individual benchmarks (although such commentary is made below) but a look at the purpose of their use. Another way of evaluating Reservoir Computers is CHARC [26], and there are other methods for evaluating generic task-independent properties such as Kernel Rank, Generalisation Rank, and Memory Capacity (sec. 3.8). So why use problem-based benchmarks,

specific to a single task, instead of the task-independent property measures like CHARC?

We propose that the answer is twofold: a benchmark can help place a specific instance of a Reservoir Computer in context with the literature, and it is a measure that is both practical and quantifiable.

Placing our work in the context of the literature is useful for several reasons: we can compare actual results, often (although not always) through the use of a common and well-described experimental procedure. One such example is the procedure of the spoken digits task, as first described in [156]. It can also help us showcase particular properties of a Reservoir Computer, such as the long-short term memory benchmarks described in [66].

The quantifiable aspect is also useful: by being able to express the performance of a Reservoir Computer on a given benchmark, one can say that it is better or worse than others, as opposed to simply "different".

Although several good reasons for using benchmarks exist, care should be taken when choosing specific benchmarks for different cases. Certain benchmarks may be better suited to a given argument than others, while other arguments may be better served by using something else entirely.

3.1.3 Types of Reservoir Computing Works

In the literature, benchmarks can be used for practical reasons (the benchmark is especially suited to make the argument the work wishes to), or historical ones (other works of similar sort use that benchmark).

We divide the Reservoir Computing literature into three categories, similar to those described by Tanaka et al. [145]. Although not every reference falls cleanly into one or other of these, the categorisation provides a useful tool to identify what arguments a work is making, and which benchmarks may be best suited to support those arguments. We propose the following categories:

The Reservoir Computing approach to problem X. These works typically focus on particular problems, and whether Reservoir Computing can provide a solution. An example is investigating the usefulness of Reservoir Computing when controlling robots [115].

Works in this category typically do not use benchmarks, but they may still have a comparative element to them, such as comparing the Reservoir Computer's performance to the current state of the art; for example, such as discrete Hidden Markov Models in investigations of speech recognition [149, 156].

Although this category does not use benchmarks in service of the main argument, it is helpful to distinguish it, to provide a contrast to the categories that do, described below. It can also help to identify tasks that Reservoir Computing may be well suited to, such as Spoken Digit Recognition [156], or predicting Spatio-Temporally Chaotic Systems like the Kuramoto–Sivashinsky Equation [79], that may subsequently become good benchmarks.

A Novel or Improved Reservoir Computing model. Works in this category typically build on the original ESN [63] and LSM [94] Reservoir Computing Models, in order to change or improve them. They may use benchmarks to demonstrate that their model is better than previous models; for example, where Jaeger [69] introduces the concept of Leaky Integrator based ESNs. They may also use them to show that their model performs similarly to other models, but has other advantages, such as being simpler to implement in hardware. Examples of these include the introduction of the delay-line architecture [4], and in the analysis of reservoir topologies [120].

A physical implementation of a Reservoir Computer. These works are typically investigations into whether a particular material substrate is suitable as an implementation of a hardware Reservoir Computer, in terms of both performance and practicality. One of the first examples in this category, instantiating an LSM in a bucket of water [39], was used to illustrate properties of Reservoir Computers. The works in this category are reviewed by [145]; examples include a reservoir instantiated in a swarm model [89], in a robot modelled on an octopus arm [107], and in carbon nanotubes [25].

3.1.4 Structure of this chapter

In section 3.2 we provide some preliminary definitions and classifications: input and output types, supervised and unsupervised learning, and reservoir performance evaluation measures. In section 3.3 we identify the major classes of RC benchmarks

task type	dynamics	benchmark
imitation	known	NARMA-N
imitation	unknown	channel equalisation; two-jointed arm; Van del Pol controller; pole balancing
prediction	known	Mackey–Glass equation; Multiple Superimposed Oscillators (MSO); lazy eight figure; Lorenz chaos; Kuramoto–Sivashinski equation
prediction	unknown	sunspot numbers; Santa Fe LASER data; McMaster IPIX radar data
computation		XOR; parity
classification		MNIST handwritten digits spoken digits; Japanese vowels; Santa Fe sleep apnea data
property		linear memory capacity (MC); nonlinear memory capacity; kernel rank (KR); generalisation rank (GR)

Table 3.1: The benchmarks reviewed here, arranged by kind of task.

reviewed here, and explain how they differ. We next review multiple examples of each of these classes: imitation tasks (section 3.4), prediction tasks (section 3.5), computation tasks (section 3.6), classification tasks (section 3.7), and property measures (section 3.8). The specific benchmarks we review in these sections are given in table 3.1. We finish by describing best practices in choosing, using, and comparing benchmarks for reservoir computing, in section 3.9.

3.2 Reservoir Computing Terminology

Here we introduce and define some terminology and notation used in our reservoir computing benchmark review. Reservoir Computing as a general topic is reviewed in more detail in chapter 2. Here, we focus only on the aspects of RC that pertain to benchmark tasks.

3.2.1 Reservoir Input

Dimensionality

The general reservoir equation (eqn. 2.5) allows for a vector-valued input at each timestep. Most of the benchmarks described here, such as NARMA (section 3.4.1) and the sunspot prediction benchmark (section 3.5.6), have a single-valued, or scalar-valued, input at each timestep ($N_u = 1$). Some more complex multi-valued, or vector-valued, ($N_u > 1$) input benchmarks, such as the XOR task [39], the Van der Pol Oscillator task [45], and the sleep apnea task (sections 3.5.8, 3.7.4) are also available.

Stationary v. Non-Stationary Input

The input to a reservoir is *stationary* in the statistical sense if its statistical properties are not a function of time: it has no long term systematic changes or trends, no preferred zero time point. So a random uniform input is stationary; a steadily growing value is not. A cyclic, or seasonal, data set (for example, the sunspot data set in section 3.5.6) may be stationary if the statistical properties are calculated over timescales longer than the cycle length, but not over shorter ones.

A stationary data set is necessarily infinite (in theory), since a beginning or an end are special time points. In practice, no data set is infinite, but provided it is long enough that its end points are not visible to the system under test, and its statistical properties are the same across the data set, that is sufficient.

A data set generated from an equation can be stationary, and has the advantage that the data set can be extended, simply by running the generating equation for longer. Gilpin [50] provides a catalogue of 131 chaotic dynamics systems equations, suitable for generating stationary time series for benchmarks. The catalogue includes the Mackey–Glass (section 3.5.1) and Lorenz systems (section 3.5.4).

Experimentally-generated data sets can be stationary in the limited sense described above, if they can in principle be extended, by running the experiment for longer, and taking further observations. The Santa Fe LASER data (section 3.5.7) and sleep apnea data (sections 3.5.8, 3.7.4) are such examples, provided that the statistical values are calculated over long enough timescales to encompass the semi-periodic variations; the sunspot data (section 3.5.6) is also (we hope!) such an example.

Many experimental data sets are, however, naturally bounded, and so non-stationary. These include various spoken word (section 3.7.2) and spoken vowel (section 3.7.3) data sets: each word or vowel sound is discrete, with a beginning and an end point.

Washout Data

Typically, we reserve an initial subset of the data for a *washout* period. This puts the reservoir into a state so that results are independent of its earlier history, such as running a previous experiment, or of being switched off. This forgetting is possible because of the reservoir's fading memory property.

The need for a washout period can limit the training and testing periods of experimental data sets. Even for data sets that are effectively stationary, they nevertheless have a limited number of data points.

There is little guidance in the literature on how washout is handled for non-stationary data (for example, spoken digits, section 3.7.2). Several approaches are possible, depending on the precise application: (i) provide explicitly zero input as washout, and let the reservoir settle into a resting state; (ii) present the data multiple times; (iii) use an initial subsequence of the data as washout.

3.2.2 Reservoir Output

Real Valued Encoding

Here, the reservoir has one or more output nodes, each of which takes a real value, from which the result is read and used as the output of the task. If the output is a scalar, the reservoir has a single node encoding the scalar value, otherwise the reservoir has one node for each dimension of the output vector $\mathbf{v}$ (see equation 2.6).

The output is typically scaled to some specific range, such as $[-1, 1]$. The precision of the output is constrained by encoding or readout methods.

This encoding method is typically used for tasks where the reservoir is trained to behave like some dynamical system with a non-binary (or even continuous) encoding. The time-series input is matched by a corresponding time-series output of real values.

Categorical Encoding

When the reservoir's output is one of several categories, such as in classification tasks, a different form of output encoding can be used.

Multiple categories: one-hot encoding. In "one-hot" encoding, the reservoir has N output nodes, one for each categorical value. At any given timestep, the output of the reservoir is taken to be the value of the node with the highest response (the one 'hot' node). As such, this encoding is suited to many classification tasks.

In this encoding, each node is trained on a binary choice : whether the output of the reservoir is or is not its corresponding value.

When using this encoding, it is possible that multiple output nodes to have non-zero values. This may count as an error. In other cases, the strongest response be chosen.

Intermediate values may correspond to uncertainty of ambiguity measures.

Two categories : a single binary choice. When there are only two categories, or a single binary choice, the number of nodes can be reduced to one, with '0' denoting one category, and '1' denoting the other. In this case, distance from the 0 or 1 results may represent categorisation error.

3.3 A Taxonomy of Tasks

3.3.1 Main Classes of Benchmark Tasks

We identify five main classes of benchmark task. The first two relate to training a reservoir to *imitate* an open (non-autonomous) dynamical system, or to *predict* an autonomous one. The next two classes relate to reservoirs performing specific *computations* on their inputs, or *classifying* their inputs. The final class is a set of measures of specific *properties* of reservoirs.

Imitation Tasks

In an imitation task (see Figure 3.1), the Reservoir Computer is trained to replicate the dynamics of an *open* dynamical system. As such, it is given the same time

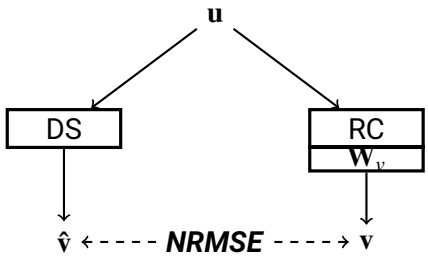


Figure 3.1: Training and testing stages of an imitation benchmark. The reservoir RC is fed the same time series input $\mathbf{u}$ as the target dynamical system DS. In the training stage, the output weights $\mathbf{W}_v$ of the reservoir are trained such that the resulting reservoir output $\mathbf{v}$ resembles the target dynamical system output $\hat{\mathbf{v}}$, the Normalised Root Mean Square Error, or NRMSE can be used as an evaluation of the training. In the testing stage, the trained reservoir output weights are used, and we evaluate the reservoir using the error between the observed reservoir output $\mathbf{v}$ and the target dynamical system output $\hat{\mathbf{v}}$. NRMSE is defined by $NRMSE(\hat{\mathbf{v}}, \mathbf{v}) = \sqrt{\frac{\langle(\hat{\mathbf{v}}-\mathbf{v})^2\rangle}{\langle(\hat{\mathbf{v}}-\langle\hat{\mathbf{v}}\rangle)^2\rangle}}$, where $\langle x\rangle$ is the mean $\frac{1}{N}\sum_{i=1}^{N} x_i$.

series input as is given to the target dynamical system. This input is typically, but not necessarily, uniform random noise, such as with NARMA (section 3.4.1). The reservoir is then trained to produce the same time series output that the target system produces.

The performance of a Reservoir Computer on imitation tasks can be tested in two different ways. The first of these is to measure the error between the reservoir output values and the desired output values. In an imitation task, the desired output values are the output of the dynamical system being imitated. If the dynamical system being imitated is part of a greater whole, such as the control system of a robot, then another adequate test may be, “can the trained reservoir adequately replace the target system in situ”?

Prediction Tasks

In a prediction task, the reservoir imitates a *closed* dynamical system. The reservoir must here predict the output of the dynamical system based on the previous output (figure 3.2). This is useful for learning the behaviour of a dynamical system where the input is unknown, or perhaps does not even exist. However, that is not a requirement. Any dynamical system that can be used for an imitation task can also be used for a prediction task: the only difference is the information provided to the reservoir.

During training, the reservoir is fed outputs from the target dynamical system as inputs, an the output weights trained to minimise the difference. There are two

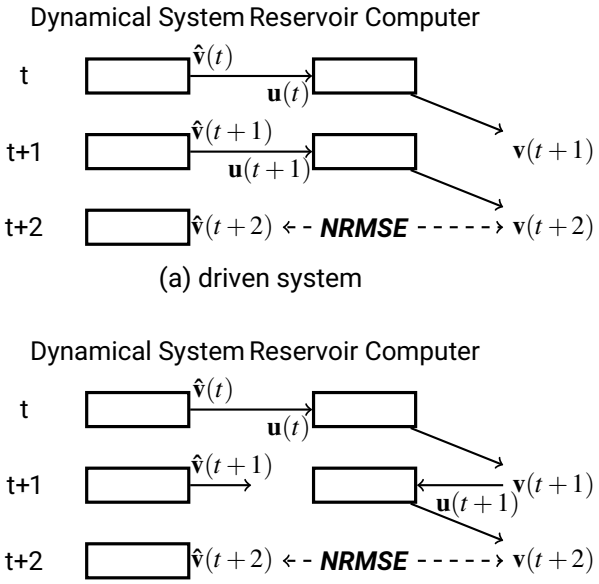


Figure 3.2: Training and testing stages of a prediction benchmark. During the training stage (a), the reservoir is given as inputs the target outputs of the dynamical system. The reservoir output weights are trained such that the reservoir outputs resemble the target outputs of the dynamical system. There are two cases for testing. (a) *Driven*: During the testing stage, the reservoir is again given the target outputs of the dynamical system. The reservoir outputs are compared to the target outputs of the dynamical system. (b) *Free-running*: During the testing stage, the reservoir is fed back its own output. The reservoir outputs are compared to the target outputs of the dynamical system.

options during the testing stage. In the *driven* case, the reservoir is driven with the target outputs as in the training stage. This provides a one-step look-ahead prediction, continually corrected by the actual system outputs to correct for drift. In the *free-running* case [67] the reservoir is fed back its own previous output as its next input. This provides a multi-step look-ahead prediction, potentially subject to accumulating errors.

Computation Tasks

In a computation task, the focus is on the reservoir performing logical or arithmetical computations on its inputs. Examples of logical operations include XOR (section 3.6.1), parity (section 3.6.2), and logic gates [136]. Examples of arithmetical operations are covered by other classifications: non-linear memory capacity (section 3.8.1) measures how well the reservoir can compute various polynomial functions of its inputs, and imitating or predicting systems with known dynamics (section 3.3.2) requires the reservoir to compute those dynamics.

Classification Tasks

Classification tasks have categorical outputs (section 3.2.2): the task is to output the category corresponding to the input. This reflects a typical use of neural networks, but here can explicitly capture the temporality of recognition, requiring memory over time.

There are three subclasses, depending on the form of the input to the reservoir

1. static non-temporal input; single categorical output : example, MNIST (section 3.7.1)

2. non-stationary temporal input; single categorical output : example, spoken digits (section 3.7.2) and spoken vowels (section 3.7.3)

3. stationary temporal input; time-series categorical output : example, sleep apnea data (section 3.7.4)

Speech recognition is a traditional classification benchmark task [49]. As such, it has also found a home as a Reservoir Computing benchmark, although not as

widespread as the dynamical systems tasks, which tend to translate better to the intrinsically temporal Reservoir Computing paradigm.

For non-stationary data tasks, the high dimensional input may be provided in parallel, as a single input vector, sequentially, as a non-stationary time series, or a combination of both. The categorical result is typically read out at a single time point after some time delay.

Tasks making a binary classification need only a single output node.

Tasks classifying into more than two categories typically used multiple output nodes, where each output node corresponds to one of the categories (see section 3.2.2).

A style of classification task better suited to the medium of RC is time series based classification tasks, where the goal is to identify what the state of the time series is currently. This type of task has frequently been used in biomedical applications of RC, such as classifying gestures based on EMG signals [34, 47], identifying arrhythmic heartbeats [21], or detecting seizures based on EEG signals [78]. No standard RC benchmark task for this style of classification appears to have yet emerged, however. We suggest that the Santa Fe sleep apnea data set would be a suitable candidate for an RC classification benchmark task (section 3.7.4).

Property Measures

Instead of measuring performance on a particular task, some benchmarks instead measure high level computational properties of the reservoir. The object of these tasks is not to maximise a success rate or minimise an error rate. Instead, we can take these properties and see how the behaviour of the reservoir may make them better suited to different tasks.

3.3.2 Other Axes of Classification

We identify some other axes of benchmark classification: these are not applicable in all cases.

- known *v* unknown dynamics
- supervised *v* unsupervised learning

Known *v.* Unknown Dynamics

Given the dynamical systems formalism, many Reservoir Computing tasks involve training them to behave like other dynamical systems, producing the same time series outputs, in order to predict or to replace those others.

Are the dynamics of the benchmark task known and understood by us, or are they unknown, with only the outputs and perhaps the inputs being observed? In the first case, we typically have a dynamical system model that we can use to generate the behaviour, such as NARMA (section 3.4.1) or Mackey–Glass (section 3.5.1). The second case typically uses observed natural phenomena, like LASER data (section 3.5.7) or sunspot observations (section 3.5.6), where we do not have a full model to generate the underlying dynamics, and are restricted to the supplied dataset.

Supervised *v.* Unsupervised Learning

Most Reservoir Computers are trained through linear regression. This requires training data where the outputs are known, which is not always available. There are, therefore, some works in the literature that are focused on finding a way to train reservoirs that do not include the expected output. Two examples of unsupervised learning benchmarks are channel equalisation (section 3.4.2) and pole balancing (section 3.4.5).

3.4 Dynamics Imitation Tasks

3.4.1 NARMA: Imitating Known Dynamics

NARMA (Nonlinear Auto-Regressive Moving Average) is a family of imitation tasks widely used as benchmarks. It is based on two systems for modelling time series based dynamical systems [164], the Auto-Regressive (AR) and Moving Average (MA) models.

Connor, Martin, and Atlas [19] and Connor, Atlas, and Martin [20] introduce and define a generalisation of linear ARMA models; their generic NARMA(p,q) model

depends on p past states and q past inputs:

$$x(t) = h(x(t-1), \ldots, x(t-p), u(t-1), \ldots, u(t-q)) + u(t) \tag{3.1}$$

In their particular analysis, h is an unknown smooth function to be estimated, and the input u has zero mean. They demonstrate a robust learning algorithm for approximating this function with a recurrent neural network (RNN).

Atiya and Parlos [5, eqn.79] introduce a second order NARMA system:

$$x(t+1) = 0.4x(t) + 0.4x(t)x(t-1) + 0.6u^3(t) + 0.1 \tag{3.2}$$

This is sometimes referred to in the literature as 'NARMA2' [41], but is not of the same functional form as the family of equations usually referred to as NARMA-N.

Atiya and Parlos [5, eqn.86] also introduce a tenth order NARMA system, which is now referred to as NARMA-10. The functional form has been generalised [118] as the Nth order, or NARMA-N, system:

$$x(t+1) = \alpha x(t) + \beta x(t) \left(\sum_{i=0}^{N-1} x(t-i) \right) + \gamma u(t-N+1)u(t) + \delta \tag{3.3}$$

Its non-linearity comes from the various $x(t)x(t-i)$ terms, and the $u(t-N+1)u(t)$ term; it also requires memory of the previous N states.

NARMA-10 using the original parameter values and the input stream generated from a uniform random distribution in the range [0, 0.5] is one of the commonest usages in the literature; see for example Verstraeten et al. [157], Rodan and Tiňo [120], Holzmann and Hauser [57], Paquot et al. [112], Goudarzi, Shabani, and Stefanovic [53] Vinckier et al. [160], Inubushi and Yoshimura [60], Dale [23].

Adapting this for different values of N reveals that the equation can rapidly diverge if not tuned correctly. Indeed, even the standard NARMA-10 setup does itself occasionally diverge [77]. Other authors have defined the form for further values

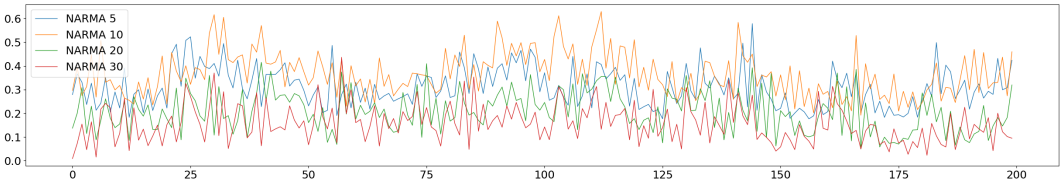


Figure 3.3: 200 timesteps of NARMA-5, NARMA-10, NARMA-20 (with tanh), and NARMA-30, using the starred parameter values from table 3.2, and the same input stream $u(t)$ drawn from $U[0, 0.5]$.

N	α	β	γ	δ	input	fig. 3.3	introduced in
5	0.3	0.05	1.5	0.1	$U[0, 0.2]$		Fujii and Nakajima [41, eqn.18]
5	0.3	0.05	1.5	0.1	$U[0, 0.5]$	*	Dale [24, chap.3]
10	0.3	0.05	1.5	0.1	$U[0, 0.5]$	*	Atiya and Parlos [5, eqn.86]
15	0.3	0.05	1.5	0.1	$U[0, 0.2]$		Fujii and Nakajima [41, eqn.18]
20	0.3	0.05	1.5	0.01	$U[0, 0.5]$	*	Rodan and Tino [118, eqn.6], with tanh(.)
20	0.3	0.05	1.5	0.1	$U[0, 0.2]$		Fujii and Nakajima [41, eqn.18]
30	0.2	0.04	1.5	0.001	$U[0, 0.5]$	*	Schrauwen et al. [131, sec.3]
30	0.2	0.004	1.5	0.001	$U[0, 0.5]$		Dale [23]

Table 3.2: Summary of NARMA-N parameters found in literature. The NARMA sequence is plotted for the starred values in Fig. 3.3

of N, and taken a variety of approaches to controlling divergence. For example, Schrauwen et al. [131, p.1164] defines NARMA-30 with different parameter values of $(0.2, 0.04, 1.5, 0.001)$, nevertheless, it too very occasionally diverges); Rodan and Tino [118, eqn.6] define NARMA-20 and add a $\tanh(.)$ wrapper to stop divergence; Fujii and Nakajima [41, eqn.18] define NARMA-5, 10, 15, and 20 with the original parameter values but a restricted input range $u \in \text{UNIFORM}[0, 0.2]$. Parameter values for these and other cases from the literature are given in table 3.2.

Furthermore, various NARMA time series each have their own range of output x values. For example, Atiya and Parlos [5]'s NARMA-10 x values range between (ignoring divergences) 0.15 and 1, Fujii and Nakajima [41]'s NARMA-10 with reduced u range has x values ranging between 0.15 and 0.25, and Schrauwen et al. [131]'s NARMA-30 x values range between (again ignoring divergences) 0 and 0.6. Even using consistent parameter values and input range yields systematically different output ranges for different values of N. This makes any performance comparison as a function of N problematic.

The relative simplicity of the task, and the ability to parameterise the difficulty by

using a range of N values, has made this benchmark popular when evaluating small reservoirs, as is often true for *in materio* reservoir computers [24]. As noted, however, however, there are several issues: the potential for divergence when generating the series, a multitude of parameter values in the literature making comparisons difficult, and no clear trends as a function of N. As such, its use as a benchmark should be approached with caution. NARMA is a non-linear memory task, so using the generic, more readily interpretable, albeit more computationally intensive, Information Processing Capacity (IPC) measure (section 3.8.1) can overcome these limitations.

Authors using this benchmark typically use it alongside other benchmarks, such as the Santa Fe LASER Task [60] (sec. 3.5.7), the Mackey–Glass System [53] (sec. 3.5.1) or the Multiple Superimposed Sines task [57] (sec. 3.5.2).

3.4.2 Channel Equalisation: Imitating Unknown Dynamics

The Channel Equalisation task is one that involves restoring a noisy signal to its original state. It is a task that is chosen for its real-world applicability.

Channel Equalisation can be modelled as an imitation task: the Reservoir Computer imitates a “perfect” filter, which completely removes the noise. This filter is unknown and may even be impossible, but as the output is known (the noiseless signal) and the training relies solely on the output, it can still be imitated to some degree. This task may be evaluated using a Symbol Error Rate [67].

Jaeger and Haas [67] first applied the task to Reservoir Computing. A description of the method used can be found in their supplementary materials [62]. The channel model used by Jaeger and Haas [67] was first introduced by Mathews and Lee [100].

The system takes as input a signal to which nonlinear noise has been added: the task is to output the signal with the noise removed. Jaeger [62] use a signal $d(n)$ that is a sequence of randomly chosen values from the set $\{-3, -1, 1, 3\}$. The signal

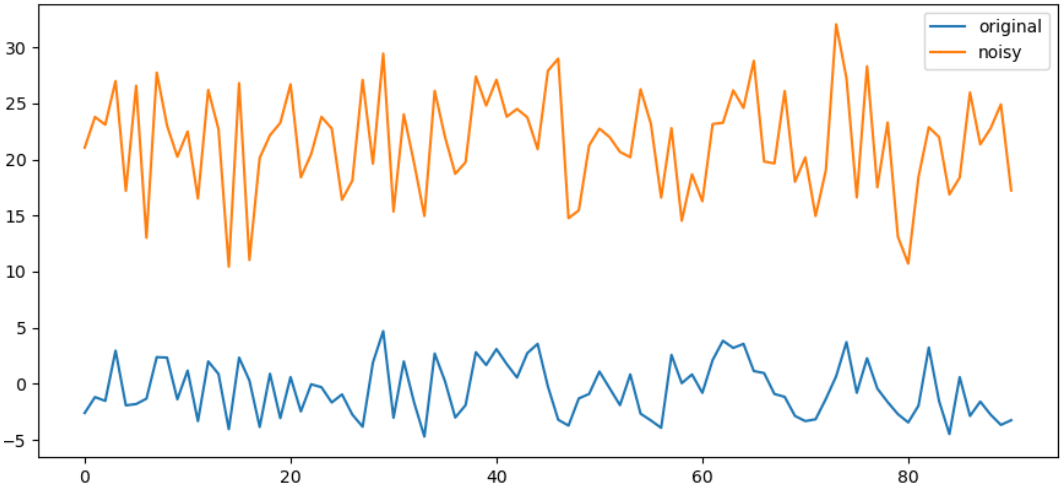


Figure 3.4: A channel signal (blue), and the signal with nonlinear noise applied (orange).

$d(n)$ is passed through a linear channel[1]:

$$\begin{aligned} q(n) = {} & 0.08d(n+2) - 0.12d(n+1) + \\ & d(n) + 0.18d(n-1) - 0.1d(n-2) + \\ & 0.09d(n-3) - 0.05d(n-4) + 0.04d(n-5) + \\ & 0.03d(n-6) + 0.01d(n-7) \end{aligned} \tag{3.4}$$

Then nonlinear noise applied:

$$u(n) = q(n) + 0.036q(n)^2 - 0.011q(n)^3 + v(n) \tag{3.5}$$

where $v(n)$ is Gaussian white noise, applied to ensure that the signal-to-noise ratio of the output is between 12 and 32 decibels [67].[2] An example of such a signal, as well as the signal with noisy applied, can be found in figure 3.4.

One can also find certain tasks in the literature that are called "Channel Equalisation", but do not follow the same methodology as the best known version of the benchmark. For example, Boccato et al. [6] uses a similar task, but the signal is composed solely of the values −1, +1, and the evaluation is done via Average Mean Squared Error. Boccato et al. [7] follows this up by adding an unsupervised channel

[1] In Jaeger [62] the coefficient is written as 0.09 1; however, both the original [100] and the "other formulation" [120] use 0.09, indicating that this may be a typographical error.

[2] Jaeger and Haas [67] specify 12 to 32 dB; the supplementary material [62] specifies 16 to 32 dB. Most authors who use the benchmark [3, 36, 112, 160] use the lower bound of 12 dB. There are some other formulations; for example Rodan and Tiňo [120] have no noise term, and add an offset of +30.

equalisation task, where the noiseless signal is not known. While this task has a similar purpose to the classic, supervised version of the task, the lack of "clean" signal used in training signifies that it is not an imitation task. This provides us with an interesting distinction between the form of the task (imitation of an unknown dynamical system, or not) and its function (channel equalisation, in both cases).

Unsupervised Channel Equalisation

An unsupervised version of the Channel Equalisation task exists [7]. The argument is that most distorted signals do not have a "clean" version that can be used to train the nonlinear filter, therefore it is useful to have a version of the task that uses unsupervised learning as opposed to imitation-based learning. Instead of training the received signal against the perfect, noiseless signal, the reservoir was trained against the information conveyed. The authors argue that, since the explicit signal itself was not used in training, this qualifies the task as an unsupervised learning task. As the 'perfect signal' is still known, however, the task is evaluated against that perfect signal.

3.4.3 Two-Jointed Arm: Imitating Unknown Dynamics

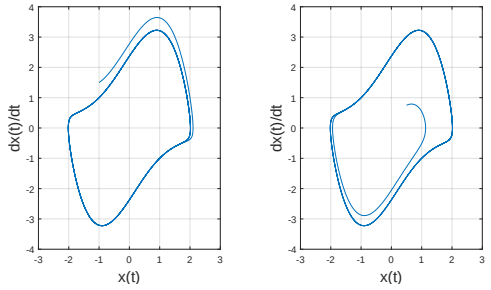
The two-jointed arm task [139] involves controlling a two-jointed (robotic) arm in order to move it from point A to point B. Like the Channel Equalisation task, this task can be seen as an imitation of an unknown, perfect controller, of which only the outputs are known.

Joshi and Maass [73, 74] use this task in the context of Reservoir Computing. The training data used are points along the trajectory that the robotic arm was intended to follow. The robotic arm is then tested in a "closed loop", where no training data is available, what we in section 3.3.1 refer to as "in-situ" testing.

The arm moves from point A to point B along a horizontal plane, allowing gravitational forces to be ignored. The dynamics of the arm are modelled by:

$$\mathbf{H}(\theta)\ddot{\theta} + \mathbf{C}(\theta, \dot{\theta})\dot{\theta} = \tau \tag{3.6}$$

where $\theta = [\theta_1, \theta_2]^T$ are the angles of the arm's joints, $\tau = [\tau_1, \tau_2]^T$ are the joint input torques, $\mathbf{H}$ is a 2×2 inertia matrix, and $\mathbf{C}$ is a 2×2 matrix of the Coriolis and centripetal



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

Figure 3.5: Phase plane of VdP oscillator in Equation 3.7 with *F(t)* = 0 (no external force applied). Parameter settings: μ = 1.5, the initial condition of each plot is given.

terms. The reservoir is then given a target trajectory, for which it must imitate a "perfect" controller in a model-free manner. See Joshi and Maass [73, 74] for details.

In this context, we see the advantage of testing the task *in situ*: the results show that, despite not following the same trajectory presented in the training data, the robotic arm still reaches its destination.

3.4.4 Van der Pol Oscillator: Imitating Unknown Dynamics of a Controller

The Van der Pol (VdP) oscillator was first described by the Dutch physicist Balthazar Van der Pol in the 1920s [116] and later employed to model the oscillations of the heart [151]. It is particularly useful for modelling systems that exhibit non-linear behaviour, such as relaxation oscillations and self-sustained oscillations. Within the context of machine learning, the VdP oscillator is commonly used to test the effectiveness of neural networks [110, 172]. Shougat and Perkins [135] apply the VdP oscillator as the substrate of physical reservoir computing.

The VdP oscillator is characterised by a nonlinear damping term, and the forced VdP with an external force $F(t)$, is expressed as a nonlinear differential equation:

$$\frac{d^2x}{dt^2} - \mu(1-x^2)\frac{dx}{dt} + x = F(t) \tag{3.7}$$

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. In the standard VdP system the external force term is zero $F(t) = 0$, in which case the phase plane trajectory of the VdP is as shown in Figure 3.5).

The VdP task can be set up as an imitation task, where a reservoir is trained to

imitate a PID controller. Once trained, the reservoir controller is tested in situ, where the output is compared to the target trajectory as opposed to the output of the PID controller.

In a recent study, the VdP oscillator is used as a model-free control task where a reservoir is trained to produce an external force to make the VdP system produce a circle [45, 46].

3.4.5 Pole balancing : Unsupervised Learning

Pole balancing is a family of benchmark tasks for controllers. It involves simulating a cart moving back and forth along a single axis, trying to keep one or several poles that are attached to the cart by a hinge upright. The task is considered a success if the poles remain upright for a certain amount of time.

In the field of Reservoir Computing, this task is primarily used to evaluate unsupervised learning in reservoirs. It has been used for exploring different methods of training reservoirs, such as Neuroevolution [71, 72] and NEAT [17]. One could also implement this task as an imitation task, and have the reservoir imitate the unknown dynamics of a perfect controller.

While there are multiple formulations of the pole balancing task, double pole balancing without velocity information is the most popular in this context. Given a cart that can move along the x axis, on which are mounted two poles by a hinge, the goal is to move the cart such that the poles stay balanced upright. The poles are of different lengths and masses.[3] At each timestep, the Reservoir Computer outputs a force to be applied to the cart. The task is considered a success if the poles stay upright for 100,000 steps, which is equivalent to roughly thirty minutes of simulation time.

3.5 Dynamics Prediction Tasks

In this section, we describe the systems commonly used as prediction benchmarks in Reservoir Computing. A greater variety of datasets that can be used for benchmark tasks can be found in Gilpin [50]'s database of chaotic dynamical systems, all

[3] An implementation of the pole balancing problem that is frequently used in the literature is that of Kenneth O. Stanley, and can be found at http://nn.cs.utexas.edu/?dpb-esp

specified according to guidelines proposed by Gebru et al. [48].

3.5.1 Mackey–Glass Equation : Known Dynamics

The Mackey–Glass benchmark is derived from a delay differential equation introduced to model certain physiological systems [102].[4] These models characterise illness through changes to the variables that lead to these systems turning chaotic. The authors examine various models, one for the breathing patterns of patients with Cheyne-Stokes illnesses, and two variants for the growth rates of white blood cells in leukaemia patients.

The latter model is used as the basis for the benchmark; Micheal C. Mackey [102, eqn.4b] state:

$$\dot{P} = \frac{\beta \theta^n P_{t-\tau}}{\theta^n + P_{t-\tau}^n} - \gamma P_t \tag{3.8}$$

where $P_t > 0$ is the concentration of circulating blood cells at time t; τ is the time delay between initiating blood cell production and the mature blood cells being released; $\beta > 0$ is the base level production rate of cells; $\theta > 0$ is the baseline concentration; $n > 0$ is a real-valued non-linearity parameter; $\gamma > 0$ is the decay rate of cells.

Normalising the concentration with respect to the baseline, $x_t = P_t/\theta$, gives the more usual form of the Mackey–Glass chaotic equation [51, eqn.1]:

$$\dot{x} = \frac{\beta x_{t-\tau}}{1 + x_{t-\tau}^n} - \gamma x_t \tag{3.9}$$

Here x is a dimensionless normalised quantity, expressed in 'units' of the parameter θ. θ is still a parameter of the model, but is now implicit in the equation: it provides the scale for the dependent variable x. Eqn. 3.9 has chaotic dynamics for $\beta = 2$, $n = 9.65$, $\gamma = 1$, $\tau = 2$ with an initial condition of $x = 0.5$ for $t < 0$ [51].[5]

Jaeger [63] states that the Mackey–Glass system is often used for learning dynamical systems from data, 'invariably' with parameter values of $\beta = 0.2, n = 10, \gamma = 0.1$, and that with these values the system is chaotic for $\tau > 16.8$; see [38] for a detailed analysis. Jaeger [63] investigates $\tau = 17$ (mildly chaotic) and $\tau = 30$

[4] This delay differential equation is also used elsewhere in Reservoir Computing to create a nonlinear node in delay line reservoirs (see, for example, Appeltant et al. [4]). This other usage is not examined here.

[5] The text of [51] states $\beta = 0.2, \gamma = 0.1, \tau = 2$, but the captions of its figures 1 and 2 state $\beta = 2, \gamma = 1, \tau = 2$. Figure 3.6 here favours the values from the captions. Equivalently, one could use $\beta = 0.2, \gamma = 0.1, \tau = 20$.

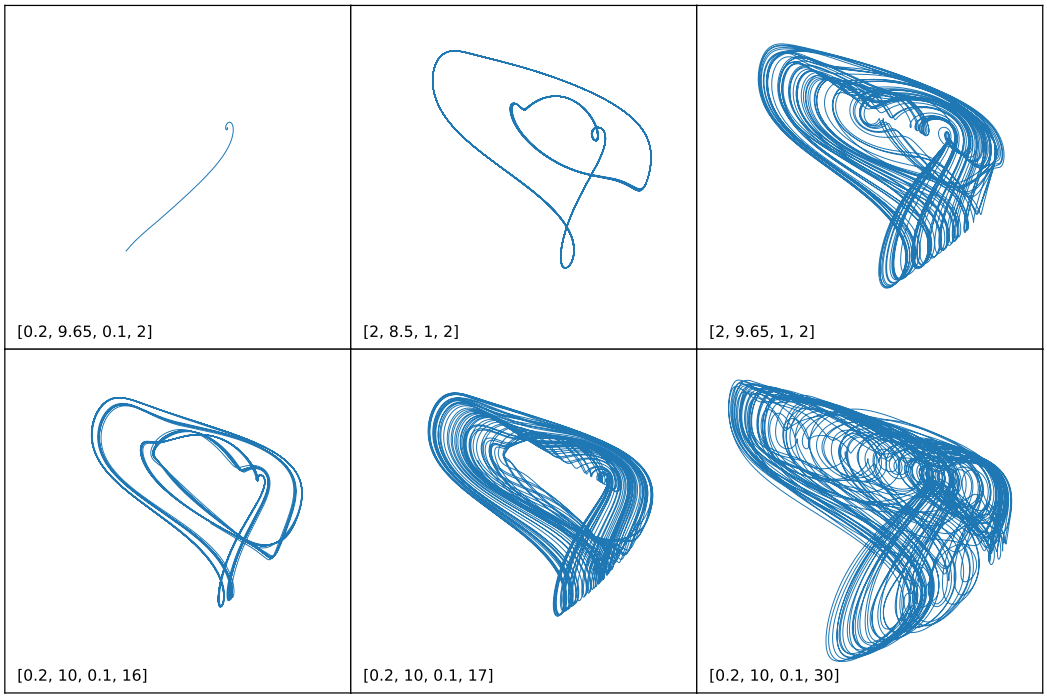


Figure 3.6: Time delay embedding plots of the Mackey–Glass equation, for various parameter values. Each figure is a plot of $x(t-\tau)$ against $x(t)$ for a particular set of parameter values $[\beta, n, \gamma, \tau]$. The x and y axes of all plots run from 0.1 to 1.5. Plotted for 150τ time units after the transient, and an initial condition $x(t<0)=0.5$. The transient of the first 100 time units are not plotted (except for the top left plot, where the entire transient is shown).
Top: $[\beta, n, \gamma, \tau]$ values from Glass and Mackey [51]:
$[0.2, 9.65, 0.1, 2]$: transient to a point attractor
$[2, 8.5, 1, 2]$: periodic behaviour [their fig.5]
$[2, 9.65, 1, 2]$: chaotic behaviour [their fig.2,fig.7]
Bottom: $[\beta, n, \gamma, \tau]$ values from Jaeger [63]:
$[0.2, 10, 0.1, 16]$: periodic behaviour
$[0.2, 10, 0.1, 17]$: mildly chaotic behaviour
$[0.2, 10, 0.1, 30]$: wildly chaotic behaviour
(Plots produced using the python ddeint delay differential equation solver, https://pypi.org/project/ddeint/ with an integration timestep of 0.02 time units.)

(which he dubs 'wildly' chaotic). See figure 3.6.

The parameter values used by Glass and Mackey [51] and by Jaeger [63] appear to differ considerably (figure 3.6); however, timescale rescaling shows they are comparable. There are three timescales in eqn. 3.9: $1/\beta$ (production, or growth, timescale); $1/\gamma$ (decay timescale), and τ (time delay). If we normalise these with respect to the decay timescale, we have two independent timescales: γ/β and $\gamma\tau$. Glass and Mackey [51] use $\gamma/\beta = 1/2$, $\gamma\tau = 2$; Jaeger [63] uses $\gamma/\beta = 1/2$, $\gamma\tau = 1.7, 3.0$.

Glass and Mackey [51] explore the transition between periodic and chaotic behaviour using fixed timescales and varying the non-linearity parameter n (the timescales have biological meaning whereas the non-linearity is a parameter of the model); Jaeger [63] fixes the non-linearity at $n = 10$ and instead varies the delay feedback timescale τ (the model is being used in a more abstract manner, and varying the time delay is a standard approach to investigating chaotic behaviours in DDEs [111]).

Eqn. 3.9 describes a continuous system. Jaeger [63] describes a discretisation process for converting it to a discrete system suitable for generating time series data.

- Discretise eqn. 3.9 using $dx/dt \approx (x(t+\Delta t) - x(t))/\Delta t$:

$$\hat{x}(t+\Delta t) = \hat{x}(t) + \Delta t \left(\frac{\beta \hat{x}(t-\tau)}{1+\hat{x}^n(t-\tau)} - \gamma \hat{x}(t) \right) \tag{3.10}$$

- Take $1/\Delta t = N$, an integer (Jaeger [63] uses $N = 10$), to produce the time series $\hat{x}(t), \hat{x}(t+\Delta t), \hat{x}(t+2\Delta t), \ldots \hat{x}(t+(N-1)\Delta t), \hat{x}(t+1), \ldots$

- Subsample the full series every N steps to produce the reduced series $\hat{x}(t), \hat{x}(t+1), \hat{x}(t+2), \ldots$

- Finally, transform the individual values to the interval $[-1, 1]$:

$$y(t) = \tanh(\hat{x}(t) - 1) \tag{3.11}$$

The time series $y(t), y(t+1), \ldots$ provides the training data for the benchmark.

Using this discretisation method, Jaeger [63] generates two sequences for each value of $\tau = 17$ and 30, one sequence of length 3,000 and of 21,000 (four sequences in

source	no. of test seqs	seq length	τ value	washout length	test value	no. of runs
Jaeger [63]	1	3000	30	1000	84	50
—"—	1	21000	30	1000	84	50
—"—	1	3000	17	1000	84	20
—"—	1	21000	17	1000	84	20
Jaeger and Haas [67]	1	3000	?	1000	84	100
Holzmann and Hauser [57]	2	3000	17, 30	1000	84	100
—"—	2	21000	17, 30	1000	120	100
Roeschies and Igel [121]	1	3000	17	100	84	50
Goudarzi et al. [53]	1	2000	?	?	2000	10
Moon et al. [105]	1	500	18	?	50–52	?

Table 3.3: A summary of different parameters used in the literature for the Mackey–Glass prediction benchmark. Values marked with "?" are not reported in the cited work. Goudarzi et al. [53] use NMSE rather than NMSRE.

total). These form the combined washout and training sequences. The ESN is trained on this data, then tested on its ability to predict the 84th output value, compared to the expected 84th value, computing the relevant Normalised Mean Squared Error (NRMSE_{84}) of the value over multiple runs.

Mackey–Glass is used as a benchmark by Jaeger again in [67], this time with a single 3,000-step training sequence, and the NRMSE of the 84th subsequent value taken over 100 runs. The τ values used are not specified. Holzmann and Hauser [57] also use the system as a benchmark, using a method almost identical to [63], though with the NRMSE taken over 100 runs, and an additional NRMSE of the 120th value used for the training sequence of length 21,000. Roeschies and Igel [121] base their method on [63], this time using a single sequence of length 3,000 with $\tau = 17$. One other notable difference is the use of a washout of 100 values, whereas prior works have used washouts of 1,000. Goudarzi, Shabani, and Stefanovic [53] and Moon et al. [105] depart from this traditional setup in order to propose their own. These various different uses are detailed in table 3.3.

This benchmark is an ideal one to use in prediction of known systems, not only because its past use is prevalent and well documented, but also because, as a chaotic system, it is a rather challenging task, particularly when certain parameter values are set, such as $\tau \geq 16.8$. The use of the NMSRE over a given number of runs on a single value is unusual, however, and should be highlighted to avoid confusion.

3.5.2 Multiple Superimposed Oscillators : Known Dynamics

The Multiple Superimposed Oscillator task, sometimes also referred to as the Multiple Superimposed Sines, or the Multiple Sines task, is a known dynamical system prediction benchmark.

The object of the task is to imitate a sequence generated by a sum of sines with incommensurate frequencies, and hence with a very long overall period, particularly for large n:

$$x(t) = \sum_{i=1}^{n} \sin(\alpha_i t) \tag{3.12}$$

where the frequencies $\alpha \in [0.2, 0.311, 0.42, 0.51, 0.63, 0.74, 0.85, 0.97]$.

The task was originally introduced in a presentation by Jaeger [64], for $n = 2$ (what would now be called MSO-2, but there called "additive dynamics"), as an example of a task that an ESN finds "impossible to learn". Wierstra, Gomez, and Schmidhuber [165] increase task difficulty by extending the list of α with higher frequency values to give MSO-5, and Roeschies and Igel [122] further extend the α values to give MSO-8. Other authors who used this benchmark include Xue, Yang, and Haykin [171] (MSO-2) and Koryakin, Lohmann, and Butz [76] (MSO-8).

Since Jaeger [64] states that the task cannot be solved by the standard ESN model, some authors use this benchmark on extensions to the ESN model [76, 128, 165, 171], for example, by evolving the weights, or using feedback, to demonstrate superior performance.

However, if one uses just the standard ESN model with similar training and testing dataset lengths as these variants (many hundreds of timesteps each), one can also achieve apparently reasonable NRMSE values (see section 3.9.3), implying that the ESN has learned the data. Jaeger et al. [69, fig.4] demonstrate that this good prediction does not hold indefinitely; after many *thousands* of timesteps, the prediction diverges. This is what Jaeger [64] means when referring to the task as "impossible to learn".

3.5.3 Lazy Figure 8 : Known Dynamics

The figure 8 task is a "perennial task" in the field of Recurrent Neural Networks [69], and a challenging one.

It involves predicting the next point along a sequence that traces the shape of

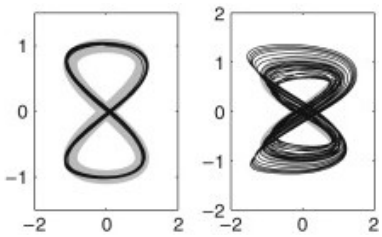


Figure 3.7: Lazy Figure 8 [69, fig.6]. This figure follows a trajectory of 200 points, which is then repeated over a training and subsequent testing period. The two figures are of the output from the ESNs, with (left) constant and (right) "time-warped" inputs.

the digit 8, and as such is a prediction of a known system.

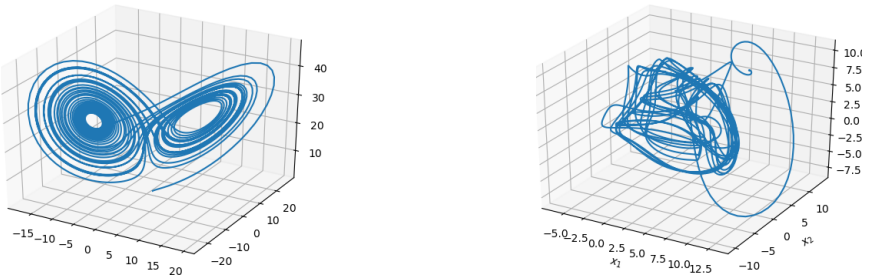
In Reservoir Computing, this task is adapted to the "Lazy[6] Figure 8" [69], where the full shape is composed of 200 points. Training is performed over 3000 steps, which includes a washout period of 1000 steps. Two experiments were performed, the first where the 200 points were equally spaced over the figure, and a second "time-warped" where the spacing of the points varied over time.

Like the MSO task, this benchmark is one that cannot be satisfactorily performed by a classical ESN, although Jaeger et al. [69] show that their leaky–integrator based model is capable of accomplishing it. This is an example of a benchmark being used to show that a newer model performs better than classical ESNs.

3.5.4 Lorenz chaos: Known Dynamics

Lorenz chaos refers to two systems of equations initially derived to model the behaviour of certain weather-based dynamical systems. The task is to predict the behaviour of the equational model, and the benchmark is thus a prediction of a known system. These systems have chaotic behaviour, where small differences in state can lead to large divergences in behaviours. This makes predicting it an interesting task in Reservoir Computing.

[6] While the "Lazy" in "Lazy Eight" typically refers to the orientation of the figure 8 drawn on its side (∞), Jaeger instead calls it thus due to the timescale over which the figure is drawn.



(a) Lorenz'63 model (eqn. 3.13) with $\sigma = 10$, $\rho = 28$, $\beta = 8/3$.

(b) Projection of a Lorenz'96 model (eqn. 3.14) with $N = 5$, $F = 8$

Figure 3.8: Illustrations of the two Lorenz systems.

Lorenz'63 System

The first system, introduced in 1963 [84, eqn.25–27], is frequently referred to as Lorenz'63. The set of three coupled ODEs are:

$$\begin{aligned} \dot{x} &= \sigma(x - y) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z \end{aligned} \tag{3.13}$$

The values of the parameters σ, ρ, and β change behaviour between periodic and chaotic. The parameter values used are typically $\sigma = 10$, $\rho = 28$, $\beta = 8/3$ [15, 113, 121]. The behaviour for this chaotic choice is illustrated in figure 3.8a.

Lorenz'96 System

The second system, introduced in 1996 [83, eqn.1], is typically referred to as Lorenz'96. It is described by N variables x_i, $i \in 0 \ldots N-1$, governed by N coupled ODEs:

$$\dot{x}_i = (x_{i+1} - x_{i-2})x_{i-1} - x_i + F \tag{3.14}$$

where the index arithmetic is modulo N, and the constant F is independent of i. For small F, the solution is $x_i = F$ for all i; for intermediate values of F there are periodic solutions; for larger values of F solutions are chaotic. (The value of F that marks the transition to chaos depends on N.) The behaviour for one choice of N and F is shown in figure 3.8b. Table 3.4 gives values of N and F used in the literature.

source	N	F
Vlachas et al. [161]	0–40	4–16
Vlachas et al. [162]	0–40	8, 10
Canaday, Pomerance, and Gauthier [15]	11	8

Table 3.4: Parameter values used when generating a Lorenz'96 sequence.

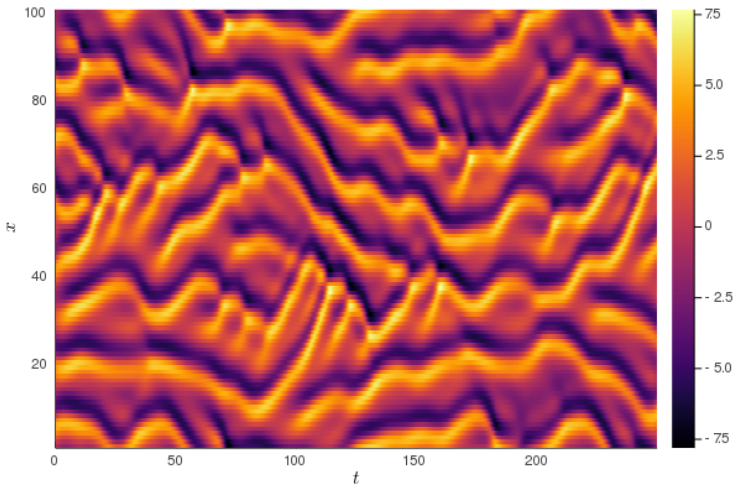


Figure 3.9: A space-time plot of a solution of the KS equation for $L = 100$, $0 < t < 250$. (From https://commons.wikimedia.org/wiki/File:KuramotofiSivashinsky_spatiotemporal_evolution.png, provided with CC0 licence.)

3.5.5 Kuramoto–Sivashinski Equation: Known Dynamics

The Kuramoto–Sivashinski (KS) equation is a fourth order PDE originally derived to model diffusion-induced chemical turbulence [137, 138]. When used as a benchmark, the task is to predict the next output of the system, and therefore is a prediction task for a known system.

Like the Lorenz equation, the KS equation can describe a chaotic system. As such, it is a difficult system to predict. A formulation of the equation in one spatial dimension, which is used in a prediction task, is:

$$y_t = -yy_x - y_{xx} - y_{xxxx} \tag{3.15}$$

where $y_t = \partial y/\partial t$ and $y_x = \partial y/\partial x$. The initial condition (at $t = 0$) is usually defined on a finite x domain ($0 \leq x < L$) with periodic boundary conditions. The time behaviour is periodic for small L; as L is increased, the periodic behaviour undergoes bifurcations, and eventually becoming chaotic.

The KS equation has been used in the context of Reservoir Computing. Initially it was used as an actual task [113]: here the Reservoir Computer's task is to predict the next output of the equation, to test if the Reservoir Computer can imitate the Lyapunov exponents of the equation, whether or not divergence takes place. The K–S equation is used to find out whether Reservoir Computing could solve the observer problem, which involves deducing the full state of a dynamical system based on partial information of the state and of the system dynamics [86].

The KS equation has also been used as a prediction benchmark. Vlachas et al. [162] uses it with a spatial size of $L = 200$; Mushegh Rafayelyan, Jonathan Dong, Yongqi Tan, Florent Krzakala, Sylvain Gigan [106] use $L \in [12, 22, 36, 60, 100]$.

Pathak et al. [114] also compare against a modified form, to investigate the effect of spatial inhomogeneity:

$$y_t = -yy_x - y_{xx} - y_{xxxx} + \mu \cos(2\pi x/\lambda) \tag{3.16}$$

The size L is an integer multiple of the wavelength λ, with $L \in [100, 200, 400, 800, 1600]$.

3.5.6 Sunspot Numbers : Unknown Dynamics

Sunspots are dark spots that appear on the sun on a temporary basis. Sunspots have been recorded consistently since 1610 [109]. As such, the observations can be used as a dataset for the prediction of the behaviour of an unknown dynamical system.

Predicting the next value in a dataset of sunspots is a perennial task in Machine Learning, first used in the early twentieth century [174]. The task varies greatly in the details: while each involves predicting the next value in the dataset, which dataset is used, how the values are calculated, and what preprocessing is applied, all vary across the literature. The only consistency appears to be that the datasets typically start in 1749, although there remains some inconsistency on whether the first value should be taken in January [141] or July [118, 132].

At least four different sources for "Sunspot Numbers" are used in the literature, as shown in table 3.5; the Zurich dataset is shown in figure 3.10. These have differing ways of dealing with missing data. In the NGDC dataset, a value of -99 indicated missing data; other datasets appear to have removed this data in preprocessing.

dataset name	source	used in
NOAA dataset	NGDC [109]	Schwenker and Labib [132]
Zurich Monthly sunspot numbers	[9]	Stepney [141]
NASA Greenwich sunspot numbers	[108]	Dale [24]
Carrington sunspot numbers	[117]	Shougat et al. [136]

Table 3.5: Sources found in the literature for the sunspot numbers, and an example citation of the source's use in the literature as the "Sunspot Prediction" benchmark.

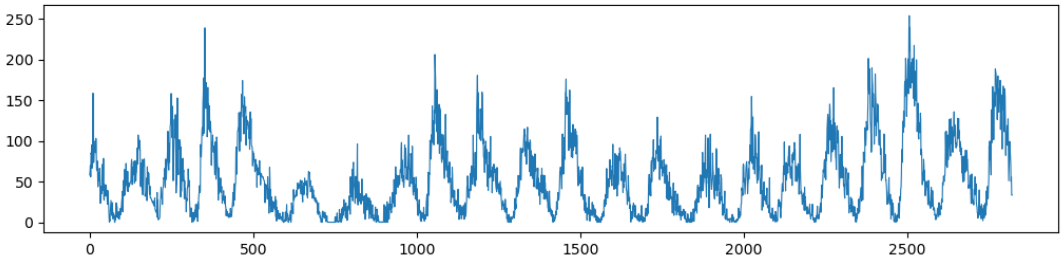


Figure 3.10: The Zurich Monthly Sunspots Numbers

Some authors who use the NGDC dataset refer to preprocessing to deal with missing data, but do not define how this is performed [132].

As well as the existence of differing datasets, other problems with this task are found. There is the inconsistency of the time period used: some use months [141], others days [136]. There does not seem to be any specification on how monthly averages are computed, and whether or how they account for the different lengths of months. There is also the impossibility of making perfect observations: issues may arise from cloud cover leading to missing data, quality of telescopes, or difficulty deciding on how to count sunspots. All these render this data unreliable as data about a dynamical system, and make comparison between different systems problematic.

That said, finding out whether a data-driven model driven by imperfect data can produce usable results is a potentially valuable problem; all data is, after all, imperfect: data is merely a representation of the real state, and like all representation, some details are abstracted away. Having a dataset where some of the limitations are known and acknowledged thus has its own value; but if this is the case, it is important for users of the dataset to acknowledge this.

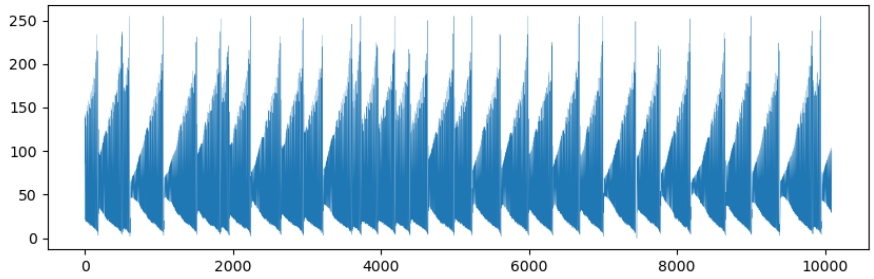


Figure 3.11: The Santa Fe LASER dataset.

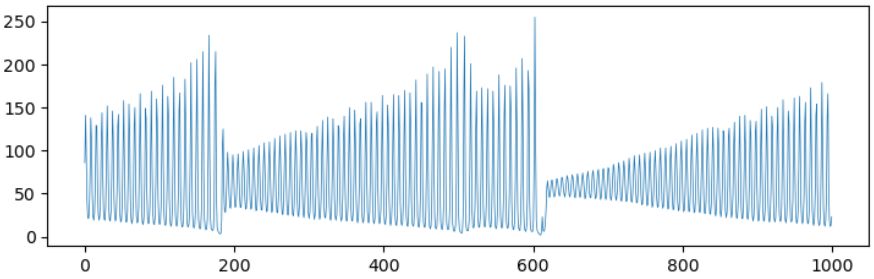


Figure 3.12: Zoom in on the first 1000 values of the Santa Fe LASER dataset.

3.5.7 Santa Fe LASER Dataset: Unknown Dynamics

The Santa Fe LASER dataset is a dataset originally produced in an experiment trying to replicate Lorenz-like chaos in NH3–FIR LASERs [58]. It was initially distributed as one of the datasets in the Santa Fe 1992 Time Series prediction competition, the proceedings of which were published in 1994 [164]. The dataset is now frequently used as a prediction task for an unknown system.

The original dataset was only 1000 data points long, the shortest of the datasets distributed in the competition. It was also noted for being stationary, low dimensional, clean, scalar-based, and nonlinear. A particularity of the dataset is that the data is characterised by catastrophes: the values grow in a somewhat predictable manner, until one of these catastrophes take place and the values change drastically. See figures 3.11 and 3.12.

The most readily available version of the dataset, with the original 1000 values, can be found at Salles [125][7]. The amount of data used varies in the literature, and is occasionally not recorded, as we can see in the examples listed in table 3.6.

The different experimental setups may lead to issues in interpreting results, as

[7] the full dataset is available on GitHub: https://github.com/MaterialMan/CHARC/blob/master/Support%20files/other/Datasets/laser.txt

source	total length	training	testing
Larger et al. [81]	not recorded	not recorded	not recorded
Brunner et al. [10]	4000	3200	800
Dale [24]	5000	25000	1250 for validating individual reservoirs, 1250 more for evaluating evolved reservoirs
Guo et al. [54]	4000	3000	1000
Shougat et al. [136]	not recorded	not recorded	not recorded

Table 3.6: The parameter values used in the literature for the Santa Fe LASER benchmark.

they may be unaccounted for independent variables; in such cases, it becomes difficult to distinguish where the variation in results comes from.

Some of this variation, such as the topology of the reservoir when this is randomly selected, may be smoothed out by repeating the experiment enough times. However, differing experimental setups may lead to systematic errors that persist over each experiment in a publication.

The Santa Fe 1992 Time Series Prediction competition also distributed five other datasets. These are not used as benchmarks, except for the sleep apnea dataset (see section 3.5.8). It is generally safe to assume that any work referring to the "Santa Fe benchmark" with no further elaboration is using the LASER dataset.

3.5.8 Santa Fe Sleep Apnea Dataset: Unknown Dynamics

The sleep apnea benchmark is physiological data collected from a patient with suspected sleep apnea, first distributed in 1991 in the Santa Fe Time Series Prediction Competition [163]. The data used here [59, 126] is provided via PhysioNet [52]. The data was measured over 4 hours and 43 minutes, and digitised at 250 Hz, and sampled every 0.5 seconds. The data documents the patient's (i) heart rate (ii) respiration rate (iii) blood oxygen saturation. Additional information on the patient's sleep stage is provided in supplementary material.

The authors of the competition describe several peculiarities of the data that makes time series prediction particularly complicated, and not an accurate depiction of a physiological system. Here, we are not attempting to simulate human physiology. Instead, we are studying whether, by studying three related time series in tandem, we can achieve better results than we can by studying them individually.

While this dataset has a number of attributes that make it interesting to study from

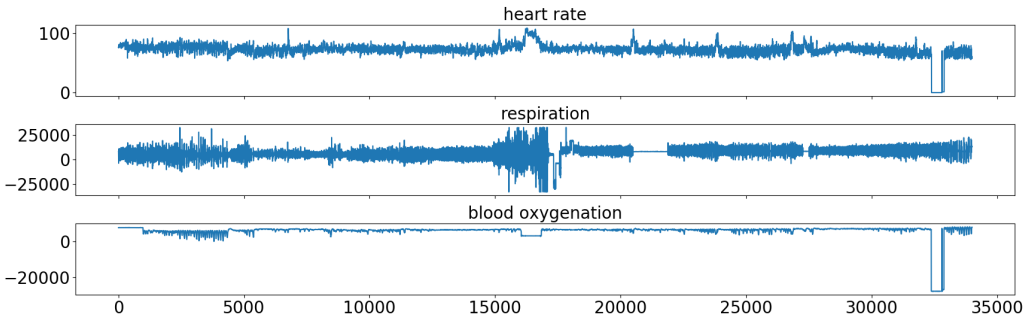


Figure 3.13: Full sleep apnea dataset.

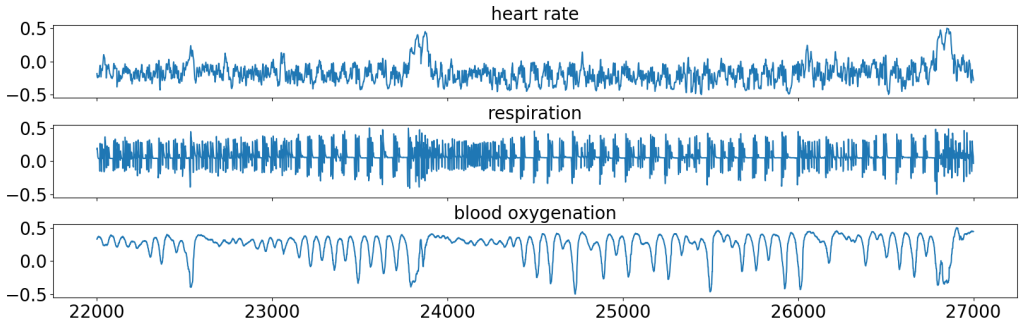


Figure 3.14: Portion of the sleep apnea dataset, as used in Wringe, Stepney, and Trefzer [169]; each dataset portion is separately normalised to the range $[-0.5, 0.5]$ using equation 3.17.

a Reservoir Computing perspective, we only find one work that uses it a reservoir computing benchmark [136]. That work uses each dataset independently as an individual benchmark. In chapter 7, we study it as a multi-input benchmark.

Shougat et al. [136] state that the input is normalised before being input to the reservoir, but it is unclear how this normalisation is performed. In chapter 7, we normalise the inputs to fall between $[-0.5, 0.5]$ using:

$$x' = \frac{x - \min(x)}{\text{range}(x)} - 0.5 \tag{3.17}$$

The data lengths used by Shougat et al. [136] are not stated. We start our input at point 22000 (fig. 3.14), and use 1000 points as washout, 3000 points as training, and 1000 points as testing. These values are chosen to ensure both the training and testing phases include both normal sleep and apnea periods.

We propose that this dataset can also be used as a stationary classification benchmark, and discuss this in section 3.7.4.

3.6 Computation Tasks

3.6.1 the XOR Task : Binary Output

The XOR task is a family of tasks based on the bitwise operation XOR, and entered the Reservoir Computing field through the field of Neural Networks. Unlike many of the benchmarks discussed here, there is no single XOR task, nor consensus on what the goal of the task is. Here we define it as a single-output classification task, and describe various variants, to demonstrate the ways the task can be used.

An XOR task is used early on in Reservoir Computing, where a bucket of water is shown to support a simple Liquid State Machines [39]. The task to output the XOR of the two simultaneously applied inputs. Much post-processing is needed to read the output. The task of outputting the XOR of two consecutively supplied input bits is also used [80, 136].

The XOR task has been adapted to measure the memory of a Reservoir Computer as well as the nonlinearity [158]: the task is to output the XOR of two initial values after a sizeable delay. A similar task has been used to benchmark the memory of an ESN [66].

3.6.2 Parity Task : Binary Output

The parity task extends the XOR task to multiple inputs: its task is to distinguish an odd from even number of 'high' inputs, to compute the parity.

Here we designate this a computation task: it computes the parity. It can also be considered a temporal classification task, or a known dynamical system imitation task, depending on one's viewpoint.

Given a stream of inputs $u(t)$, the output of the PARITY-n task is given by the parity of n consecutive bits, read out after a delay τ [56]:

$$y(t) = PARITY(u(t-\tau), u(t-\tau-1), ...u(t-\tau-n)) \tag{3.18}$$

The input $u(t)$ is randomly sampled from $[-1, 1]$ [32] or $[0, 1]$[56].

The parity task has been used in reservoir computing, with a range of different n values (see table 3.7).

source	n
Haykin et al. [56]	3
Schrauwen, Buesing, and Legenstein [130]	5
Dion, Mejaouri, and Sylvestre [32]	$2-5$
Shougat et al. [136]	$4,6$

Table 3.7: Some of the different parameter values used for the RC parity benchmark.

3.7 Classification Tasks

3.7.1 MNIST Handwritten Digits : Static Classification

The handwritten digits benchmark is a benchmark commonly used to identify handwriting, and is thus a multiple-output classification task.

The MNIST Handwritten Digits Database [103] is a database of digits between 0 and 9 handwritten by highschool students and United States Census Bureau employees. The database consists of multiple datasets, most notable among which is the training dataset, composed of 60,000 images created by 250 writers. While originally black and white, the images were normalised to fit into 20 by 20 pixel arrays, and subsequently centered in a 28 by 28 pixel image, with the resulting images being in grayscale.

This task has been used as a Reservoir Computing benchmark. [35] describe a preprocessing technique that converts the image of the digit into a black-and-white 22×20 pixel image. Each row is then converted into a time series input fed into the reservoir. In order to make the task easier for the reservoir, it takes two inputs, sampled at different rates. It is unclear here how the data is fed in to the reservoir.

Another use of this database when benchmarking Reservoir Computing involves turning each image into a time series [127], in this case feeding the image to the reservoir a single pixel at a time. Another variation is to permute the pixels, in order to remove some of the internal structures of the image. This variation is called the Permuted Sequential MNIST task (psMNIST) [97, 98].

3.7.2 Spoken Digits : Non-Stationary Classification

Identifying words that have been spoken is a classic task in neural networks, and is a multiple output category benchmark used in Reservoir Computing almost since its

inception. The goal of the task is to identify some words being spoken, whoever the speaker. The most common version of it uses the TI-46 dataset.

The TI-46 Isolated Spoken Words Dataset [49] is a dataset of 20 individual words, spoken by 8 men and 8 women. It contains ten digits, and ten words common in speech recognition, such as “help” and “stop”. The dataset as available has 26 utterances per female speaker of each word digit, 10 for training and 16 for testing. The intent of the project was to have a dataset of words that personal assistant-style technology could be tested on, particularly ones suited to office environments, rather than home ones. The dataset can be found online [147], but is not open access.

A subset of the TI-46 Dataset consisting of the digits from “zero” to “nine” said by five different speakers was first used in the context of Reservoir Computing to the speech recognition capabilities of Liquid State Machines to the then state-of-the-art Hidden Markov Models [156]. This work was expanded to include ESNs [155]. The authors detail the experimental setup, which we summarise here:

- Noise was added to the words, using the NOISEX database of Varga and Steeneken [154].
- The words were first preprocessed using the Lyon Passive Ear Model [90], a more biologically inspired model for preprocessing than the one performed for Hidden Markov Models. (Some other authors preprocess the data differently.)
- The words were then encoded into spike trains using the Bens Spiker Algorithm (BSA) [129]
- The evaluation was performed using the Word Error Rate (WER), calculated as $(N_{\text{correct}}/N_{\text{total}})$

Verstraeten, Schrauwen, and Stroobandt [155] added a degree of confidence measure to the output, which increased the accuracy of their results.

The Isolated Spoken Digits task is by far the most popular classification benchmark, particularly after its first use. See, for example, Antoine Dejonkheere, Francois Duport, Anteo Smerieri, Li Fang, Jean-Louis Oudar, Marc Haelterman, Serge Massar [3], Appeltant et al. [4], Brunner et al. [10], Butcher et al. [12], Dion, Mejaouri, and Sylvestre [32], Duport et al. [36], Larger et al. [81], Moon et al. [105], Paquot et al. [112], Soriano et al. [140], Verstraeten et al. [157], and Vinckier et al. [160].

A simple version of this benchmark was used in one of the earliest reservoir computing publications [39], getting a Reservoir Computer to recognise the spoken digits “0” and “1”, with these digits being input as recorded speech samples.

Isolated spoken digits used as a standard benchmark, as described here, is separate from the larger problem of general speech recognition, which attempts to identify words and phonemes from larger datasets [149].

3.7.3 Japanese Vowels : Non-Stationary Classification

Another speech recognition task uses a dataset[8] of vowels spoken by nine Japanese men [69]. Unlike the spoken digits task, the task is not to categorise by what has been spoken, but rather to categorise by speaker. Hence it remains a multiple-output classification task. In the original paper, 4 experimental setups were tested, the most successful of which we summarise here.

The experiment used 9 output nodes, one for each speaker, and had a training length of 270 samples.

The inputs were processed by first being partitioned into $D = 3$ subsequences of equal length; these subsequences were then joined into single input vectors of size D. The outputs were then trained using linear regression. Because overfitting was a concern, the reservoir size was kept to $N = 4$.

While this task is not frequently used, it is interesting to contrast its goals to the more popular Spoken Digits Recognition task.

3.7.4 Santa Fe Sleep Apnea Dataset: Stationary classification

The Santa Fe sleep Apnea dataset (discussed in section 7.3) is a series of readings from a patient with sleep apnea. During the 4 hours and 43 minutes during which the readings are taken, the patient experiences both normal sleep and periods of apnea.

We propose that this may be used as a stationary temporal classification task, as it could potentially be used to classify normal *v* apnea periods. However, in the literature, the dataset tends to be used for a time series prediction task, either predicting each datastream individually, or using a combination to improve prediction.

[8] Jaeger et al. [69] give two links where the dataset can be found. Those links no longer work; the dataset can be found at https://github.com/MaterialMan/CHARC/tree/master/Support%20files/other/Datasets/JapaneseVowels.

For example, Shougat et al. [136], who treat each of the datasets as entirely distinct benchmark tasks, while in chapter 7, we study how the datasets interact with each other.

3.7.5 McMaster IPIX Radar Data : Unknown Dynamics

The McMaster IPIX Radar is an X-band radar, originally created to detect icebergs, but capable of collecting data on all sea clutter. The website [101] indicates that the movement of sea clutter is influenced by several dynamical systems, and is itself a nonlinear dynamical system. As such, it makes an ideal prediction benchmark for Reservoir Computing.

The radar's website makes two databases available for general use [101], one created from data recorded in Dartmouth, Nova Scotia, in 1993, and the other from data collected in Grimsby, Ontario, in 1998. The latter database is marked as incomplete by the radar's website, and is encouraged to be used only comparatively with the Dartmouth data.

This data has been used as a prediction task in Reservoir Computing [171]. It is unclear exactly which of the datasets are used, or how these are provided as input for the Reservoir Computer. 2000 datapoints are used, of which 200 are used as washout data, 800 for training, and 1000 for testing, but no more information is given.

Subsequent uses of this dataset in Reservoir Computing [3, 16, 36, 118, 120] appear to be consistent with the first use, citing [171] and using the same dataset sizes.

3.8 Direct Property Measures

3.8.1 Memory Capacity

Linear Memory Capacity

Certain tasks are interesting because they directly reveal properties of a given Reservoir Computer. One of these is the Linear Memory Capacity task, which quantifies the fading memory of the reservoir. This was introduced by Jaeger [61], and has been investigated in the context of reservoir computing and used as a benchmark task [36, 53, 120, 140, 157].

The Linear Memory Capacity of a given Reservoir Computer that takes scalar input is defined as follows. Consider an input stream $u(t) \in U[-1,1]$. Train the reservoir to reproduce this input, delayed by a number of timesteps k: the target output is $\hat{v} = u(t-k)$, the observed output is $v_k(t)$, the linear memory capacity for this delay is defined as the covariance squared of the delayed input (target output) and the observed output, normalised by the variances of the input and the observed output:

$$MC_k = \frac{cov^2\left(u(t-k), v_k(t)\right)}{var(u(t))var(v_k(t))} \tag{3.19}$$

The total Linear Memory Capacity is the sum over all delays:

$$MC = \sum_{k=1}^{\infty} MC_k \tag{3.20}$$

For higher k, each MC_k tends to decrease (and the corresponding validation NRMSE tends to increase), which means that inputs after longer delays are remembered less well, and are dominated by noise. In the formal definition of MC (eqn. 3.20), the sum over delays goes to infinity; however the small values at high k are essentially noise, and should be neglected, so in practice a cutoff is used. Dambre et al. [30, SupMat3.2] define a threshold based on the size of the reported capacity. Dale [24] uses a cutoff of $k_{max} = 2N$ (number of nodes).

All the values of MC_k can be found in a single run: instead of training a target scalar output for a single k using $\hat{v} = u(t-k)$, train for all k up to the relevant threshold, using target vector output $\hat{\mathbf{v}}(t) = (u(t-1), u(t-2), \ldots, u(t-k_{\max}))^T$.

Nonlinear Memory Capacity

While the Linear Memory Capacity measure gives a useful quantification of a reservoir's memory capacity, it cannot on its own model the full computing power of dynamical systems. [30] generalise the definition of linear memory capacity in a way that allows them to define various non-linear capacities, too: they call this *information processing capacity* (IPC). These measure a Reservoir Computer's ability to compute a nonlinear function of past inputs, for example, a cubic function of delayed inputs, such as $u^3(t-1)$ or $u(t-1)u^2(t-2)$. The cases where the polynomials involve different time delays are called cross memory capacities [36].

Duport et al. [36] consider quadratic polynomials. Dambre et al. [30] consider

complete sets of orthogonal polynomials, to cover all possible non-linear capacities and cross memory capacities with no double counting. They use normalised Legendre polynomials for inputs $u(t) \in [-1, 1]$. They note that different sets of orthogonal polynomials should be used for different input distributions, for example, Hermite polynomials for Gaussian-distributed inputs. Orthogonal sets of trigonometric functions can also be used as the basis.

As for linear memory capacity, the reservoir is trained to output the relevant (here, polynomial) function of its delayed input. For a given degree d of polynomial (linear, quadratic, cubic, etc), the contributions for all delays k, including all combinations of delays in the cross memory capacities, are summed. Then the total non-linear memory capacity, or IPC, is the sum over all polynomial degrees. Dambre et al. [30] prove that the total IPC (from the contributions of all the linear and non-linear polynomials) is $MC = N$.

This process is computationally intensive, as there are many combinations of polynomials and delays as degree d increases. Contributions at high d decrease, and a cutoff at large d is used, and the same observed state data can be used in all the training.

NARMA as a Memory Capacity Proxy

The NARMA benchmark (sec. 3.4.1) also measures how well a reservoir can reconstruct a non-linear polynomial function of delayed inputs. Despite the benchmark's limitations (as discussed in sec. 3.4.1), the ability to successfully learn NARMA-N can be used as a proxy for saying the reservoir has a memory capacity of N.

3.8.2 Rank-Based Measures

There are some task-independent measures that involve calculating the rank of a matrix constructed from multiple observations of the reservoir's state over time, when driven with random input. The different rank measures depend on the form of the input and the state measurement points.

Büsing, Schrauwen, and Legenstein [11] and Legenstein and Maass [82] introduce two particular rank-based measures: kernel quality (kernel rank, KR), and generalisation rank (GR). These have been adapted for use with ESNs in two different ways in

Algorithm 1 KR, Vidamour et al. [159]

S := number of input streams $\geq N$
T := length of each input stream (timepoints)
run reservoir with washout stream
for $i \in 1..S$ **do**
    $u_i(t), t \in 1..T$:= $U[-1,1]$
    run reservoir with input stream u_i
    $\mathbf{x}_i(T)$:= reservoir state at final time T
end for
$\mathbf{X} := [\mathbf{x}_1(T)\,\mathbf{x}_2(T)\,\ldots\,\mathbf{x}_S(T)]$
return rank($\mathbf{X}$)

the literature, discussed below. Calculating the rank is discussed in section 3.8.2.

Kernel Rank

Kernel Rank (KR), also referred to as Kernel Quality, or Separation Rank, is a measure of how rich the nonlinear dynamics of the reservoir are. It measures how well the inputs are projected into a high dimensional state space, such that they can be separated by the linear output weight matrix. High KR indicates good separability.

Vidamour et al. [159] use a direct translation of the original definitions, and implement this measure as follows. Consider S maximally distinct input streams, with values drawn from $U[-1,1]$, each of length T, and measure the reservoir state at the end of each input stream. KR is the rank of the resulting $N \times S$ matrix $[\mathbf{x}_1\mathbf{x}_2\ldots\mathbf{x}_S]$, see algorithm 1.

Dale et al. [26] use a different adaptation, which is computationally less intensive (requiring S inputs, rather than $S \times T$), and adapted to time series tasks (where the reservoir state at each timestep is used), but is further from the original definitions. In this approach, there is a single input stream, of length S, and the reservoir state is measured at each timestep. The stream's values are again drawn from $U[-1,1]$, thereby making them maximally distinct. See algorithm 2.

Note that these two algorithms are the same in the case that $T = 1$.

Generalisation Rank

Generalisation rank (GR) measures how robust the reservoir is to noise and avoiding overfitting. The intent is to produce a measure of whether the matrix can generalise over inputs that are similar. Low GR indicates good robustness to noise.

Algorithm 2 KR and GR, Dale et al. [26]

S := length of input stream $\geq N$
r := range of input, 1 for KR, 0.1 for GR
run reservoir with washout stream
for $t \in 1..S$ **do**
    step reservoir with input $u(t) \in U[-r, r]$
    $\mathbf{x}(t)$:= reservoir state at time t
end for
$\mathbf{X} := [\mathbf{x}(1)\,\mathbf{x}(2)\,\ldots\,\mathbf{x}(S)]$
return rank($\mathbf{X}$)

Algorithm 3 GR, Vidamour et al. [159]

S := number of input streams $\geq N$
T := length of each input stream (timesteps)
τ := length of common input stream
$tail(t), t \in 1..\tau := U[-1, 1]$
run reservoir with washout stream
for $i \in 1..S$ **do**
    $u_i(t), t \in 1..(T - \tau) := U[-1, 1]$
    $u_i := u_i + tail$ ▷ append the tail
    run reservoir with input stream u_i
    $\mathbf{x}_i(T)$:= reservoir state at final time T
end for
$\mathbf{X} := [\mathbf{x}_1(T)\,\mathbf{x}_2(T)\,\ldots\,\mathbf{x}_S(T)]$
return rank($\mathbf{X}$)

GR is computed in a similar manner to KR, but instead of running the reservoir over maximally different inputs, the reservoir is fed similar inputs, each with a small amount of noise added.

Vidamour et al. [159] again use a direct translation of the original definitions. Consider S input streams, each of length T, each with values drawn from $U[-1, 1]$, except that the last few values in each stream are set to be the same for all streams. Measure the reservoir state at the end of each input stream. GR is the rank of the resulting matrix, see algorithm 3.

Dale et al. [26] again use a different adaptation of the original definitions. The algorithm for GR is the same as for KR, except that the streams' values are drawn from a reduced range $U[-0.1, 0.1]$, thereby making them similar. See algorithm 2.

Calculating the rank

The standard way to calculate the rank of a matrix is to use singular value decomposition (SVD). The *rank* of a matrix is the number of non-zero singular values. Rank is an integer, so, for small reservoir size N, this can result in a very granular measure.

Due to numerical effects and noise, typically *all* the singular values are non-zero, but some may be very small. To make the measure meaningful (rather than always N) in practice a threshold is chosen, below which the singular values are taken to be effectively zero. This threshold is typically expressed as some percentage of the maximum singular value. The threshold value is essentially arbitrary, and affects the measured rank, so should be stated in any results.

An alternative to this integer-valued rank is the real-valued *effective rank* [85, 123]. Normalise the singular values σ_i to sum to one: $p_i = \sigma_i / \sum_i \sigma_i$. The effective rank is defined as $\exp(-\sum_i p_i \ln p_i)$. If R of the singular values are the same, and the rest are zero, then $p_i = 1/R$ or 0, giving an effective rank of R, which is the same as the standard rank value. For other cases, effective rank gives continuous values, has no arbitrary cutoff, and weights the singular values according to their size, potentially giving a more meaningful result.

The number of measured states S should not be less than N, as the rank of the resulting matrix is $\leq \min(S, N)$. Dale et al. [26, sec.D.a] note that the measured rank increases with S until it eventually converges. Preliminary investigation should be performed to establish a suitable value for S.

3.8.3 Benchmarks for 'Free'

Certain tasks, such as MC and GR, can be calculated from the same data output from the ESN [85]. Both tasks use the same input; the observations of the state can be used to calculate the KR, and an output layer can be trained on the same observations to find the memory capacity. This is useful when using slow physical reservoirs, or computationally intensive search methods such as CHARC.

3.8.4 CHARC

Memory capacity, Kernel Rank, and Generalisation Rank are all relevant properties of Reservoir Computers, but the aim should not be to maximise them all. There is no

agreed upon principle that allows us to correlate these values of these metric. Indeed, different tasks are best accomplished by reservoirs with different properties [26]. Given this, they construct a framework based on Linear Memory Capacity, Kernel Rank, and Generalisation Rank. The Reservoir can then be evaluated not through its performance at the tasks used to measure these properties, but instead by the breadth of the behaviour space available to the substrate or model it is built upon.

When using these property measures, including in the context of CHARC, care needs to be taken to ensure that any arbitrary constants are fully documented to allow comparison between different results. These include washout times, input stream lengths and data ranges, and particular rank algorithm including thresholds.

Dale et al. [26] map four of the benchmark tasks described here to their positions in the CHARC behaviour space. One piece of interesting further work might be to map more of them, in order to enable studies to select tasks that not only take a range of different forms, but also correspond to different areas of Reservoir behaviour spaces.

3.9 Best Practices for Benchmarking

Having looked at many of the benchmarks used in the field of Reservoir Computing individually, we now draw out some best practices for using them.

Many of these best practices relate to experimental setups. One of the advantages of using a benchmark is the ability to compare the results over different Reservoir Computers. These comparisons can typically be done by statistical tests. However, as it is not the norm to share one's complete experimental data output sets, it is essential that researchers be able to reproduce each other's experiments. For that reason, the experimental protocol of benchmark tests should be described in detail.

This section primarily concerns running and reporting benchmarks in reservoir computing. For a more general introduction and tutorial to performing RC, see Cucchi et al. [22].

3.9.1 Datasets and Data Parameters

Using Existing Datasets

State which data source is used. Certain uses of cited benchmarks do not always point to the same dataset. This may be because there are several sources for the same kind of data, as with the various sunspot datasets (table 3.5). It may also be that a single source has recorded several sets of data, as is the case with the IPIX Radar (section 3.7.5).

State which subset of the data is used. There may be one single data source, but only a subset of the data is used. This is the case with the Santa Fe LASER readings (section 3.5.7), of which the most commonly available source contains over 10,000 data points, and is often sub-setted.

State what parameter values are used, and why these were chosen. Some equation-based benchmarks, such as the NARMA (section 3.4.1 and Mackey–Glass (section 3.5.1 systems, rely on specific parameter values. Different values may lead to different behaviour, as with the Mackey–Glass system, where a τ of 16.8 or more will lead to chaotic behaviour. Sometimes several different sets of values are used in the literature.

Introducing New Datasets

State the algorithm and parameter values used in the dataset generation. As noted by Gilpin [50], if a benchmark is based on a differential equation and numerically integrated, then the values used in the integration, such as timestep and gridsize, or the details of a more sophisticated algorithm, can change the detailed generated dataset.

Provide suitable metadata along with new datasets. If using a new experimental or generated data set, it needs to be published with the results, and accompanied by metadata. Gebru et al. [48] provide comprehensive guidelines for what metadata should be provided, covering issues including who created the dataset and who funded them, how and when the data was collected or generated, ethical concerns,

the structure of the data, sampling, noise and errors, labelling, data cleaning and preprocessing, previous uses, distribution, and maintenance. Gilpin [50]'s catalogue of 131 chaotic dynamics benchmark systems follows the approach given there.

3.9.2 Experimental Method

Reservoir Specifications

State all the reservoir parameters used. The first aspect of experimental setup that needs to be reported are the parameter values of the Reservoir Computer used. One example of this being done well can be found by Jaeger [63], where the setup for the Mackey–Glass benchmark is described as having 400 nodes, bias input, inserted noise, and output feedback. When reporting an experiment involving an ESN, some parameters that should be reported are:

- the number of nodes in the Reservoir State
- the input bias, if any
- the leakage rate, if any
- the details of any noise added to the input
- the details of any output feedback
- the connectivity of the Reservoir State
- the network topology, if not random
- the weight matrix distribution, sparsity, and scaling, and how generated

A guide to setting these parameters can be found in Lukoševičius [87].

When reporting an experiment involving an *in materio* RC experiment, the parameters will be device dependent. They should be reported in similar detail.

Washout, Training, Testing data

State which subsets of the data source are used for washout, training, and testing, and justify the values chosen. Reservoir Computing tasks are typically composed of an input sequence of scalar of vector values, fed sequentially into the reservoir.

For benchmark testing, experimental setups divide the input into three categories: washout, training, and testing. Washout data is ignored in the evaluation, is usually at least the length of the reservoir's fading memory, and is used to ensure that the reservoir is driven solely by the inputs and not influenced by its initial state (see also section 3.2.1). The training set is used to train the reservoir output weights, which are evaluated against the testing set.

When these sets are of different lengths across different experiments, this may affect the performance of the reservoir.

It is important to note the length of the washout set, particularly for common dataset-based benchmarks, where different washout lengths mean that the Reservoir Computer is being tested and trained on different subsets of the dataset. The training length of the reservoir can have an effect on its performance: up to a certain training length, increasing the training will lead to more consistent results across experiments, leading to a smaller standard deviation of different performances. After this saturation is reached, increasing the training length has little effect, and may result in overfitting. This saturation can be seen both in performance-based tasks such as NARMA, and value-based tasks such as Kernel Rank [24].

The saturation length of a system will depend on properties of the reservoir, such as the size, as well as the difficulty of the task: therefore, there is no one training length that is optimal for all reservoirs and all tasks: researchers should instead find the saturation length and use that to benchmark. Reporting the saturation length for specific tasks and reservoir sizes may also give researchers another axis along which to compare the behaviour of different types of reservoirs.

Similarly, different testing lengths may affect overall performance. For example, in a free-running prediction task (section 3.3.1), the experiment that uses a longer testing set may yield worse performance, due to the accumulation of errors leading to growing errors [69].

Multiple Runs

Gather data from multiple runs. It is standard to perform multiple runs of a benchmarking experiment, and get a range of results. Any measure of performance that is taken over multiple runs with different inputs and/or internal weight values is more likely to measure how well the dynamics are emulated in general, as opposed to the

ability to follow a specific sequence.

The number of runs should be reported, along with how the setup is changed for each run: different input data, and/or different reservoirs (different random weight matrices for ESNs, different input matrices and physical configurations for *in materio* reservoirs).

For datasets generated from known dynamics, multiple runs on different inputs can be performed with different generated sequences, either generated from different random inputs, or from different time slices of the single generated stream.

With experimental datasets, if the experimental data set is large enough, multiple training and testing sets can be sliced from the overall dataset. In this case, the testing dataset may not immediately follow the training set, and so a separate testing washout period will be needed. If the data set is not large enough for this approach, multiple runs can still be performed on different reservoir configurations.

Other Experimental Parameters

State all other relevant experimental parameter values and algorithms. Various training parameter values may be used. These should be reported in enough detail that the experiment can be reproduced. For complex experimental setups, a good way to clarify the method is to use a pseudocode description of the experiments, possibly abstracted from the experimental harness code.

3.9.3 Evaluation Measures

Time-series Evaluation Measures

With a few exceptions, dynamical systems-based benchmarks typically output a scalar value, rather than a multidimensional vector value, each timestep. Evaluation is then typically performed using one of Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Normalised Mean Squared Error (NMSE), or Normalised Root Mean Squared Error (NRMSE).

Given a desired target output time series $\hat{\mathbf{v}} = \{\hat{v}_t\}_{t=1}^{N}$, an observed output time

std dev	0.1	1	2
MSE	0.009	0.813	3.713
RMSE	0.098	0.901	1.927
NMSE	1.007	1.002	1.039
NRMSE	1.003	1.001	1.019

Table 3.8: Various error measures for 100 randomly generated target values $\hat{v}$ drawn from a Gaussian distribution with a mean of 0 and different standard deviations, compared to an observed constant output value $v = 0$. When the standard deviation of the target output varies, there is a marked effect on the error in the unnormalised measures, but the normalisation removes this difference.

series $\mathbf{v} = \{v_t\}_{t=1}^N$, with the mean $\langle \mathbf{x} \rangle \triangleq \frac{1}{N}\sum_{t=1}^N x_t$, these are defined by:

$$MSE(\hat{\mathbf{v}}, \mathbf{v}) \triangleq \frac{1}{N}\sum_{t=1}^{N}(\hat{v}_t - v_t)^2 \tag{3.21}$$

$$= \langle (\hat{\mathbf{v}} - \mathbf{v})^2 \rangle$$

$$RMSE(\hat{\mathbf{v}}, \mathbf{v}) \triangleq \sqrt{MSE(\hat{\mathbf{v}}, \mathbf{v})} \tag{3.22}$$

$$NMSE(\hat{\mathbf{v}}, \mathbf{v}) \triangleq \frac{MSE(\hat{\mathbf{v}}, \mathbf{v})}{MSE(\hat{\mathbf{v}}, \langle \hat{\mathbf{v}} \rangle)} \tag{3.23}$$

$$NRMSE(\hat{\mathbf{v}}, \mathbf{v}) \triangleq \sqrt{NMSE(\hat{\mathbf{v}}, \mathbf{v})} \tag{3.24}$$

A smaller error indicates better performance. Normalisation makes the NMSE and NRMSE measures dimensionless and independent of any scaling or units of the outputs, so are more comparable across experiments with different systems. Note that here normalisation is performed with respect to the target output, *not* the observed output [87, p.661]. This makes baselining a given system's error using a constant observed value well-defined. In particular, NMSE = 1 and NRMSE = 1 if each v_t is set to the mean of the target values $\langle \hat{\mathbf{v}} \rangle$. Experimental values greater than one indicate very poor perfomance; values less than one may also be achievable with just simple approaches (section 3.9.4).

Depending on what a given experiment is intended to convey, different choices of measure may be appropriate. To see how the difference between the measures, their behaviours are listed in table 3.8.

This shows how normalising the error removes the effect of changing the standard deviation, which may be due to the choice of units or scaling.

While these measures are appropriate for judging the performance of a task

where there is an error margin, it is frequently unclear what has led works in the literature to choose one over the others. When benchmarking using an existing task, it is advisable to use the method existing in the literature, to enable direct comparisons. When creating a new task, however, it is worth considering different methods and choosing the one best suited to the task. The value of a normalised measure is more readily interpretable, and the NRMSE is interpretable as a form of standard deviation.

Classification Evaluation Measures

There are many standard machine learning measures for classification success. We mention just a few common ones here. An appropriate choice for the particular benchmark experiment should be made, and documented.

For a binary classification, success (true positive and true negatives), false positives (type I errors) and false negatives (type II errors) are the simplest measures. If there is some threshold or other classifier parameter, a ROC (receiver operating characteristic) curve can show the performance as a function of that parameter value: a larger area under the curve (AUC) indicates a better classifier.

For classification into a larger number of categories, a confusion matrix plot of predicted versus actual category shows both the proportion of correct predictions, and the distribution of incorrect predictions.

3.9.4 Presenting the Results

Present results as clearly as possible, with tables and pseudocode preferable to reporting it textually.

Base Cases

Calculate baseline success measures for your experiments. As stated above, NRMSE = 1 can be achieved from a constant output set equal to the mean target output. Hence an NRMSE less than 1 is typically claimed to be a success. However, naive predictions can easily result in NRMSEs below 1 for some benchmarks. Thus a stricter success criterion is needed in these cases.

	$v(t) = \hat{v}(t-1)$
NARMA-10	mean=0.826, sd=0.044
Sunspots	0.396
Santa-Fe LASER	0.969

Table 3.9: NRMSE of 'base case' solutions for some common benchmark tasks: (i) NARMA-10, using parameters from table 3.2, averaged over 100 runs each of 1000 data points; (ii) Sunspots, using the full Zurich dataset; (iii) the Santa-Fe LASER task, using the full dataset.

In time series based tasks, for example, a naive prediction is to use the previous target value, by setting $v(t) = \hat{v}(t-1)$.[9] To illustrate this, we calculate the NRMSE of this base case for three popular benchmark tasks (table 3.9). This demonstrates that the baseline for success should be NRMSE $\lesssim 0.8$ for NARMA-10, and $\lesssim 0.4$ for Zurich sunspots.

Statistical Tests

Perform and report statistical tests. When presenting results averaged over multiple runs, present (as a minimum) means *and* standard deviations (or medians *and* quartiles) to demonstrate both average behaviour, and how much variation is present in that behaviour.

When comparing results, perform the appropriate statistical tests for statistical significance and effect size. If doing multiple comparisons, use a Bonferroni correction to reduce the likelihood of false positives.

For more complex experiments, more sophisticated statistical tests may be appropriate. See any standard textbook on statistics for definitions and choices of the appropriate tests.

3.10 Potential Pitfalls of Consistency

Consistency is important to be able to reproduce experiments or compare one's results to that of others. However, if all authors test their Reservoir Computers on the same benchmarks using the same data, we encounter a problem: are Reservoir Computers good at capturing the dynamics of complex dynamical systems based on incomplete data, or are they simply good at predicting this specific sequence

[9] This is the persistence model of weather forecasting: tomorrow will be like today [2].

of values that we have chosen to represent data? One must balance the ability to compare results with other tasks in the literature with the need to avoid overfitting.

We propose that the way to strike this balance is by making any variation deliberate and documented, rather than accidental and based on missing information. Some of the possible approaches to this are detailed below; other approaches are also possible.

Use a diverse set of benchmarks. Different benchmarks require different behaviour from a Reservoir Computer [26]. One can chose benchmarks that span this behaviour space, as well as benchmarks that have different approaches to data and what should be done with it.

Use a mix of standard and bespoke benchmarks. Certain authors choose to mix benchmarks that are common in the literature with ones introduced to illustrate something specific features of their work.

This allows them both to place their work within the literature, and to compensate for where those benchmarks may be lacking.

3.11 Conclusion

Benchmarks are a useful way of evaluating Reservoir Computing in different contexts, particularly in terms of performance, or perhaps “usefulness”. Many of these benchmarks are inherited from the wider Machine Learning community, while others have been developed specifically for the field of Reservoir Computing. This provides a varied pool to choose from, and allows authors to choose benchmarks based on the argument they wish to present, while still placing their use of benchmarks in context with other works in the literature.

However, this mixture of sources has also led to a less well-defined benchmarking culture than in many other fields: there is no one “Reservoir Computing Benchmark Suite”, and even the closest equivalent, the NARMA benchmarks, has half a dozen different parameter sets and experimental setups that are inconsistent across different publications. This makes direct comparison between results of different works within the literature difficult, if not impossible.

New approaches such as CHARC [26] would allow us to survey a given substrate or reservoir model's behaviour space, telling us whether a given substance would make a viable reservoir. While this is valuable work, however, with no direct link to benchmark tasks, it remains merely abstract. There is currently no direct mapping from benchmark tasks to areas within the behaviour space, and it is unclear if such a mapping would be possible.

While this review is by no means comprehensive, we hope that it is a step towards bridging the gap between the various uses of benchmarks, and that we use it to inform our choices in later chapters.

4 Methodology

4.1 Research Questions

The objective of this thesis is to study the question of how we might scale up reservoirs by combining multiple smaller reservoirs together.

In service of this, we seek to answer the following questions:

- How can we combine reservoirs that operate on different timescales in order to improve performance on multi-timescale tasks?
- How can we combine reservoirs with different qualities in order to explore more complex tasks?
- How can we design a heterogeneous reservoir that is specifically adapted to a difficult task, using multiple timescales and materials?

4.2 Experimental Approach

We explore these concepts in simulation, by developing a number of models based on the Echo State Network (ESN). Standard ESNs are initialised using the process described in infobox 4.1. Any variation on the standard ESN will have its implementation described in the relevant chapter.

The *restricted* ESN (chapter 5) is a framework we introduce to describe combinations of ESNs found in the literature. We choose this model over the *modular* ESNs described in chapter 2 as it has a stronger basis in the literature, and because on a single timescale, the entire system can be described by a pair of equations.

We then design a multi timescale ESN using this framework in order to experiment with heterogeneous timescale ESNs (chap. 5).

Infobox 4.1: Initialising a classical Echo State Network

Given an ESN with K inputs, N inner nodes, and L outputs:

We initialise $\mathbf{W_u}$ as a uniform random[a] $N \times K$ matrix with values between $[-1, 1]$.

The inner weight matrix $\mathbf{W}$ is a uniform random $N \times N$ matrix. The values are distributed between $[-\alpha, \alpha]$, with the value of α determined by the desired value of the spectral radius.

The output weight matrix $\mathbf{W_v}$ is trained using either the pseudoinverse method, or with ridge regression, using a regularisation coefficient of 10^{-8}. Both of these methods are detailed in Lukoševičius [87].

[a] All randomness is generated using the Python numpy library's random generator, seeded with the experimental run value.

These models are implemented in python3, with the source code available on GitHub[1].

In order to simulate the constraints of real-world reservoirs, we also introduce a number of "mock materials", which are defined by a collection of parameters we use when building our reservoirs. While physical matderials may not be adequately modeled by an ESN, the mock materials allow us to explore the limitations placed on us by physical materials.

When designing our reservoirs, we perform a number of preliminary experiments, in order to allow us to compare the best instantiation of every model.

These preliminary experiments are described in the relevant chapters, and their results are reported in the appendix.

4.3 Evaluation

We evaluate our models using benchmark tasks, which we review and discuss in chapter 3. We perform each experiment over a number of runs, and then record the Normalised Root Mean Square Error (NRMSE, see eqn. 4.1) of the results over these runs.

[1] The source code can be found at https://github.com/FromAnkyra/NymphESN and https://github.com/FromAnkyra/TempESN

$$NRMSE(\hat{\mathbf{v}}, \mathbf{v}) = \sqrt{\frac{\langle(\hat{\mathbf{v}} - \mathbf{v})^2\rangle}{\langle(\hat{\mathbf{v}} - \langle\hat{\mathbf{v}}\rangle)^2\rangle}} \tag{4.1}$$

In this equation, $\hat{\mathbf{v}}$ is the desired output and $\mathbf{v}$ the observed output. The mean of a vector $\mathbf{x}$ is referred to by $\langle\mathbf{x}\rangle$. Our results are then analysed using descriptive and nonparametric inferential statistics.

5

Modelling and Evaluating Restricted ESNs on Single- and Multi-Timescale Problems

5.1 Introduction

The aim of this thesis is to explore scaling up in materio reservoirs by combining several reservoirs with differing properties. In this chapter, we focus on homogeneous reservoirs, to compare performances of multiple connected small reservoirs against a single larger one, and to provide a baseline for future work.

We introduce a notation for describing a form of reservoir combination. We perform some experiments using the ESN model (see section 2.4), using NARMA-10 (see 3.4.1), sunspots (see section 3.5.6), and a variation we introduce on the MSO benchmarks (see section 3.5.2 for the original MSO benchmark, and section 5.4.1 for our variation).

5.2 the Restricted ESN model

Here we investigate the restricted ESN (rESN) model (see sec. 2.6.2). This provides a model that should allow for the simulation of in materio subreservoirs implemented with different materials, with some physical interconnection between subreservoirs. We introduce a notation that can be used to describe a variety of possible restrictions that may occur in practice, including the models reviewed in sec. 2.6.2.

Infobox 5.1: Concepts Studied in Chapter

Restricted ESNs, homogeneous materials & timescales.

5.2.1 the Standard ESN model

The original ESN model [63, 68] (fig. 2.3) is a Random RNN where only the output weights are trained. A standard ESN can be described by three state vectors and three weight matrices (corresponding to input, internal, and output states and weights), and a set of state update equations.

At time t, the state of the ESN is described by the input vector $\mathbf{u}(t)$, the internal state vector $\mathbf{x}(t)$, and the output vector $\mathbf{v}(t)$. The connections between the nodes represented by the vectors are described by the weight matrices $\mathbf{W_u}$ for the random input weights, $\mathbf{W}$ for the random internal weights, and $\mathbf{W_v}$ for the trained output weights.

We use the update equations for the ESN from Stepney [142]:

$$\begin{aligned} \mathbf{x}(t+1) &= f(\mathbf{W_u}\mathbf{u}(t) + \mathbf{W}\mathbf{x}(t)) \\ \mathbf{v}(t+1) &= \mathbf{W_v}\mathbf{x}(t) \end{aligned} \tag{5.1}$$

where f is a nonlinear function, typically the hyperbolic tangent tanh(.).

As a Recurrent Neural Network, the ESN is a particularly good model for temporal tasks, and tasks requiring fading memory. We explore this in more detail in chapter 3. They do, however, have a number of limitations. The long-term memory of an ESN, for example, is limited by thed number of nodes in the network (see section 3.8.1).

5.2.2 Restricting the Standard Model

The rESN is a variant of the standard ESN model that partitions the internal reservoir state $\mathbf{x}$ into several smaller subreservoir states. This division may be interpreted as restrictions on the connections between parts of the internal state, and thus on the internal weight matrix $\mathbf{W}$. The state vector $\mathbf{x}$ of a restricted ESN with n subreservoirs is the concatenation of the subreservoir state vectors:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \dots \\ \mathbf{x}_n \end{pmatrix} \tag{5.2}$$

where $\mathbf{x}_i$ is the state of the subreservoir i. N_i is the number of nodes in subreservoir i; $N = \sum_{i=1}^{n} N_i$ is the number of nodes in the entire state.

The weight matrix is the concatenation of internal subreservoir weight matrices, and weight matrices describing the connections between subreservoirs:

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_1 & \mathbf{B}_{12} & \dots & \mathbf{B}_{1n} \\ \mathbf{B}_{21} & \mathbf{W}_2 & \dots & \mathbf{B}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{B}_{n1} & \mathbf{B}_{n2} & \dots & \mathbf{W}_n \end{pmatrix} \tag{5.3}$$

where $\mathbf{W}_i$ is the weight matrix that represents the connections within subreservoir i, and $\mathbf{B}_{ij}$ represents the connections from subreservoir i to subreservoir j; $\mathbf{B}_{ij}$ is square if subreservoirs i and j have the same size. The output and input weight matrices are unchanged.

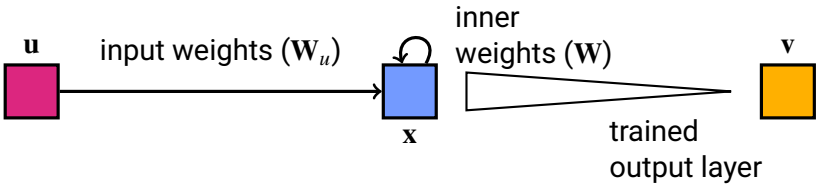
These elements are illustrated in figure 5.1, using the notation introduced in figure 5.1a. Equation 5.1 still defines the transfer from the overall state at time t to time $t+1$.

In general, the submatrices may each have their own, independent properties such as connection density D, the proportion of non-zero weight values. In this chapter we consider uniform subreservoirs (all the $\mathbf{W}_i$ have the same average densities D_W) and uniform connectivities (all the $\mathbf{B}_{ij}$ have the same average densities D_B).

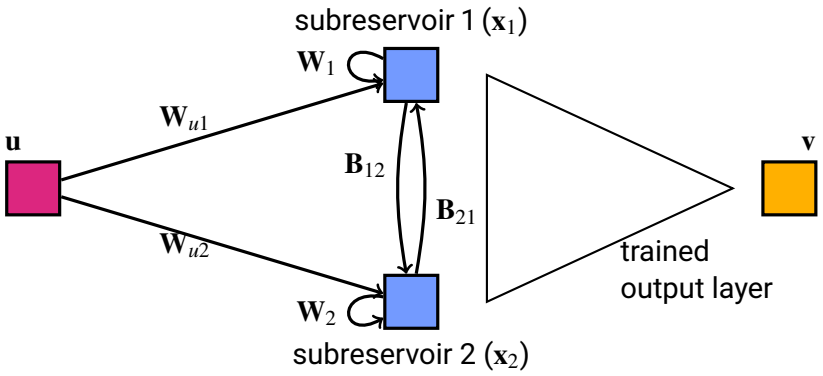
5.3 Density Experiments

We are developing this model in order to provide a means to join *in materio* reservoirs with different properties and different timescales. Before investigating such heterogeneous systems, however, we need to investigate homogeneous restricted reservoirs, to determine the effect of restriction alone. Does an rESN (with its N nodes partitioned into loosely connected subreservoirs) perform significantly differently from a standard ESN of the same dimension (a single reservoir of N nodes)?

In order to test this question, we must determine what constitutes a fair comparison between an rESN and a standard ESN with the same number of nodes. We perform a comparison of the two models over a range of different sizes, on two



(a) Elements of a classical ESN



(b) Elements of a restricted ESN

Figure 5.1: A representation of the elements of a classical and a restricted ESN. (a) The classical ESN takes one or more inputs $\mathbf{u}$ which are then sent to the inner state $\mathbf{x}$ through weighted input edges $\mathbf{W}_u$. The weights within the reservoir, $\mathbf{W}$, are recurrent and randomly initialised. The output state $\mathbf{v}$ receives the inner state through the trained output layer. (b) Elements of a restricted ESN with 2 subreservoirs, showing the partitioned state and components of the internal weight matrix.

common benchmark tasks[1].

5.3.1 Experimental Setup

We wish to discover whether any difference in performance found is due merely to the architecture, or to some other parameter affected by the restriction.

We further wish to ensure that the standard ESN and rESN can each exhibit their best performance on the given task; however, what this entails is not obvious. In the case of the standard ESN, we may perform a simple search to find some "optimal" weight matrix density for the task. Given this optimised density, we investigate two options for the rESN, which we call *patch consistency* and *overall consistency*.

[1]Preliminary results for these two benchmarks are reported in [170]. Here we extend those results to include larger reservoirs (up to 512 nodes), more subreservoirs (up to 8 subreservoirs), and a further benchmark (MSO).

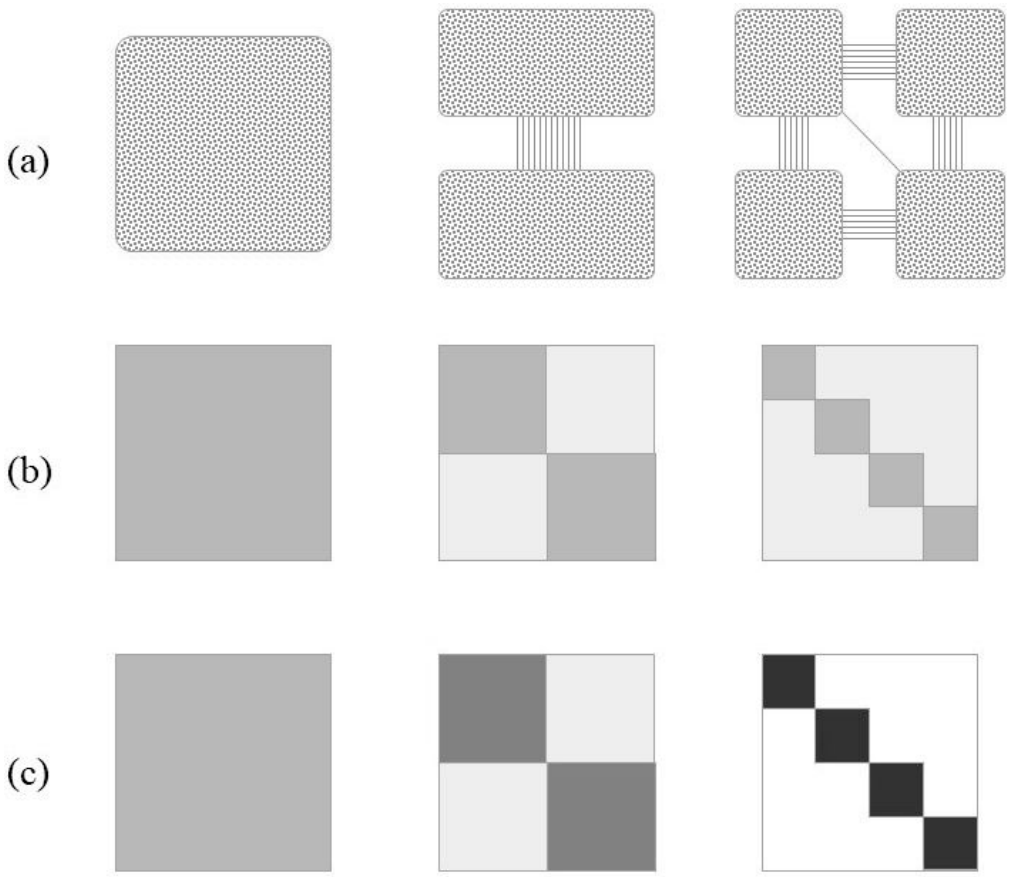


Figure 5.2: Illustration of the two density options: (a) a hypothetical physical reservoir, as one large piece of material, or divided into two or four smaller loosely connected pieces; (b) corresponding model patch-consistency weight matrix for a material with a weight matrix density D: each subreservoir has the same material density $D_W = D$, and there is a lower connection density $D_B < D$ between subreservoirs; (c) overall-consistency weight matrix: the total number of connections is constant, so the subreservoirs get increasingly higher densities, and the subreservoir connections get increasingly lower densities.

Patch-Consistent Density

For this approach, we use a physical analogy to describe the structure of the rESN. If we see this restriction as directly combining multiple physical (material) reservoirs, then restricting a standard reservoir is analogous to having multiple small pieces of a material, and joining these together, in order to emulate a larger reservoir. As such, we should keep the density within the subreservoirs consistent with the overall density of the standard ESN, with sparser connections between subreservoirs. This is illustrated in figure 5.2b.

Overall-Consistent Density

For this approach, we use a neuronal analogy to describe the structure of the rESN, with an underlying neural network architecture being "rewired". Unlike the patch-consistent approach, this does not lead to a lower overall density of the restricted ESN. Having found the optimal connection density for a standard ESN, we redistribute the edges. In order to do this, we change the probabilities of having an edge in the $\mathbf{B}$ and $\mathbf{W}$ regions of the reservoir. We go into more detail on how this is accomplished in infoboxes 5.2 and 5.3.

Thus, the overall density of the rESN remains the same as the density of the standard ESN, while ensuring the constraints on topology that makes it an rESN.

5.3.2 Benchmarks

In order to determine optimal densities and evaluate the reservoir models, we use two benchmarks, NARMA-10 (an open system, or driven system, task) and Sunspots (a closed system task). All training is performed using ridge–regression.

NRMSE

The results are reported as the Normalised Root Mean Square Error (eqn.5.4) evaluated over 50 runs.

$$NRMSE(\hat{\mathbf{v}}, \mathbf{v}) = \sqrt{\frac{\langle(\hat{\mathbf{v}} - \mathbf{v})^2\rangle}{\langle(\hat{\mathbf{v}} - \langle\hat{\mathbf{v}}\rangle)^2\rangle}} \tag{5.4}$$

where $\hat{\mathbf{v}}$ is the desired output; $\mathbf{v}$ is the observed output; $\langle x\rangle$ is the mean $\frac{1}{N}\sum_{i=1}^{N} x_i$.

NARMA10

In our evaluation, we use the following NARMA10 system (see sec. 3.4.1):

$$x(t+1) = 0.3x(t) + 0.05x(t)\sum_{i=0}^{9} x(t-i) + 1.5u(t-9) + 0.1 \tag{5.5}$$

The input at time t, $u(t)$, is uniformly sampled between 0 and 0.5. We use a training length of 3000 data points and washout and testing lengths of 1000 data points each.

N	D_W	D_B		
		2	4	8
64	0.1	0.0010	0.0039	0.015
128	0.1	0.0002	0.0010	0.0039
256	0.1	1.5×10^{-5}	0.0002	0.0010
512	0.1	1.5×10^{-5}	6.1×10^{-5}	0.0002

N	D_O	f		
		2	4	8
64	0.1	114.57	10.84	6.18
128	0.1	491.32	147.02	138.76
256	0.1	459.47	524.01	325.48
512	0.1	1311.52	3603.28	981.64

Table 5.1: Densities used in the NARMA experiments, for 2, 4, and 8 subreservoirs. (left) patch-consistent; (right) overall-consistent

N	D_W	D_B		
		2	4	8
64	0.3	0.051	0.053	0.066
128	0.4	0.05	0.076	0.079
256	0.9	0.1	0.1	0.2
512	1	0.05	0.23	0.23

N	D_O	f		
		2	4	8
64	0.3	308.70	4.87	4.77
128	0.4	265.74	4.38	4.38^a
256	0.5	986.74	4.99^b	4.26^a
512	0.5	6557.1	1116.92^c	591.84^c

Table 5.2: Densities used in the Sunspots experiments, for 2, 4, and 8 subreservoirs (left) patch-consistent; (right) overall-consistent.
Notes: ^{a} $D_O = 0.3$; ^{b} $D_O = 0.5$; ^{c} $D_O = 0.1$. The ideal density in these cases, 0.89, is too high to distribute. The best density given these constraints is used instead, which leads to a worse performance of the standard reservoir.

Sunspots

The Sunspots benchmark is a dynamical systems benchmark task that involves predicting the next output of the dataset based on the previous outputs. We discuss it in detail in sec. 3.5.6.

We use the monthly readings from the Zurich dataset[2], from January 1749 to December 1983. As the existing data limits our input lengths, the training length for this experiment is 1500 data points, with a washout length of 500 data points, and a testing length of 820 data points.

5.3.3 Optimal Density

To find the optimal density D_O for a standard ESN on a given benchmark, we use a two-level grid search (algorithm 4).

For the patch-consistent rESN, the density within each subreservoir, D_W, is set equal to D_O, while a further two-level grid search is used to find the optimal density

[2] https://machinelearningmastery.com/time-series-datasets-for-machine-learning/

Algorithm 4 Optimal density for standard ESN

```
1:  procedure GRIDSEARCH(start, end, step, N)
2:                                          ▷ search over densities d
3:      for d in (start, end, step) do
4:          testsum := 0
5:          for run in range N do
6:              create ESN with density d
7:              v := test ESN on benchmark
8:              testsum += NRMSE(vhat, v)
9:          end for
10:         means[d] := testsum / N
11:     end for
12:     return means
13: end procedure

14: coarse := GRIDSEARCH(0, 1, 0.1, 50)
15: d_1 := argmin(coarse)
16: d_2 := density of the smaller of d_1's neighbours
17: fine := GRIDSEARCH(d_1, d_2, 0.01, 50)
18: return min(fine)
```

between subreservoirs, D_B. In this case, the algorithm is modified to use a step of 0.025 for the first level, and 0.0025 for the second. We also change the *start* and *end* values in the GRIDSEARCH procedure. We set the starting density in our search to $(n/N)^2$, where n is the number of subreservoirs, and N the number of nodes in the full reservoir. This creates a lower bound for D_B, in order to ensure that every connection weight matrix $\mathbf{B}_{ij}$ has at least one entry on average. Our end value for the search is $D_W/4$. This is to ensure that D_B is materially different from D_W, as if no such constraint is set, then the optimal value for D_B is simply D_W.

For the overall-consistent rESN, we introduce a parameter $f = D_W/D_B > 1$, specifying how much higher we wish the internal density in the subreservoirs to be, compared to the connections between them. We then derive D_B and D_W in terms of this f, the overall optimal density D_O, and the number of subreservoirs n (see infobox 5.2). There is an upper bound on the value of f: too high and it is impossible to achieve the desired weight ratio for a given number of connections (see infobox 5.3). Given this upper bound for possible f values, we use a similar two-level grid search[3] to find the best f value for a reservoir of size N, density D_O, for the given benchmark.

[3]The grid search is modified to split the range of f into 10 and use that as the initial step, and then split the range between the optimal value and its neighbour into 10 for the secondary step.

Infobox 5.2: Calculating D_W

Given an ESN with N nodes and an average density $0 \leq D \leq 1$, we wish to restrict that ESN to have n subreservoirs of equal size; we assume n divides N. We set the density within the subreservoirs, D_W, to be greater than the density outside the subreservoirs by a factor of f, that is, $D_W = fD_B$.

In a restricted ESN with n subreservoirs, each of size N/n, there are n regions in the edge matrix **W** of size $(N/n)^2$ with density D_W, and a further $n^2 - n$ regions also of size $(N/n)^2$ with density D_B.

Hence the average density D of such a restricted ESN is:

$$D = \frac{nD_W + (n^2 - n)D_B}{n^2} \tag{5.6}$$

Substituting $D_W = fD_B$, and rearranging to get an expression for D_B in terms of D, we get:

$$D_B = \frac{Dn}{f + n - 1} \tag{5.7}$$

Once D_B is known, we also have D_W from $D_W = fD_B$.

Having found the optimal densities and distributions, we then evaluate the standard and restricted reservoirs against the task over 50 runs. The experiments are performed for ESNs of size $N \in [64, 128, 256, 512]$, and with 2, 4, and 8 equal-sized subreservoir restricted ESNs.

The densities used for each size for each task are given in tables 5.1 and 5.2. We find that the ideal overall density for the NARMA-10 experiment stays the same as the reservoir gets larger, while the ideal density for the sunspots test increases as the reservoir size increases. This is likely a consequence of the different characteristics of the task (discussed in chapter 3). How these characteristics affect the ideal density for the task bears further investigation in future work.

5.3.4 Results

NARMA-10

In this task, the optimal density D_O is consistent at 0.1 (table 5.1).

In the overall consistency case, the results as summarised in the boxplots (fig-

Infobox 5.3: Optimising f

In order to find the best possible restricted ESN within our constraints, we optimise over the parameter f. However, we must somehow limit our search space.

In the restricted ESN, we want D_B to be strictly less than D_W (less dense connections than subreservoirs); therefore, $f > 1$.

To find an upper bound, we assume that every subreservoir is connected to every other subreservoir, that is, every connection weight matrix $\mathbf{B}_{ij}$ has at least one entry. This requires $D_B \geq (n/N)^2$. (In the experiments, the weight matrices are generated probabilistically, so when close to this density limit, it may be the case that there is not an edge between all subreservoirs.)

Rearranging eqn. 5.7 gives:

$$f = \frac{Dn}{D_B} - n + 1 \tag{5.8}$$

The lower limit on D_B gives an upper limit on f:

$$f \leq \frac{N^2 D}{n} - n + 1 \tag{5.9}$$

We also have an upper limit on the derived density, $D_W \leq 1$ (equality implies there are no zero elements in the relevant weight matrix). Substituting for D_W in eqn. 5.7 gives:

$$\frac{fDn}{f + n - 1} = D_W \leq 1 \tag{5.10}$$

Rearranging gives another upper limit on f:

$$f \leq \frac{n-1}{Dn-1} \tag{5.11}$$

Hence we have the upper and lower bounds on f:

$$1 < f \leq \min\left(\frac{N^2 D}{n} - n + 1, \frac{n-1}{Dn-1}\right) \tag{5.12}$$

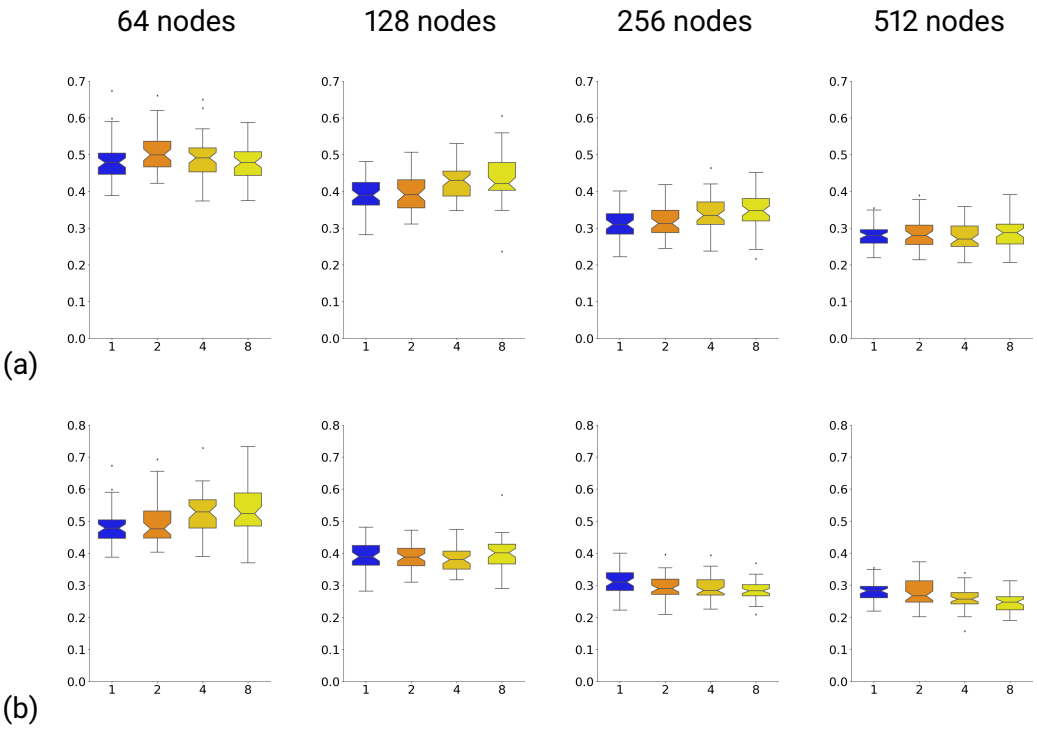


Figure 5.3: The results for the NARMA-10 experiments, for row (a) overall consistency; row (b) patch consistency. In each chart, the x axis labels the number of subreservoirs (the standard reservoir is labelled '1'); the y axis is the NRMSE.

ure 5.3a) show slightly better behaviour for the 4-subreservoir and 8-subreservoir ESNs 64 node case. There are no significant differences in results between the standard and 2–subreservoir ESNs of these sizes, however. For 128 and 256 nodes, the results are slightly worse for the rESNs, getting worse as the number of subreservoirs increases. At 512 nodes, the results level out across all the ESNs.

We hypothesise that the progression of these results, with rESNs working better in the smaller size, worse in the medium sizes, and equally well in the largest size may be explained by searching for an optimal reservoir structure using the f value. When searching for the optimal configuration of the restricted ESN, we find a maximal f-value, and then perform a two-level grid search between 1 and this maximum. The maximal f-value is smaller with smaller ESNs and with more subreservoirs, meaning that the search in these cases would be finer, and hence more likely to find a good result.

We hypothesise that there is therefore a greater chance of finding a good configuration in these smaller experiments. It may also follow that we could replicate these better results for larger ESNs by performing a more thorough search.

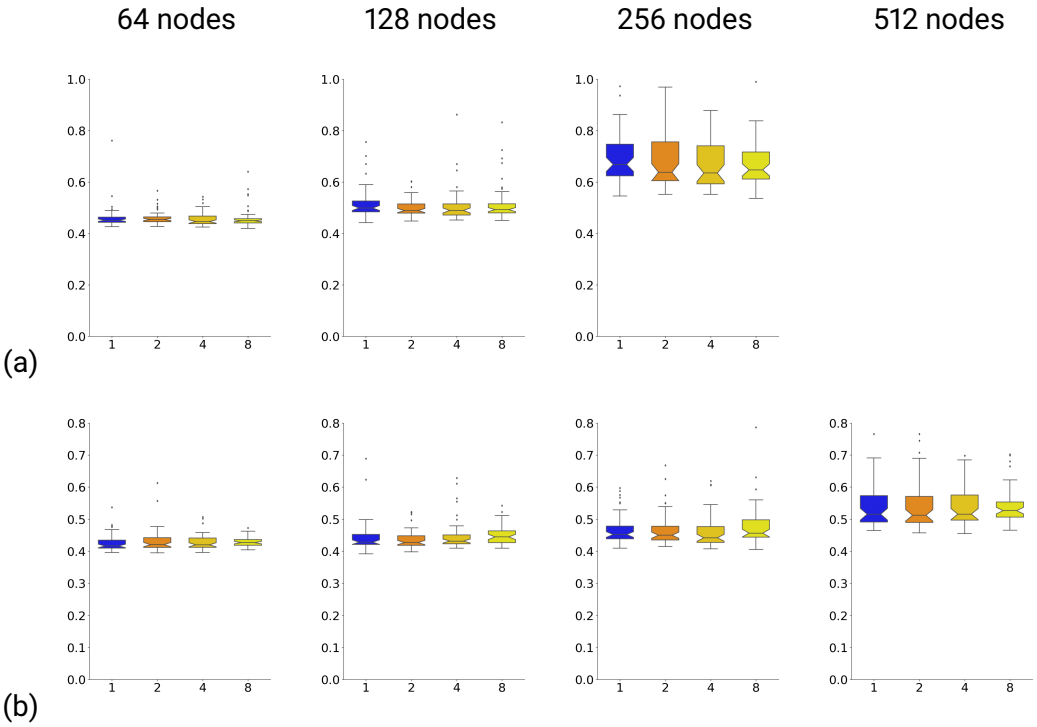


Figure 5.4: The results for the Sunspots experiments, for row (a) overall consistency; row (b) patch consistency. In each chart, the x axis labels the number of subreservoirs (the standard reservoir is labelled '1'); the y axis is the NRMSE. The task requires a certain density to be possible; the highest achievable density for the 512–node case (see table 5.2) lead to divergent results, so they have not been reported here.

In the patch–consistent experiments we can observe that, for the 64 node case, the 4–subreservoir case leads to a worse performance, although the 2–subreservoir case is similar to the standard one. In the 128 and 256 node cases, we observe similar results across standard and restricted ESNs.

Sunspots

Unlike in the NARMA experiments, we observe no consistent optimal density across reservoir sizes; instead the optimal density increases with reservoir size (table 5.2). We also observe that there is much less variation in performance across different ESN sizes (figure 5.4). It follows that any effect that restricting the ESN has will also, for the most part, be much smaller.

In the overall–consistent experiments, we observe little variation between the results from the standard and restricted ESNs, with the 4 and 8-subreservoir cases performing slightly better than the standard and 2-subreservoir one. However, as noted in table 5.2, the ideal density in the 256-node case cannot be redistributed

in an overall-consistent manner. Thus, while the restricted reservoirs in this case perform the same as their standard counterpart, this is not the optimal performance of a 256-node reservoir in this task. This effect gets worse in the 512-node task, as we cannot redistribute the connections between nodes so that the results do not diverge. As such, we do not report those results.

In the patch–consistent experiments, we observe similar results across configurations for all experiments.

5.3.5 Conclusions

Throughout the experiments, there is no large difference between the standard and restricted ESNs. What few differences there are lessens as the ESNs grow larger, disappearing completely by the time we reach the 512-node case.

The more physically realistic of these models is the patch-consistent density. This model also has the advantage of not placing any constraints on the initial standard reservoir's density.

However, it is also the one with the greater differences in performance for smaller sizes. This is particularly evident in the NARMA experiment, where the 8-subreservoir 64 node restricted ESN performs particularly badly.

When modelling these reservoirs, work may be needed to determine what makes a given subreservoir "reasonably large". We will therefore focus on these larger reservoirs in our future work.

Nevertheless, these results indicate that the restricted ESN model, using either overall or patch consistency, does not have a detrimental impact on performance when compared to a single large ESN. Hence restricted ESNs can form a suitable basis for building models of scaled-up reservoirs, heterogeneous reservoirs comprising subreservoirs of different materials, and for working on multiple timescale models.

5.4 MSO* Benchmark Experiments

Having concluded that rESNs are a suitable basis to model larger reservoirs without a detrimental affect on performance, we now look at a more challenging task, which explicitly includes multiple timescales, in order to provide a baseline for future work on

heterogeneous reservoirs. As such, we perform some experiments on two variations of the Multiple Superimposed Oscillators benchmark.

5.4.1 the MSO* Benchmark

The Multiple Superimposed Oscillators (MSO) task is a family of open system prediction benchmarks, which we discuss in detail in sec. 3.5.2. The task involves predicting the next value in a sequence generated by summing multiple sines of the input. For MSO-n, the time series is defined by:

$$y(t) = \sum_{i=1}^{n} \sin(\alpha_i t) \tag{5.13}$$

where t is the timestep, and $\alpha = [0.2, 0.311, 0.42, 0.51, 0.63, 0.74, 0.85, 0.97]$.

In order to provide a suitable baseline for future work, we modify the MSO benchmark as follows. The original benchmark is made harder by adding higher frequency components. Our long term aim is to investigate heterogeneous reservoirs with multiple timescales, with the fastest reservoir focusing on the highest frequency input component, and *lower* frequency sub-reservoirs focusing on lower frequency components, but without undersampling the higher frequency components. The reasoning for this undersampling is discussed in more detail in chapter 6.

Hence here we match the baseline frequency of the reservoir with the *maximum* frequency sine wave, given by $\alpha_8 = 0.97$. To do so, we sample eqn. 5.13 eight times more frequently (or, equivalently, reduce all the original MSO frequencies by a factor of eight):

$$y^*(t) = \sum_{i=1}^{n} \sin\left(\frac{\alpha_i t}{8}\right) \tag{5.14}$$

We further modify the task to scale the input to belong to $[-0.5, 0.5]$ by dividing $y^*(t)$ by $2n$. This rescaled equation is

$$y^*(t) = \frac{\sum_{i=1}^{n} \sin\left(\frac{\alpha_i t}{8}\right)}{2n} \tag{5.15}$$

We refer to this modified benchmark as MSO*. This is different from the original benchmark: the frequencies are lower, potentially making the task easier, but, given the same number of training and testing samples, less of the curve is sampled, potentially making the task harder. So we do not here compare results against other

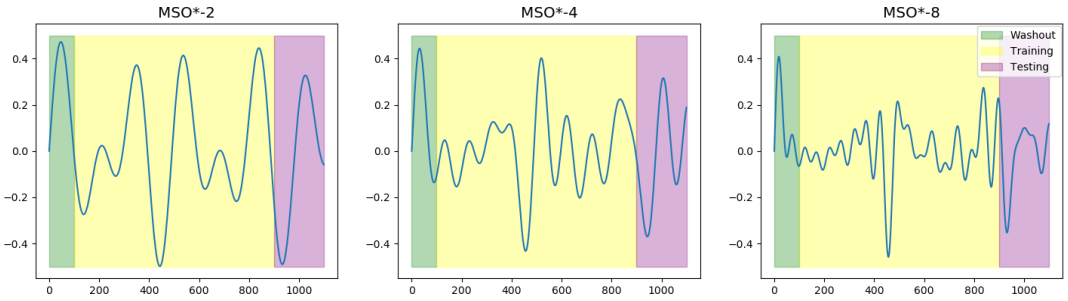


Figure 5.5: Data used in our MSO* experiments (including range scaling), which each include 1100 datapoints (ranging from $t = 0$ to 138 in the original MSO equation). The coloured zones indicate the washout, training, and testing points.

work; rather, we investigate the effect on performance of using (homogeneous) subreservoirs.

We use MSO*-2, 4 and 8.

5.4.2 Experimental Setup

We base our dataset lengths of washout $= 100$, train $= 800$, test $= 200$ in the existing literature [171] and our preliminary experiments. These functions, along with the data lengths used, are shown in figure 5.5.

Continuing on from our conclusions in section 5.3.5, we perform the multi–timescale experiments on patch–consistent rESNs. Instead of finding individual optimal densities as in the single–timescale experiments, we instead use $D_W = 0.005$ and $D_B = 0.001$, chosen using preliminary experiments using the methodology described in section 5.3.

The experiments involve testing the MSO*-2, MSO*-4 and MSO*-8 tasks on a standard ESNs and rESNs with 2, 4, or 8 equal-sized subreservoirs, for ESNs of size $N \in [64, 128, 256, 512]$.

5.4.3 Results

As in our preliminary single-timescale experiments, we take the NRMSE of the output over 50 runs, which are shown in figure 5.6 (note that here we report the logarithm of NMSRE, as the results vary dramatically across systems). In all the experiments, we observe very similar behaviour in the standard ESN as it grows in size:

- best performance (lowest NRMSE) remains the same

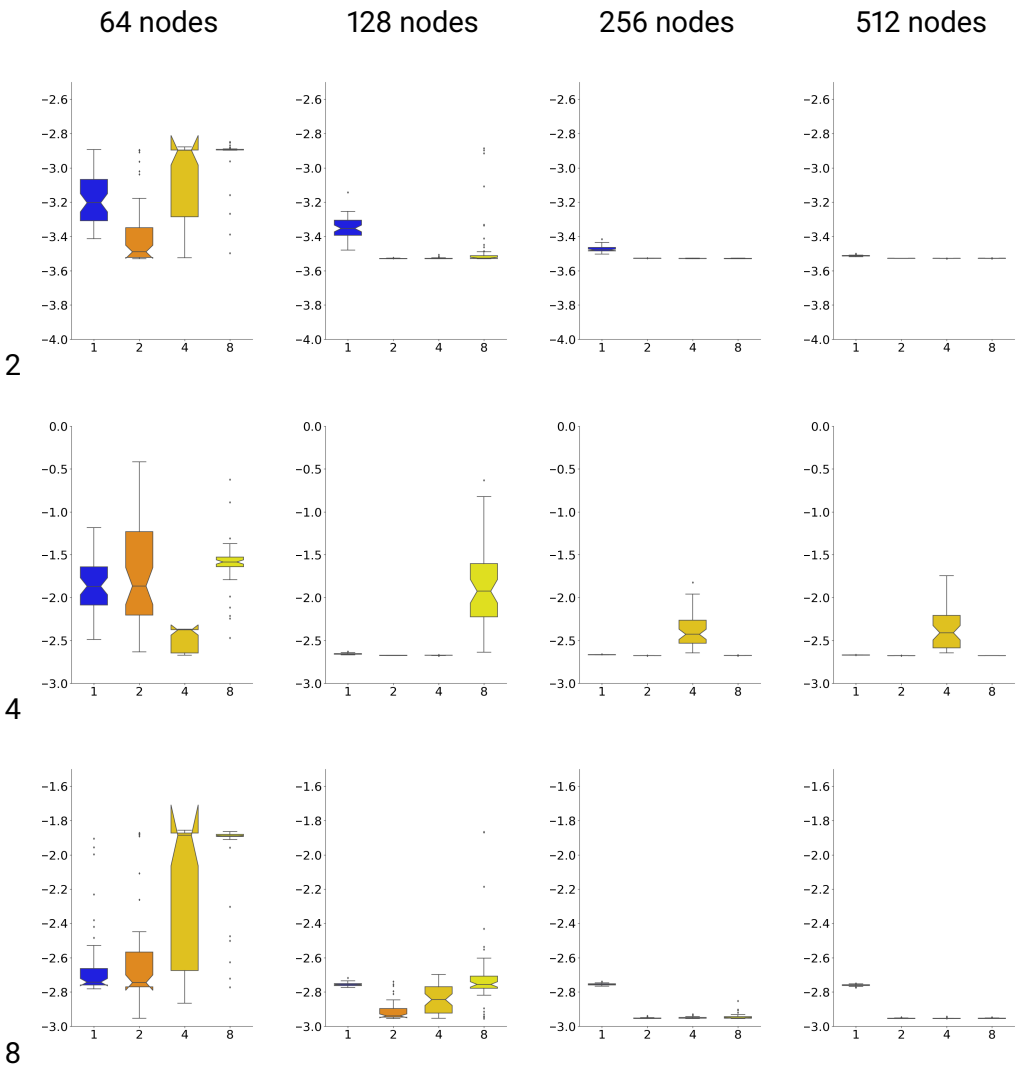


Figure 5.6: The results for the MSO* experiments; the rows are the results for MSO*-2, MSO*-4, and MSO*-8. In each chart, the x axis labels the number of subreservoirs (the standard reservoir is labelled '1'); the y axis is $\log_{10}$(NRMSE).

- worst performance (highest NRMSE) improves

We dub the best performance in this case the “saturation point”, by analogy to the “saturation length” observed in certain benchmarks where increasing the training length has no effect on the performance of the reservoir [24].

The MSO*-2 experiments in figure 5.6 show that the performance of the rESNs is variable with the smaller reservoirs, but outperforms the standard reservoir as we increase the total number of nodes. In the 64-node case, only the 2-subreservoir restricted reservoir outperforms the standard reservoir, with the 4 and 8-subreservoir cases performing worse. By 512 nodes, however, all the reservoirs have reached the saturation point.

In the MSO*-8 experiments, we see a similar behaviour of the Standard ESN as we do in the MSO*-2 task, but with a saturation point which is higher than that of the MSO*-2 task. The MSO*-8 experiment also shows us a negative correlation in the 64-node case between the performance of the restricted reservoirs and the number of subreservoirs. This correlation also exists in the 128-node case, but not for the larger reservoirs. There, we see that all the restricted reservoirs outperform the standard one, and the number of subreservoirs has no effect on this performance.

No such patterns are apparent in the MSO*-4 experiments. There, we have very varied results in the 64-node case, followed by more consistent results in the larger reservoirs, with some exceptions. The 8-subreservoir 128 node restricted reservoir and the 4-subreservoir 256 and 512 node restricted reservoir have much worse results than the rest of the reservoirs. This inconsistency contrasts with the fairly regular results of the MSO*-2 and MSO*-8 experiments.

5.4.4 MSO* Conclusions

Unlike the tasks in section 5.3, we see some direct effects from restricting our ESNs. The effect does not appear to be consistent across the task size, leading to a better performance with MSO*-2 and MSO*-8 as we reach the larger subreservoir sizes, but have some odd outliers in the MSO*-4 case. We suggest that this effect may stem from the number of timescales involved in the task; in future work we will focus on decoupling subreservoirs by using different timescales.

5.5 Discussion and Conclusions

Here we look at the effect that restricting larger reservoirs has on the performance of the reservoir on certain benchmark tasks, as a first step to determining whether joining smaller physical reservoirs together would be a good basis for scaling them up. We find that for more classical benchmark tasks like NARMA-10 and the Sunspots benchmark, there is very little effect that comes from restricting an ESN. With the more challenging task of MSO*, which explicitly incorporates multiple distinguishable timescales, though, this lack of effect no longer holds, and different complexity tasks respond to restriction in different ways.

Previous work [171] shows that a reservoir's performance at the MSO task can be improved upon using spatial decoupling within subreservoirs. We suggest that our inconsistent improvement or lack thereof comes from an analogous "accidental decoupling". In later chapters we focus on ensuring this decoupling exists, focusing on temporal rather than spatial decoupling.

6 MSO* on Multi-Timescale Reservoirs

6.1 Introduction

The Reservoir Computing model is ideal for performing computing on various physical systems, so long as they have sufficiently rich non-linear dynamics [14, 25, 39] coupling relevant degrees of freedom, with fading memory. Some materials that have these properties are magnetic ring arrays[1], spin torque oscillators[148], and arrays of semiconductor optical amplifiers[152, 153]. These in materio reservoirs can make ideal low-power devices that excel at time-series recognition.

The computational properties of these materials do not always scale well with the physical size of the reservoir [27]. In order to more fully exploit the properties of physical materials, we investigate combining multiple reservoirs. We find in chapter 5 that combining multiple homogeneous ESNs together can lead to similar performance to a larger ESN with the same total number of nodes on simple tasks ([167], chap. 5); however, this does not appear to extend to more complicated tasks, particularly those on multiple timescales.

Here, we study whether this performance can be improved by running the component ESNs on multiple timescales. To do this, we introduce a multi–timescale ESN model, built on our previous Restricted ESN model. We also refine our simulations, so that our simulated ESNs more closely resemble the kinds of physical materials that might be used.

Infobox 6.1: Concepts Studied in Chapter

Mock materials, multi-timescale reservoirs, heterogeneous timescales, homogeneous materials.

6.2 Restricted ESNs on Multiple Timescales

6.2.1 Argument for Temporal Heterogeneity

Simple Restricted ESNs perform as well as standard ESNs at simple tasks such as the NARMA and sunspot prediction benchmarks, but more demanding benchmarks, such as the Multiple Superimposed Oscillators benchmark (MSO), perform worse using restricted ESNs ([167], ch. 5). The worse performance can be addressed by decoupling the subreservoirs from each other [171]. A recent review paper [175] has compared this behaviour to that of decoupled neurons communicating with each other [40]. The neurons in Fries [40] are decoupled temporally, using different rhythms; the Decoupled ESN [171] uses physical decoupling, through the weights connecting the subreservoirs. Here, we investigate whether we can recreate this decoupling using multiple rhythms.

6.2.2 Restricted ESNs

In this chapter, we use the same restricted ESN (rESN) as in chapter 5.2. This model is then modified into a multi-timescale model.

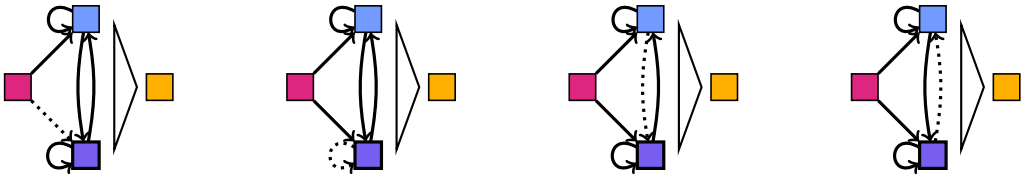
We use the ESN transfer equation (2.5, 2.6), repeated here for convenience.

$$\begin{aligned} \mathbf{x}(t+1) &= f(\mathbf{W_u}\mathbf{u}(t) + \mathbf{W}\mathbf{x}(t)) \\ \mathbf{v}(t+1) &= \mathbf{W_v}\mathbf{x}(t) \end{aligned} \tag{6.1}$$

As in chapter 5, the input weight matrix $\mathbf{W_u}$, state vector $\mathbf{x}$, and inner weight matrix $\mathbf{W}$ are represented as concatenations of the subreservoir elements.

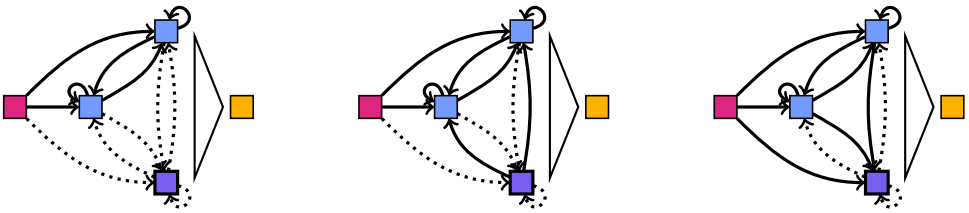
$$\mathbf{W_u} = \begin{pmatrix} \mathbf{W_{u}}_1 \\ \vdots \\ \mathbf{W_{u}}_n \end{pmatrix} \qquad \mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} \qquad \mathbf{W} = \begin{pmatrix} \mathbf{W}_1 & \dots & \mathbf{B}_{1n} \\ \vdots & \ddots & \vdots \\ \mathbf{B}_{n1} & \dots & \mathbf{W}_n \end{pmatrix} \tag{6.2}$$

We could use the same approach to assign a different function f_i to each subreservoir, for example, to model different material properties.



(a) $\mathbf{W_u 2} = \mathbf{O}$ (b) $\mathbf{W}_2 = \mathbf{I}$ (c) $\mathbf{B}_{12} = \mathbf{O}$ (d) $\mathbf{B}_{21} = \mathbf{O}$

Figure 6.1: Four components of how a subreservoir might "sleep" (notation as in fig. 5.1b, with a dashed line indicating a sleeping communication). In each case, the lower subreservoir is asleep, with one of its connections affected: (a) it receives no external input; (b) it receives no input from its previous state; (c) it receives no input from other reservoirs; (d) it sends no output to other subreservoirs.



(a) total sleep (b) input sleep (c) output sleep

Figure 6.2: The sleep modes used for our experiments, which determine the behaviour of reservoirs when asleep. In each figure, the lower subreservoir sleeps when $t = t_{\text{sleep}}$. The weight matrices that are changed are indicated by the dotted lines.

6.2.3 Sleep Modes

Here we introduce a model that allows us to define timescales on a per-subreservoir basis. We use the idea of a "sleeping" subreservoir: Instead of updating on every timestep, a subreservoir might only update "normally" on some of them. On the other timesteps, we may change the input and inner weight matrices to any combination of the following effects (see figure 6.1):

- not receiving input from its previous state ($\mathbf{W_n} = \mathbf{I}$)
- not receiving input from the input layer ($\mathbf{W_{un}} = \mathbf{0}$)
- not receiving communication from other subreservoirs ($\mathbf{B}_{\_n} = \mathbf{0}$)
- not sending communication to other reservoirs ($\mathbf{B_{n\_}} = \mathbf{0}$)

The trained output weights are not affected by the sleeping reservoirs, as these are externally set during the training phase, and do not have any effect on the reservoir state. Instead, the output will always see the last updated state of every subreservoir.

We can compose these individual sleep components in multiple ways. For our experiments here, we define three different sleep modes:

- *total sleep*: when the subreservoir is asleep, no communication takes place between it and other subreservoirs, nor from the external input; figure 6.2a.
- *input sleep*: no inputs or communication from other reservoirs affect the state of the sleeping reservoir, but communication from it can still be received by the other subreservoirs; figure 6.2b.
- *output sleep*: the subreservoir reacts to external and internal inputs, but does not send out any communication to other subreservoirs; figure 6.2c.

In each of the sleep modes, the subreservoir does not react to its own past input, so if reservoir n is in a sleep state, $\mathbf{W}_n = \mathbf{I}$.

At this time, we keep the transfer function as $\mathbf{tanh}$ during both sleep and wake states, as it is equivalent to $\mathbf{id}$ for smaller values, and provides state decay for larger values. However, an argument may be made to change the functions of sleeping reservoirs to $\mathbf{id}$, so that they have no change at all based on past input.

6.2.4 Multiple Timescales with an Extended Transfer Function

Once we have defined what the sleep state for a given subreservoir entails, we can extend our update equation to reflect this. In order to do this, we use time dependent weights in our transfer function. For illustration, consider the case of a single reservoir that is awake every odd timestep, but that sleeps (receives no input, and no internal update) every even timestep. We would have:

$$\mathbf{x}(t+1) = f(\mathbf{W_u}(t)\mathbf{u}(t) + \mathbf{W}(t)\mathbf{x}(t)) \tag{6.3}$$

$$\mathbf{W_u}(t) = \mathbf{W_u}, \mathbf{W}(t) = \mathbf{W} \quad (t \text{ odd})$$

$$\mathbf{W_u}(t) = \mathbf{0}, \mathbf{W}(t) = \mathbf{I} \qquad (t \text{ even}) \tag{6.4}$$

For a standard ESN, this model would be overly complicated; we can easily simulate a sleep state by removing every other input value from our input set. In a restricted ESN, however, this model allows us to have one subreservoir sleep while the other subreservoir is awake. For illustration, consider a two subreservoir case.

During even timesteps, subreservoir 1 is in a total sleep state. As such, it receives no input from its past state, nor input from subreservoir 2. It also sends no output to subreservoir 2. Meanwhile, subreservoir 2 is awake during every timestep.

We then have:

$$\mathbf{W_u}(t) = \begin{pmatrix} \mathbf{W_{u1}} \\ \mathbf{W_{u2}} \end{pmatrix}, \mathbf{W}(t) = \begin{pmatrix} \mathbf{W}_1 & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{W}_2 \end{pmatrix} \ (t \text{ odd})$$
$$\mathbf{W}_u(t) = \begin{pmatrix} \mathbf{0} \\ \mathbf{W_{u2}} \end{pmatrix}, \mathbf{W}(t) = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_2 \end{pmatrix} \quad (t \text{ even}) \tag{6.5}$$

This is the simplest multi-transfer equation model: it allows for only one subreservoir to be asleep at a time. Here, we use an ESN with three subreservoirs, each with its own sleep/wake *rhythm*. To allow for this, we must be able to build the weight matrix for the full reservoir at each timestep, a process which we describe in section 6.2.5.

6.2.5 Building the Reservoir Edge Matrices

Our model allows us not only to have subreservoirs be in their wake or sleep state independently of each other, but also for each reservoir to have their own sleep mode for communicating with other subreservoirs when asleep. In order to allow for this, we need to build the weight matrix at every timestep t.

The $N \times N$ full weight matrix is built by taking the Hadamard product (element-wise multiplication, denoted $\odot$) of $\mathbf{W}_{\text{BASE}}$, the inner weight matrix of our reservoir when every subreservoir is awake, and the $N \times N$ connectivity matrix $\mathbf{C}_i^t$ for each subreservoir i. $\mathbf{C}_i^t$ is constructed from:

- an $N_i \times N_i$ matrix $\mathbf{c}_i^t$ corresponding to the sleep state of subreservoir i at time t ($\mathbf{1}$ if awake, $\mathbf{0}$ if asleep; see fig. 6.1b)
- horizontal and vertical strips of blocks corresponding to the sleep modes of the connections between i and the other subreservoirs, at time t ($\mathbf{1}$ if awake, $\mathbf{0}$ if asleep; see fig. 6.1c,d)
- an all-ones matrix in every other block.

We illustrate this in algorithm 5 and figure 6.3.

Algorithm 5 building the restricted weight matrix at time t

$\mathbf{W}_{\text{BASE}}$:= weight matrix, with all subreservoirs awake
for t in timesteps **do**
    for i in subreservoirs **do**
        if subreservoir i is awake **then**
            $\mathbf{C}_i^t := \mathbf{C}_i^{\text{wake}}$
        else
            $\mathbf{C}_n^t := \mathbf{C}_i^{\text{sleep}}$
        end if
    end for
    $\mathbf{W}^t := \bigodot_1^n \mathbf{C}_i^t \odot \mathbf{W}_{\text{BASE}}$
end for

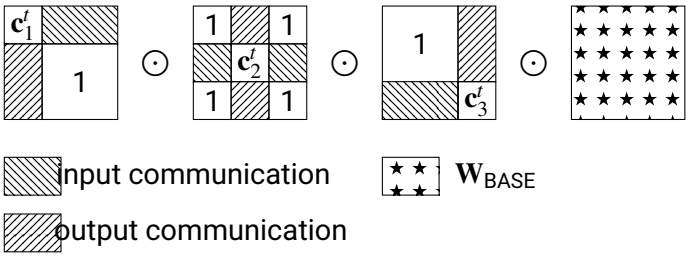


Figure 6.3: Building the restricted reservoir's weight matrix $\mathbf{W}^t$ for timestep t, taking into account which reservoirs are awake and which are not.

6.3 Mock Materials

When studying combining heterogeneous reservoirs, we encounter a plethora of possible parameter values, making them difficult to compare. Some of these parameters include: density, spectral radius, architecture/topology, number of nodes. When studying heterogeneous timescales, we introduce the following further parameters: rhythm (pattern of awake/asleep); tempo (number of timesteps that a reservoir is asleep over); sleep mode (behaviour of the reservoir when asleep).

This results in a combinatorial explosion of possibilities. In order to reduce this space, our work focuses on a set of three simulated materials, each with their own fixed properties. These materials are inspired by materials and models used for reservoir computing, but some liberties are taken, both because the ESN model does not fully correspond to existing materials, and to provide a larger range of properties. We also use different sleep modes for each material, both to reflect the fact that materials may have different sleep properties, and to study a range of different modes.

Here, we simulate three materials by constraining various parameters of the

ESN, and we focus on the effect of different timescales over multiple subreservoirs made of a single simulated material. In future work, we will focus on combining the materials. As such, this paper does not describe a tempo or rhythm for each individual material, instead, the rhythms for each subreservoir is asleep are described in section 6.4.

6.3.1 Ring

The Ring mock material topology is intended as an imitation of ring reservoirs [119], a structure later reused in delay-line RC [4]. The nodes in this substrate are laid out in a ring, connecting only to themselves and to a single neighbour.

The weight matrix of the ring material has weights drawn from $U[-a,a]$, normalised to a spectral radius $\rho(\mathbf{W}) = 1$.

A ring subreservoir communicates with other subreservoirs via two "spine" nodes: one node within the ring receives any input communications from other subreservoirs, while another transmits all output communications. This communication model is again based on the delay-line reservoir, where the input is fed into the reservoir via a single virtual node at every timestep τ. We use two distinct, randomly selected spine nodes to allow for some processing of information before the state is communicated to the other reservoirs.

For this material, we use total sleep mode for most experiments.

6.3.2 Lattice

The Lattice mock material is inspired by reservoirs made out of magnetic ring arrays [1], which can be laid out in a grid formation. As such, the nodes are laid out in a grid, with each node having an edge to itself and to every node in its Von Neumann neighbourhood. The edges do not wrap, as this would instead form a torus [28].

The weight matrix of the ring material has weights between $U[-a,a]$, adjusted to ensure the spectral radius $\rho(\mathbf{W}) = 1$.

Communication between a lattice subreservoirs and other subreservoirs in a restricted ESN takes place via "sides" of the grid, in order to reinforce the physical idea of distance between unconnected nodes. In preliminary work, the model was arranged so that the nodes on one side of the lattice received all the input commu-

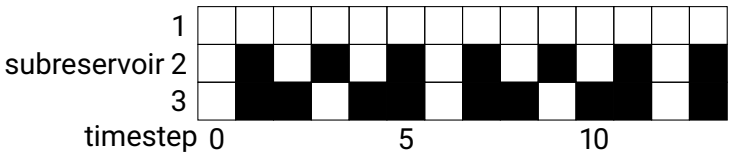
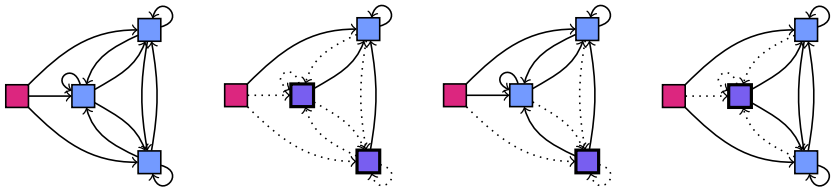


Figure 6.4: Sleep rhythms used for the multiple timescale experiment. At every timestep, if the subreservoir is awake, the square is white, and if it is asleep, the square is black.



(a) $t = 1$ (b) $t = 2$ (c) $t = 3$ (d) $t = 4$

Figure 6.5: Connections between subreservoirs during the first four timesteps of our lattice experiment. (a) $t = 1$, all subreservoirs are awake; (b) $t = 2$, both subreservoir 2 and subreservoir 3 sleep; (c) $t = 3$, only subreservoir 2 is asleep; (d) $t = 4$, only subreservoir 3 is asleep. The dotted arrows indicate which connections are affected by the sleeping reservoirs.

nications from other reservoirs, while the nodes on the opposite side transmitted outputs. However, this model performed poorly, as it took many timesteps for information to propagate through the subreservoir and on to its neighbours. Here, we use a single side for both input and output, leading to improved results.

For this material, we use input sleep mode for most experiments.

6.3.3 Bucket

The Bucket mock material is inspired by a bucket of water, one of the first materials in which reservoir computing was performed [39]. Unlike the other two materials, this one has a fully connected weight matrix, giving us a “well-mixed bucket”. To reflect the relatively simple dynamics of the system, the spectral radius $\rho(\mathbf{W}) = 0.8$.

The communication between a bucket subreservoir and other subreservoirs is performed by four communication nodes, which both receive input communications and transmit output ones. This allows us to study three restricted reservoirs with different amounts of communication between them.

For this material, we use output sleep mode for most experiments.

6.4 Experimental Setup

For each material, we compare the performance of a restricted ESN with three 64-node subreservoirs on a single timescale, to a restricted ESN with three subreservoirs on three timescales. For the multi-timescale experiment, each subreservoir has its own sleep/wake rhythm: one is awake every timestep, one is awake every other timestep, and one is awake one timestep out of three (Fig. 6.4). No attempt is made here to match the rhythms used to the task; instead, we use the simplest nontrivial set of rhythms.

We illustrate how this rhythm structure affects the connections between subreservoirs in our lattice experiment over four timesteps in figure 6.5.

6.4.1 Scaling the Weight Matrix

Scaling the weight matrix is a recommended step when working with Echo State networks, so that the reservoir state is less likely diverge. We discuss this scaling in further detail in chapter 2.4.1.

Our mock materials each have a given spectral radius. However, the spectral radius of the full reservoir weight matrix (composed of three subreservoirs of the same mock material) may be different from the spectral radius of the subreservoirs, because of the extra off-block-diagonal communication edges (matrices $\mathbf{B}_{ij}$). We find in preliminary experiments that we get better results if the off-diagonal weights are rescaled as well.

These preliminary experiments show that scaling the spectral radius of the full weight matrix to one does not give good performance; instead, we use the more restrictive method of scaling the largest singular value, $\bar{\sigma}$, to one [173]. This results in a spectral radius less than one, but here it gives improved performance.

The largest singular value of the full reservoir could be scaled to 1 by uniformly scaling the entire weight matrix. However, in the case of our restricted reservoir model, this would affect the properties of the subreservoirs, by scaling the weights within each of them. This would risk losing any properties that are particular to our mock material based on its weight matrix. This would also not readily transfer to using physical materials, as their effective weight matrix is a given. So instead, we scale only the off-diagonal connections between different subreservoirs to ensure

Algorithm 6 Scaling the off-diagonals

1: $\mathbf{W}_B$:= $\mathbf{W}$ from eqn. 5.3 with each $\mathbf{W}_i$ replaced by all-ones
2: $\mathbf{W}_B$:= $\mathbf{W}_B \times$ random matrix
3: $\bar{\sigma}$:= largest singular value of $\mathbf{W}_B$
4: $\mathbf{W}_B$:= $\mathbf{W}_B / \bar{\sigma}$

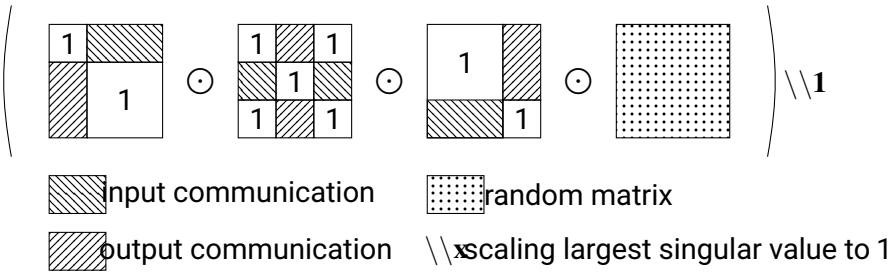


Figure 6.6: Representation of Scaling the Off-diagonals

the overall scaling, as illustrated in algorithm 6 and figure 6.6.

Once $\mathbf{W}_B$ is scaled, we superimpose the weight matrices for the subreservoirs over the diagonals. This step does change the scaling slightly, but not significantly for our purposes.

6.4.2 The MSO* Benchmark

As in chapter 5, the Multiple Superimposed Oscillators prediction benchmark task. We discuss the original task in detail in chapter 3.5.2.

We introduce MSO* ([167], chap. 5). We use this modification due to the fact that the slowest reservoir in our experiments is only "awake" one out of every three timesteps. In order to prevent potential undersampling of the higher frequency tasks (as illustrated in fig. 6.7), we sample the MSO function 8 times more frequently. We introduce the modified equation 5.14, repeated here for convenience:

$$y^*(t) = \sum_{i=1}^{n} \sin\left(\frac{\alpha_i t}{8}\right) \tag{6.6}$$

This input is then scaled, giving us:

$$y^*(t) = \frac{\sum_{i=1}^{n} \sin\left(\frac{\alpha_i t}{8}\right)}{2n} \tag{6.7}$$

We choose this task as it includes multiple incommensurable timescales. The task has previously been approached by physically decoupling the reservoirs from

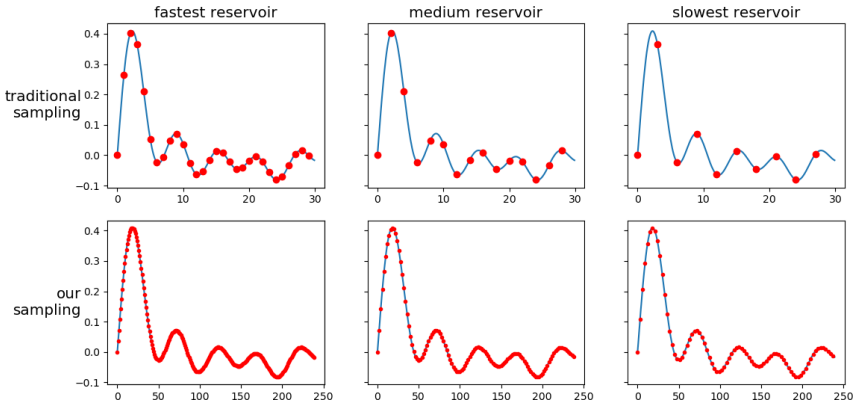


Figure 6.7: Illustration of how we sample the MSO*-8 function, compared to the typical MSO-8 sampling. The higher row shows the more commonly used sampling, where each datapoint is $y(t)$. As we can see though, in a multi–timescale reservoir, this would lead to undersampling by the slower reservoirs. Reduce the MSO frequency by a factor of 8, allowing reservoirs of all speeds to see the details of the curve.

each other [171]. In this work, we add a temporal decoupling element to this physical decoupling[1] in order to find out whether this has an impact on the results.

In this chapter, we study three MSO* problems: MSO*-2 , MSO*-4, and MSO*-8. We use the same data lengths as previous work [171]: 100 timesteps for the washout, 600 steps for the training, and 200 steps for testing.

6.5 Results

The results for all the experiments show that generally, the multi-timescale reservoirs perform worse than the single timescale ones. This effect does not apply as strongly across materials, however, being more pronounced in the “bucket” material and almost nonexistent with the “ring” material.

The “ring” material (fig. 6.8) is the one with the best performance overall. The performance of the single timescale reservoir is very similar to the “bucket” single timescale (fig. 6.10), both of which are slightly better than the lattice material. More promisingly, with this material, the multi-timescale case gives us a similar MSO*-2 performance in the best case (with a higher variance), a very similar performance overall for MSO*-4, and a slightly better performance for MSO*-8. In the MSO*-8 case, this leads to the multi-timescale “ring” reservoir having the best performance

[1]The physical decoupling is accomplished here by scaling the largest singular value of the off-diagonals of the weight matrix to 1 (see fig. 6.6).

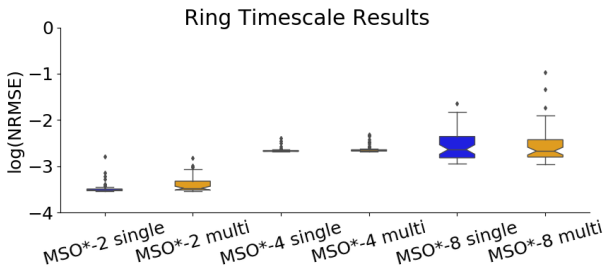


Figure 6.8: Ring material results for the MSO* experiments. The results are given as $\log_{10}(NRMSE)$.

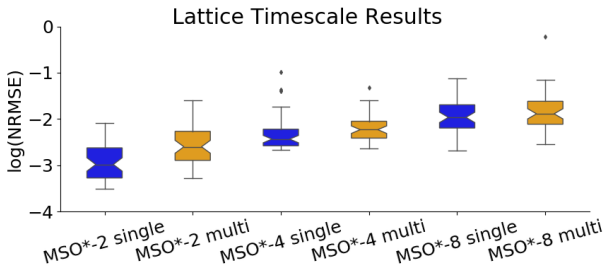


Figure 6.9: Lattice material results for the MSO* experiments. The results are given as $\log_{10}(NRMSE)$.

at this task overall.

The “lattice” model (fig. 6.9) performs poorly compared to the other materials, in both the single and multi-timescale case. This performance is worsened by adding multiple timescales, although the effect diminishes as the task gets harder.

The “bucket” material (fig. 6.10) has single-timescale results that are similar to that of the “ring” material. In this material though, the multi-timescale reservoir performs consistently worse than the single-timescale one. As with the lattice material, this difference in performance diminishes as the task gets more complex.

Two likely reasons for the difference in the effect of multiple timescales are:

- The sleep mode used has a significant effect on how well a multi–timescale

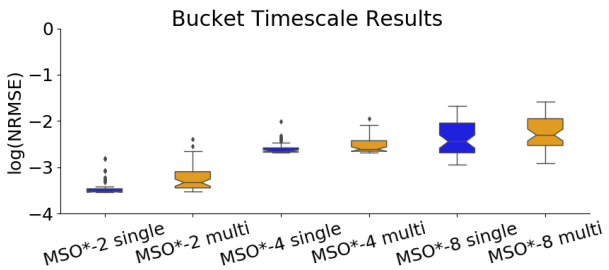
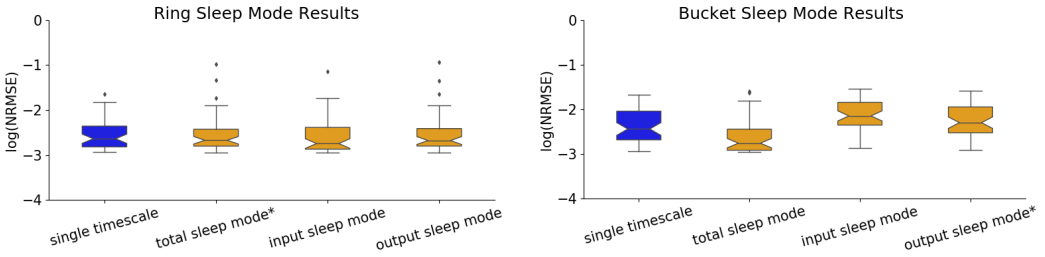


Figure 6.10: Bucket material results for the MSO* experiments. The results are given as $\log_{10}(NRMSE)$.

reservoir performs

- the physical properties of our reservoirs may suit them more or less to a multi–timescale approach.

In order to test these hypotheses, we run the MSO*-8 experiment on the “ring” (fig. 6.11a) and “bucket” (fig. 6.11b) materials, but this time using all three different sleep modes.



(a) (b)

Figure 6.11: Results for the experiments performed on the (a) Ring and (b) Bucket materials, using different sleep modes. The previously used sleep mode is marked by *.

We see in these additional results that both hypotheses play a factor. With the “ring” material, the different types of sleep modes appear to have very little effect on the results. In each case, the multi-timescale model works better than the single–timescale one, but no sleep mode appears to be better than the other. With the “bucket” material, however, the type of sleep mode has a very large effect, with the model using the total sleep mode performing as well as the single-timescale model, and the other two performing much worse.

This leads us to conclude that the different sleep modes can have an effect on performance, but that this effect changes depending on the other properties of the reservoir.

6.6 Conclusions

In this work, we introduce an extension of the ESN model that allows us to operate at different timescales over different regions of the reservoir, which we call subreservoirs. We introduce “sleep states” for the subreservoirs, which let us study different possible implementation of multiple timescales. To evaluate these timescales, we

introduce “simulated materials”, which allow us to experiment over a number of different parameter sets and sleep models.

We find that adding multiple timescales can sometimes improve performance of reservoirs at certain multi-timescale tasks, such as the MSO*-8 task. This improvement in performance depends on both the mock material properties, and on the sleep mode used. Future work is needed to explore this in further depth, for which this work can be used as a baseline.

Our model allows for mixing and matching of different materials, as well as sleep modes. In this work, we focus on the heterogeneity of the *timescales* of the different reservoirs. In future chapters, we will study heterogeneous *materials*.

7 Heterogeneous Reservoirs for Predicting Multivariate Physiological Data

7.1 Introduction

In materio Reservoir Computers can present a challenge as they are broadly reliant on the dynamics of the material used to be effective. This can be partially addressed by modifying the properties of a material to suit a task better, or by matching a given material to a task based on its properties [25]. For a more complex task, however, this may not be sufficient.

In chapter 5 we focus on scaling up reservoirs by combining multiple reservoirs. This allows us to create (homogeneous) reservoirs with heterogeneous timescales to work on multi-timescale tasks (chap. 6). In this chapter, we look at heterogeneous reservoirs on a single timescale, focusing on having distinct parameter sets between subreservoirs.

To explore this technique, we use a benchmark task that we expect a heterogeneous reservoir would be well suited to. The sleep apnea Benchmark is a prediction task using the Santa Fe Dataset B [59, 126], initially distributed during the Time Series Prediction Competition [163]. We first discuss this task in sections 3.5.8 and 3.7.4[1].

This data set consists of three linked time series: a patient's heart rate, respiration

[1] In chapter 3, we discuss using this task as both a prediction benchmark and a possible future stationary time series classification benchmark. In this chapter, we only use it as a prediction task.

Infobox 7.1: Concepts Studied in Chapter

Multi-material reservoirs, multi-input heterogeneous reservoirs, homogeneous timescales, heterogeneous materials

data, and blood oxygenation levels. We use our restricted ESN model to explore how having all three time series as input may improve the results of our predictions, and how the input methods and heterogeneous subreservoir parameters might further affect the results.

In section 7.2 we go over the restricted ESN model, and creating heterogeneous reservoirs using the mock materials introduced in chapter 6. In section 7.3 we discuss the sleep apnea benchmark and its various strengths. In section 7.4 we review our experimental setup, with particular attention to our choice of mock materials and input handling. Finally in section 7.5 we present our results, which we discuss in section 7.6.

7.2 Heterogeneous Reservoirs

7.2.1 The Restricted ESN Model

This chapter uses the restricted ESN model ([167], chap. 5), a method of describing combinations of ESNs based on the classical ESN model [63, 68].

7.2.2 Mock Materials

In chap. 6, we introduce “mock materials” when investigating scaling up reservoirs. The mock materials are three fixed sets of parameters, to allow us to investigate scaling up reservoirs with distinct ESNs, while avoiding a combinatorial explosion of possible parameter values.

Here we focus on combining heterogeneous reservoirs, and reuse these mock materials to provide three different ESN parameter sets. Each restricted reservoir consists of three 64-node subreservoirs with the parameters of one of the following mock materials.

Ring: The nodes of the subreservoir are laid out in a ring topology, with one node receiving input communications from other subreservoirs and a different node sending output communication to other subreservoirs.

Lattice: The nodes of this subreservoir are laid out as a lattice, with one side of this lattice receiving and sending communications with other subreservoirs.

Bucket: This material is a fully connected graph. It communicates with other subreservoirs through 4 randomly chosen nodes.

Combining Subreservoirs

When combining multiple subreservoirs together, we scale the full weight matrix of the reservoir using the scaling technique introduced in chapter 6. In contrast to that work, though, instead of normalising the largest singular value of the reservoir, we instead scale the spectral radius to be 1.

7.3 Sleep Apnea Benchmark

7.3.1 Bio-medical uses of RC

There are a number of bio-medical tasks that can be performed with Reservoir Computing, as summarised in a recent review [175]. Many of the reviewed tasks involve removing noise from ECG or MCG signals [37, 124, 133]. Others are various forms of classification tasks: of arythmic heartbeats [21], speech signals [8], medical images [75], or the classification of gestures based on EMG signals [34, 47] and detecting seizures using EEG signals [78]. There has also been work investigating using reservoir computing to control prosthetic limbs [78].

7.3.2 Benchmark Description

The sleep apnea benchmark is a prediction task that consists on three datasets of physiological measurements from a patient with sleep apnea. In sections 3.5.8 and 3.7.4, we discuss it as a potentially interesting RC benchmark, due to its complexity, the three related but distinct inputs, and the different timescales of the datasets.

The object of the task is to predict the next value in three datasets, which record the patient's (i) heart rate (ii) respiration rate (iii) blood oxygen saturation over a period of 4 hours and 43 minutes. As this data has only been used in reservoir computing as a single-input benchmark, we study it in detail in this section.

7.3.3 Attributes of the Data

The data being used is of the patient going through periods of normal sleep as well as periods of perturbations, when the patient is experiencing apnea[2]. During the perturbations, the physiological attributes measured by the data affect each other. This can be observed in figure 7.1, particularly with the irregularities starting just before timestep 24,000. The heart rate increases, the respiration becomes a lot more dense, and blood oxygen plateaus. As such, it is interesting to study the time series simultaneously, instead of independently.

Heart rate

The heart rate data was measured with an ECG monitor, then synthesized into an "immediate heart rate" signal, disregarding other physiological information from the ECG signal. This synthesis technique has some pitfalls when a patient has abnormal heartbeats. To avoid these pitfalls, a patient without these anomalies was selected.

The heart rate fluctuates with the respiration, accelerating during inspiration and decelerating during expiration. The fluctuations increase during apnea periods.

Respiration

Our second dataset concerns the respiration data of the patient. This data was initially measured both by measuring the chest volume and the nasal airflow of the patient. In the original competition [163], only the nasal airflow was distributed, as the chest volume data was judged to be redundant.

The source for the data we use in this chapter [59, 126] states instead that the chest volume is used. For consistency, the data is referred to purely as "respiration data" throughout this work. As this respiration data is very noisy, we speculate that it is more likely to be the nasal airflow. This noisiness, as well as the symmetricity of the data, makes this task extremely challenging, as leading to mediocre results.

[2]To fully capture this, we have deliberately selected a section of the dataset which includes both normal and apnea periods in both the testing and training phases (fig. 7.1). This allows us to observe the correlations between the datasets, but also makes the task more challenging.

Blood Oxygen Saturation

The blood oxygenation data used was measured using the colour of the hemoglobin, with a sensor clipped to the patient's ear. This sensor updates more sluggishly than the others, leading to changes in blood oxygen being recorded later than changes in the other physiological symptoms. We do not compensate for this phase shift, as we attempt to treat the reservoir as a black-box. The phase shift may allow us to predict the blood oxygenation in "advance" of the others. We do not find such an effect in this work's prediction case, but a classification prediction task may encounter such an effect.

7.4 Experimental Setup

The goal of the Sleep Apnea task as we use it, is, given as input at time t the values from our dataset, to predict the values given as input at time $t+1$. As we discuss in section 7.3.2, we hope to discover whether studying all three inputs together will yield better results than studying each input individually.

Future works may instead focus on predicting periods of apnea, in order to work towards a better understanding of the irregularities in the data. In this work, however, we focus solely on prediction, to better study the effect of multiple inputs.

Our results are reported in figs 7.6–7.8 by taking the NRMSE of the testing phase over 100 runs.

7.4.1 Data

We normalise each dataset to lie in $[-0.5, 0.5]$:

$$\mathbf{x}' = \frac{\mathbf{x} - \min(\mathbf{x})}{\text{range}(\mathbf{x})} - 0.5 \tag{7.1}$$

where $\mathbf{x}$ is the original dataset.

We set the datalengths (fig. 7.1) such that we can observe both regular sleep and apnea periods within the training phase. In order to do this, we choose a starting point 22000 points in, to get sufficiently short periods of normal sleep and apnea. The data lengths are:

- Washout: 1000

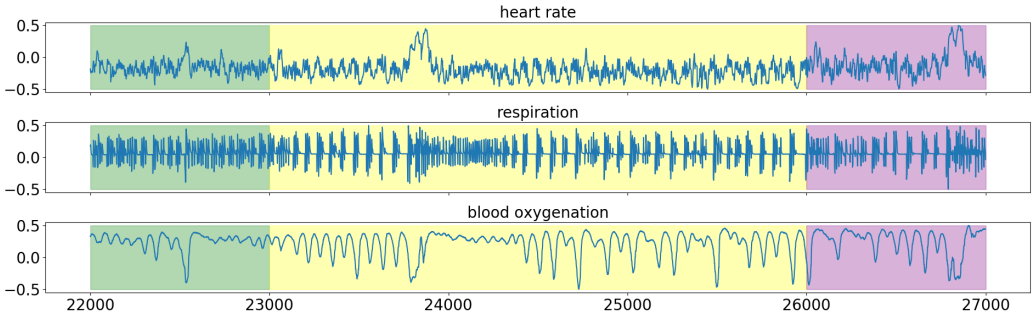
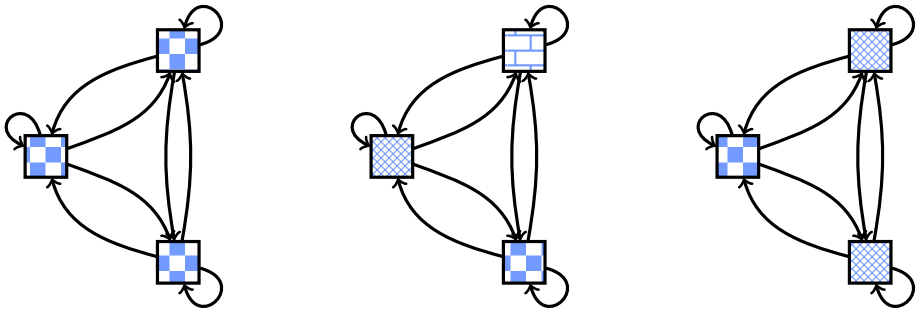


Figure 7.1: Datalengths used for the sleep apnea experiments. The normalised dataset used begins at timestep 22000. 1000 datapoints are used as Washout, followed by 3000 datapoints of Training, and 1000 datapoints of Testing.



(a) single material (b) default mixed material (c) selected materials

Figure 7.2: Three ways of organising the materials. Figure 7.2a shows a restricted homogeneous reservoir with three subreservoirs composed of the same mock material. Figure 7.2b is our "default" heterogeneous reservoir, with each subreservoir a different mock material. Figure 7.2c shows materials that are selected to work best with each dataset. The process of selecting these materials is described in section 7.4.2. A number of approaches to feeding the input into these different reservoirs are possible, as shown in figure 7.4.

- Training: 3000
- Testing: 1000

7.4.2 Material Selection and Base Case Using Single Input

We discuss how the heterogeneous reservoirs are built in this section. Here, we focus solely on the reservoir state and the subreservoirs, as we discuss the inputs and outputs to this reservoir in section 7.4.3.

As we have three mock materials and three datasets, the simplest option is to simply have one subreservoir consisting of each material. We call this our "default"

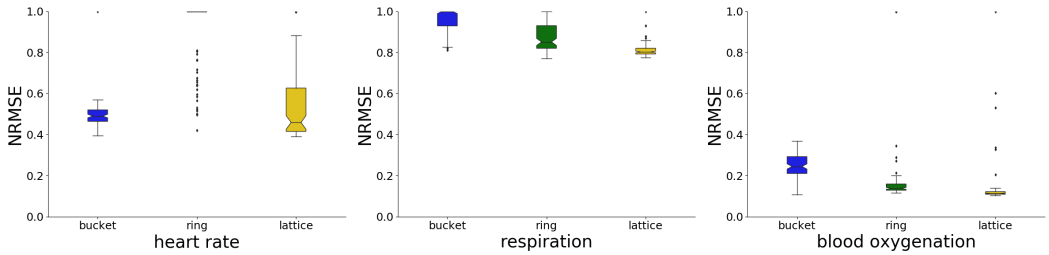


Figure 7.3: NRMSEs over 100 runs of our preliminary experiment using a single input and a single material. These results disregard the "divergent" cases where the NRMSE results are greater than 1, which are summarised in table 7.1.

material selection (fig. 7.2b).

This "default", while a potentially useful benchmark, is chosen entirely arbitrarily. This means that the chances of an improvement in results over a homogeneous reservoir (fig. 7.2a) are not guaranteed. To compensate for this, we also introduce "selected" materials, where each subreservoir is of a material best suited to one of the datasets(fig. 7.2c).

We propose that each material will have parameters that may make it more suited for working with one or another of the datasets in the sleep apnea benchmark. Different datasets may require different behaviours from reservoirs, such as a different Linear Memory Capacity (section 3.8.1) or Generalisation and Kernel Rank (section 3.8.2).

Future work may use a novelty search method such as CHARC [26] to identify which behaviours lead to better performances for which datasets.

In this work, as we have three mock materials with pre-set attributes, we take an empirical approach instead. We run experiments on each dataset as a single-input task using all three mock materials, and report the results in figure 7.3.

The single-input experiments lead to very poor results for the heart rate and respiration datasets. The blood oxygenation dataset appears to fare better, but all three datasets have a number of runs where the NRMSE results are greater than 1, a result achievable by outputting the mean of the dataset. These "divergent" results have been set to 1 in figure 7.3 for legibility, and the number of divergent results is instead reported in table 7.1.

Taking into account first the best mean results and then the fewest divergent cases, we select the following materials for each dataset:

	Bucket	Ring	Lattice
Heart Rate	1	79	15
Respiration Rate	56	17	1
Blood Oxygenation	0	10	2

Table 7.1: Number of "divergent" results (NRMSE more than 1) over 100 runs per material for the Single Input preliminary experiments. These results are used as a secondary basis for the selection of our material, after the best mean. The selected material for each dataset is marked by a box.

- heart rate: bucket
- respiration: lattice
- blood oxygenation: lattice

7.4.3 Input and Output Handling

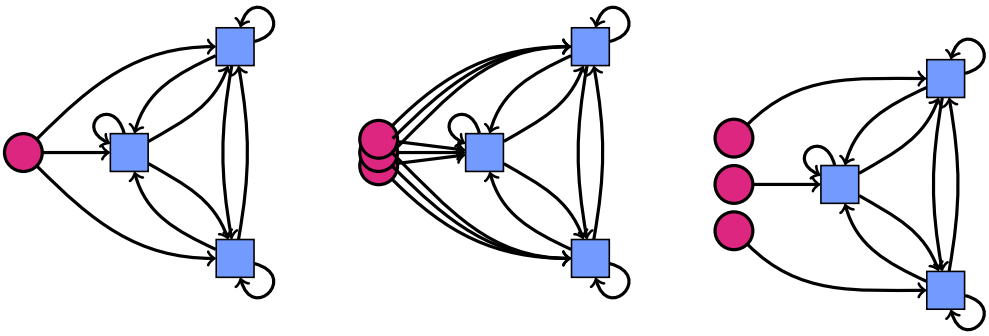
Previous work using the sleep apnea benchmark in Reservoir Computing [136] has treated each of the datasets as entirely separate benchmarks. In this work, we treat all three datasets as one benchmark, with a single restricted reservoir taking three inputs and producing three outputs. This section discusses how these inputs and outputs are handled within our experiments.

Multiple Output Handling

The Reservoir Computing model is particularly well suited to handling multiple outputs, and we take advantage of this attribute in this work. After having run the reservoir, the training phase can be repeated once for each output, producing three trained output vectors $\mathbf{W_v}$ (or equivalently a single $3 \times N$ output matrix $\mathbf{W_v}$). The outputs $\mathbf{v}$ of the reservoir as a whole is then given by

$$\mathbf{v} = \mathbf{W_v x} \tag{7.2}$$

where $\mathbf{x}$ is the state of the reservoir.



(a) single input

(b) all-to-all input mapping

(c) 1-to-1 input mapping

Figure 7.4: A representation of how inputs can be mapped to our different materials. Figure 7.4a illustrates using a single input as used in the single input experiments (section 7.4.2). Figure 7.4b has no input organisation; every input goes to every subreservoir. Finally, figure 7.4c has the inputs separated, with each input mapped to the material we have selected for it.

Organising the Input

With our restricted reservoir model, we consider two options for treating multiple inputs (figure 7.4).

The first option for input handling treats the restricted reservoir as no different from a standard reservoir. As such, every node in the reservoir receives the input, with the input weights $\mathbf{W_u}$ entirely randomly set. In doing so, we treat the reservoir entirely as a black box; any organisation of the input that takes place is entirely unsupervised. This method can be illustrated as an all-to-all mapping of inputs to subreservoirs (fig. 7.4b).

The second option for input handling is to send a given input stream to a single subreservoir in a 1-to-1 mapping (fig. 7.4c). This is interesting in the selected materials case, as we can ensure that our choices of materials per dataset are reflected in the reservoir input.

7.5 Results

We run a total of five distinct experiments, summarised in table 7.2. For each of these experiments, the results for each of the datasets are reported in tables 7.6–7.8. We document any divergences (particularly with the respiration dataset) in table 7.3.

The selected material for the heart rate dataset is the bucket material. With

Experiment	(i)	(ii)	(iii)	(iv)	(v)
Materials used	single material	single material	default mixed materials	selected materials	selected materials
Input organisation	single input	all-to-all	all-to-all	all-to-all	1-to-1

Table 7.2: Material and Input organisation for the experiments described in this section (see also Fig. 7.5). The results for these experiments can be found in figures 7.6–7.8.

Dataset	Experiment				
	(i)	(ii)	(iii)	(iv)	(v)
Heart rate	1	0	0	1	3
Respiration	1	38	43	38	6
Blood Oxygenation	2	1	0	1	3

Table 7.3: Number of divergent results over 100 runs in experiments (i)–(v) for the Single Input preliminary experiments.

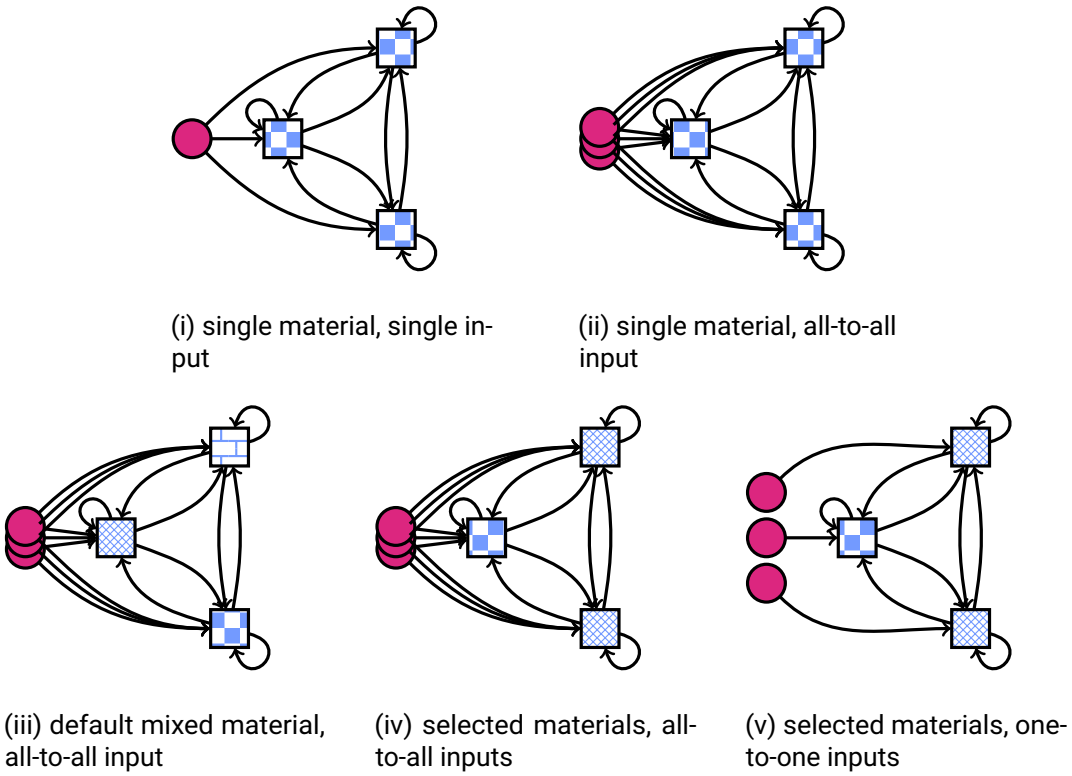
this datasets, figure 7.6 indicates that two mixed-material all-to-all input mapping experiments (iii) and (iv) yield the best results by a significant margin. The single material experiment and selected material 1-to-1 input, however, perform no better than our single-input experiment (i).

The selected material for the respiration dataset is the lattice material. The overall performance of predicting this dataset is particularly poor, and adding the other inputs makes the performance much worse. This is especially true of the selected material experiments, (iv) and (v), which have 13 and 8 divergent cases respectively. However, of the multi-input experiments, experiment (v), the selected materials with 1-to-1 mapping has the best mean, which is still worse than the single-input results.

The poor results in these experiments are most likely due to the noisiness of the data, which may indicate that the best results might be achieved by outputting the mean at every timestep. These results could thus potentially be improved by applying a low-pass filter to the data.

The selected material for the blood oxygenation dataset is the lattice material. Our results for this dataset are by far our best results overall, across all experiments.

As with the heart rate dataset, our most successful experiment overall is the default mixed materials (iii). In this case, it has a similar mean to the single-input



(i) single material, single input

(ii) single material, all-to-all input

(iii) default mixed material, all-to-all input

(iv) selected materials, all-to-all inputs

(v) selected materials, one-to-one inputs

Figure 7.5: Material and Input organisation for the experiments described in this section (see also Table 7.2). The results for these experiments can be found in figures 7.6–7.8

experiment, but no divergences and a much smaller spread of results. Unlike the heart rate results, however, we can observe that the selected materials, 1-to-1 input experiment (iv) does not perform as well as experiment (iii). Instead, as with the respiration results, we can observe some improvement in the selected material experiments by changing from all-to-all input mapping (experiment (iv)) to 1-to-1 input mapping (experiment (v)). That said, even this improved performance is slightly worse than the single input case, as well as the default mixed materials.

7.6 Conclusions

In this work, we do not achieve any consistent improvement across datasets by adding multiple inputs. We do, however, appear to achieve some improvement with the heart rate and respiration datasets by using multiple inputs and heterogeneous reservoirs. Of these heterogeneous reservoir experiments, we achieve our best results with the default mixed materials (experiment (iii)). This may suggest that the

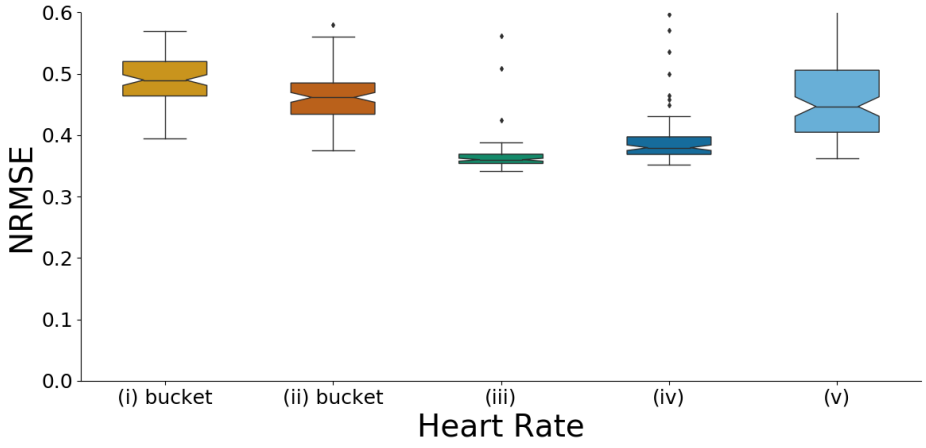


Figure 7.6: Results for our multi-input experiments for the heart dataset, reported as NRMSE over 100 runs. We show the single-input results with the selected material as our comparison, followed by multi-input experiments with the bucket material (our selected material for this dataset), and the selected material heterogeneous reservoir with all-to-all and 1-to-1 input mapping.

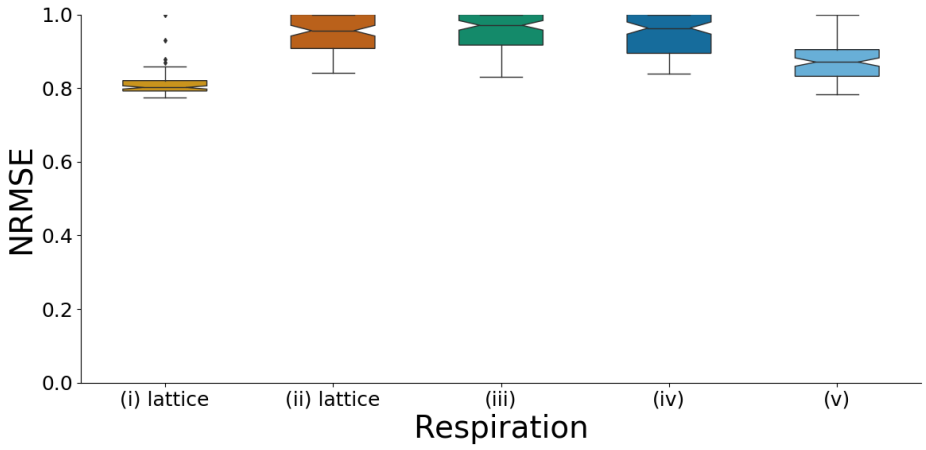


Figure 7.7: Results for our multi-input experiments for the respiration dataset, reported as NRMSE over 100 runs. We show the single-input results with the selected material as our comparison, followed by multi-input experiments with the lattice material (our selected material for this dataset) and the selected material heterogeneous reservoir with all-to-all and 1-to-1 input mapping.

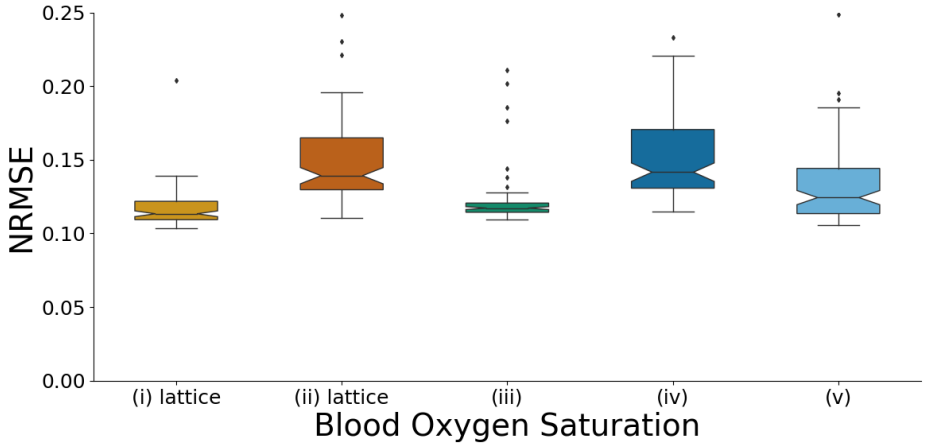


Figure 7.8: Results for our multi-input experiments for the blood oxygenation dataset, reported as NRMSE over 100 runs. We show the single-input results with the selected material as our comparison, followed by multi-input experiments with the bucket material (our selected material for this dataset) and the selected material heterogeneous reservoir with all-to-all and 1-to-1 input mapping.

heterogeneity of the reservoirs may have a greater effect than any matching that takes place for the dataset and the material.

The poor results of the selected material experiments may also be a result of a flaw in our matching process. The mock materials have arbitrarily chosen parameters, chosen primarily to provide a variety of possible parameter sets. This may have resulted in none of our materials being ideal for any of our datasets. Future work may explore this hypothesis by tailoring the parameters of each subreservoir to the dataset directly, through methods such as CHARC [26].

As the results for the different input mapping techniques for the selected materials (experiments (iv) and (v)) are inverted for the heart rate and blood oxygenation, we cannot draw any conclusions on the effectiveness of input mapping or not. It is possible that parameters selected to be a better match to our datasets may shed more light on the matter. There appears to be a slight improvement in the 1-to-1 input mapping in the respiration dataset that resemble the blood oxygen results, but the NRMSE results for this dataset are close to 1, and as such no strong conclusions can be drawn from these results. These results may be improved by denoising the data, potentially by using a low-pass filter. Alternatively, if the data available is of the nasal airflow, using the chest volume dataset may yield better results.

8 Heterogeneous Timescales and Materials

8.1 Introduction

In chapter 5 we introduce the restricted ESN with homogeneous subreservoirs. In chapter 6, we introduce the multi-timescale ESN model. We test this using the MSO* benchmark. In chapter 7, we explore heterogeneous material reservoirs. We test this with the multi-input sleep apnea benchmark. In this chapter, we combine these concepts to explore multi-timescale, multi-material reservoirs.

The main limitation we find chapters 6 and 7 are that the results are fairly poor. However, these reasons for the poor results seem to be distinct.

In chapter 6, we choose timescales randomly without much consideration for how this interacted with the task. Instead, it may be useful to match the timescale to the part of the task that it suits best.

In chapter 7, the subreservoirs appear to be too tightly coupled, and each input becomes additional noise to the others. Without using multiple inputs, however, we find that there is not enough information to effectively predict the next step.

In both cases, adding heterogeneity may help us improve our results, which is what we explore in this chapter.

Infobox 8.1: Concepts Studied in Chapter

Heterogeneous timescale, heterogeneous material reservoirs.

8.2 Rationale

In this chapter, we compare many of the models introduced in this thesis so far, and a heterogeneous material and timescale reservoir. We evaluate a number of models using the MSO*3 and sleep apnea benchmarks.

8.2.1 Reservoir Models

The models we compare across, as well as the rationale for using them, is presented in this section. Section 8.3 details the experimental setup, including the parameters used for the different models.

To provide a starting point and basis for comparison to the general literature, the first model we use is a classical ESN. We also compare against a patch-consistent restricted ESN with homogeneous subreservoirs, as introduced in chapter 5.

The remainder of our experiments are performed on our mock material reservoirs, which we introduce in chapter 6. All of our experiments with the mock materials so far have involved three subreservoirs. In this chapter, we additionally provide an analogy to the classical ESN using mock materials. To do this, we create a mock material reservoir consisting of a single subreservoir, the size of which is equal to the total size of our classical ESN. This single large reservoir is to our mock material multi-subreservoir ESNs what the classical ESN is to the restricted ESN.

We re-introduce the homogeneous material, heterogeneous timescale reservoir from chapter 6, as well as the heterogeneous material, homogeneous timescale reservoirs from chapter 7.

Finally, we combine the two types of heterogeneity studied with a heterogeneous timescale, heterogeneous material reservoir.

With three mock materials, comparing every model with every material is infeasible. Instead, for each task and model, we select the material with the best performance. The preliminary experiments we use to determine which material we use are discussed in section 8.2.5.

8.2.2 Input Mapping

In chapter 7, we observe that different ways of mapping multiple inputs to multiple reservoirs can have an effect on the performance of the reservoir. We study

two types of input mapping: all-to-all mapping, where each input is transmitted to each subreservoir, and one-to-one mapping, where each input is transmitted to a single subreservoir. One-to-one mapping allows us to match each input so that it is transmitted to the subreservoir it is best suited to. In this chapter, we continue to study both input mapping techniques: In the reservoir models where the distinction is possible, our experiments are run for both all-to-all and one-to-one input mapping.

As a multi-input task, the sleep apnea benchmark naturally lends itself to exploring different types of input mapping. MSO*, however, has a single input. As such, we modify it slightly in order to allow for different types of input mapping. We discuss this in section 8.2.3.

Having one-to-one input mapping allows us to match each input to the subreservoir it is best suited to. This could be because of the material of the subreservoir, or the timescale it is operating on. We discuss how we match the inputs to subreservoirs in sections 8.2.4 and 8.2.5.

8.2.3 Multi-Input MSO*

In chapter 7, we establish that we can tailor specific subreservoirs to different parts of a task for better performance. We do this by mapping each input of a multi-input task to the subreservoir it is best matched to.

Unlike the sleep apnea benchmark, the MSO* task does not have multiple inputs. As such, simply matching each input to a distinct subreservoir is not possible without modifying the task.

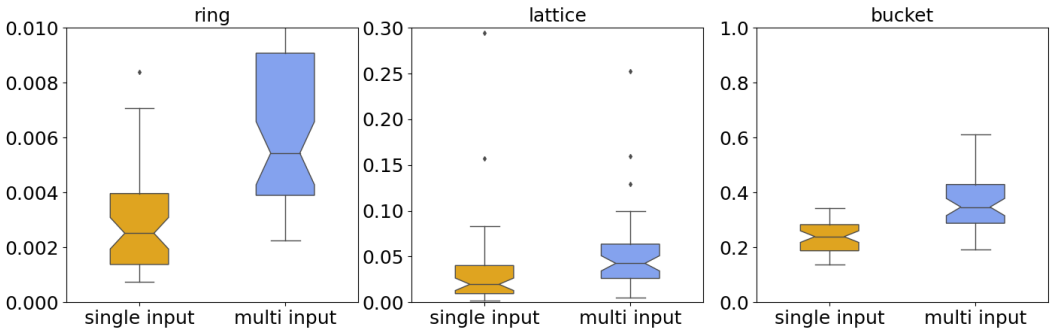
We perform this modification by separating out the terms of the MSO* equation. The single-input MSO* equation (eqn. 5.14) is repeated here for convenience:

$$y^*(t) = \sum_{i=1}^{n} \sin\left(\frac{\alpha_i t}{8}\right) \tag{8.1}$$

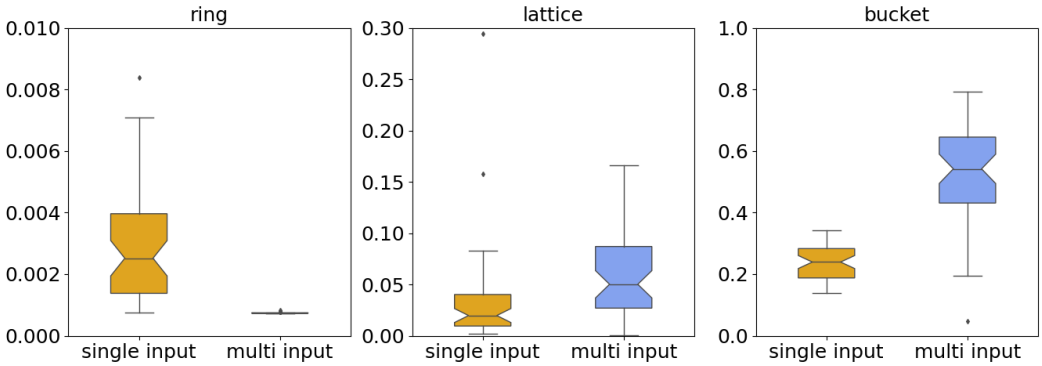
where t is the timestep, and $\alpha = [0.2, 0.311, 0.42, 0.51, 0.63, 0.74, 0.85, 0.97]$.

To separate the inputs, we merely take each $\sin\left(\frac{\alpha_i t}{8}\right)$ term and input it to the reservoir separately. Unlike the sleep apnea task, we do not have separate outputs. Instead, the task of the reservoir is still to predict the *sum* of the terms at time $t+1$.

As we have three subreservoirs, we need three inputs. As such, we use MSO*3 in these experiments.



(a) all-to-all mapping for multi-input MSO* 3



(b) one-to-one mapping for multi input MSO* 3

Figure 8.1: single vs multi-input MSO*, using both all-to-all (fig a) and one-to-one (fig b) mapping for the multi-input case.

Separating the different terms may have one or both of two effects on the task performance:

- The separation acts as preprocessing, making the task easier,
- the additional step of summing the terms makes the task harder.

To see the effect of separating the inputs and to compensate for it if needed, we look at the difference between the single and multi input MSO* 3 (see fig. 8.1). We study both all-to-all and one-to-one mapping, in order to provide a baseline for this chapter's experiments.

We find for the most part that we get worse inputs with the multi-input version of the task, with the exception of the ring material with one-to-one mapping. From this, we conclude that the separation into multiple inputs alone does not lead to an easier task.

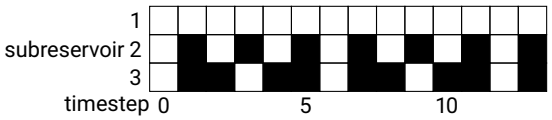


Figure 8.2: Sleep rhythms used for the multiple timescale experiment. At every timestep, if the subreservoir is awake, the square is white, and if it is asleep, the square is black.

8.2.4 Choosing Timescales

In this chapter, we continue to use pre-determined timescales instead of designing each timescale to be most appropriate for each input. We use the same timescales in this chapter as we use in chapter 6, illustrated in fig 8.2. Once again, we use these timescales as the simplest possible case of three distinct timescales. This will allow us to ascertain whether more complicated timescales, such as ones directly matched to the data, bear investigating.

Unlike chapter 6, our use of two multi-input tasks allow us to match each input to a specific timescale.

MSO*3: We match the term with the slowest dynamics ($\sin\left(\frac{0.2t}{8}\right)$) to the slowest reservoir, and the term with the fastest dynamics $\sin\left(\frac{0.42t}{8}\right)$ to the fastest reservoir.

Sleep Apnea: There are two potential approaches we can take to matching the sleep apnea inputs to different timescales. The first approach is to match the input with the fastest dynamics (respiration) and match it to the fastest subreservoir, and match the input with the slowest dynamics (blood oxygenation) to the slowest reservoir. This is the most intuitive approach, and is the approach we take with MSO*3.

The second possible approach addresses the observation we make in chapter 7 with regards to the noisiness of the respiration dataset. By matching that input to the slowest reservoir, we could attempt to reduce the noise by subsampling this input.

In order to choose the best approach, we perform a number of preliminary experiments, the results of which can be found in appendix C. We conclude that the approach of matching the fastest reservoir to the respiration leads to the best results.

Reservoir type	MSO*3	Sleep Apnea
SM large reservoir	Ring	Ring
SM & ST (A)	Ring	Lattice
SM & ST (1)	Ring	Lattice
SM & MT (A)	Ring	Lattice
SM & MT (1)	Bucket	Lattice
MM & ST (A)	R/B/L	B/R/L
MM & ST (1)	R/B/L	B/R/L
MM & MT (A)	R/B/L	B/R/L
MM & MT (1)	R/B/L	L/R/B

Table 8.1: Materials chosen for each reservoir model. Abbr: SM: Single material, MM: multi material, ST: single timescale, MT: multi timescale, (A): all-to-all mapping, (1):1-to-1 mappings, R: Ring, B: Bucket, L: Lattice. In the multi-material, multi-timescale (MM & MT) experiment, the materials are listed from fastest to slowest.

8.2.5 Choosing Materials

In chapter 7, we select the material for the individual subreservoir by evaluating the input individually against all three materials.

In this chapter, we are not only using multiple materials, but multiple timescales as well. Each of our mock materials have a distinct sleep mode (table 8.2), and we find in chapter 6 that the sleep mode used has an effect on the performance of the reservoir overall.

As such, we perform experiments for each model using all three inputs for every material, or combination of materials. To reduce the number of total experiments, we constrain multi-material reservoirs by having one subreservoir of each material, with the only difference being the ordering and input mapping of the reservoirs.

The results for these preliminary experiments can be found in appendix C. When comparing the different models to each other, we compare only the results using the best performing material. The material or combination of materials used for each model and task is summarised in table 8.1.

8.3 Experimental Setup

8.3.1 Reservoir Parameters

All the reservoirs we evaluate have 192 nodes. In models with subreservoirs, there are three subreservoirs of 64 nodes each.

material	ring	lattice	bucket
topology	ring	lattice	fully connected
spectral radius	1	1	0.8
sleep mode	total sleep	input sleep	output sleep
input comms	1 node	8 nodes	4 nodes
output comms	1 node	8 nodes	4 nodes
input comms = output comms?	False	True	True

Table 8.2: parameters for the three mock materials, summarised from chapter 6.

The standard ESN is randomly initialised with a density of 0.1, and with the spectral radius of the weight matrix normalised to 1. The restricted ESN has three subreservoirs with a density of 0.1, with the density of the weights between the subreservoirs (D_B) set to 0.025. The spectral radius of the weight matrix is normalised to 1.

The single material large reservoir is created using the materials of a mock material, but with a single 192-node subreservoir.

The properties of mock materials are summarised in table 8.2. The off-diagonals of the mock material reservoirs are normalised to 1, using the process outlined in chapter 6.4.1.

8.3.2 Evaluation

We evaluate our models with the multi-input MSO*3 and the sleep apnea benchmarks. These benchmarks are introduced in 3, and developed in chapters 6 and 7. We use the same datalengths and input normalisation in this chapter as we do in previous chapters.

We record the NRMSE of the output over 50 runs. Where a sleep apnea task run gives a divergent result (NRMSE more than 1), it is set to 1; the number of divergent results is recorded in tables 8.3 and 8.4.

8.4 Results

We report our results in figures 8.3–8.8. The divergent results for the sleep apnea experiments are reported in tables 8.3 and 8.4. The full results can be found in appendix C.

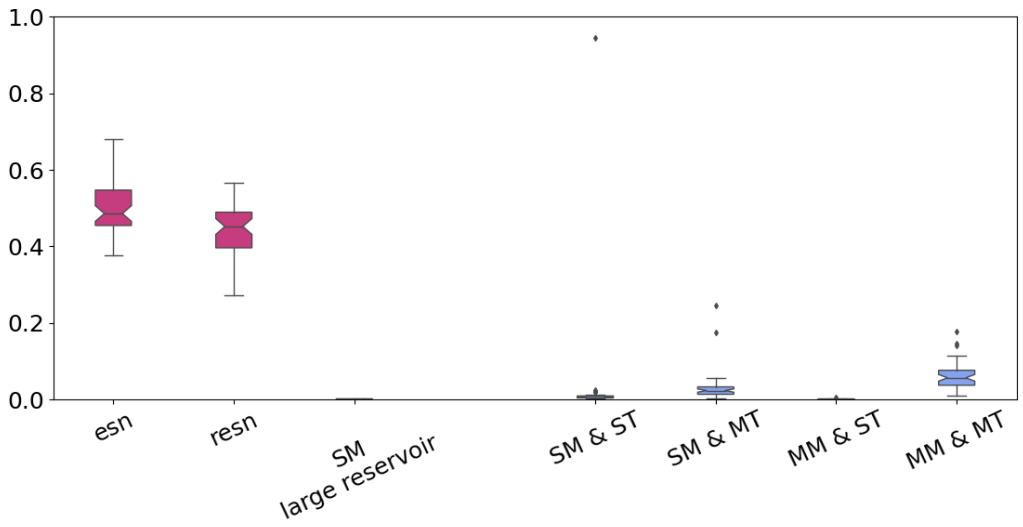


Figure 8.3: MSO all-to-all experiments. For abbr., see table 8.1

For legibility, we split up our one-to-one and all-to-all results into separate graphs. Both graphs contain the ESN, rESN, and single large reservoir results as well.

8.4.1 MSO*3 Results

The results for our MSO*3 experiments are reported in figures 8.3–8.5. As the results for the ESN and rESN are much poorer than the all-to-all results, we report the all-to-all results both alongside the ESN, rESN, and large reservoir (fig. 8.3) and without them (fig. 8.4).

The ESN and rESN results for the MSO* task are worse than the ones we find in chapter 5. This is likely due to both the use of the use of multi-input MSO* instead of single-input, as well as the normalisation of the ESN and rESN weight matrices to to have a spectral radius of 1 rather than a largest singular value of 1.

We find that the reservoirs with the all-to-all mapping performs far better in the MSO*3 task than the reservoirs with the one-to-one mapping. Unlike the sleep apnea task, the MSO* task does not have three outputs. Instead, the outputs are summed. A further subreservoir performing this summation may lead to improved results.

Of the all-to-all experiments, the reservoir with the best results is the multi-material, single timescale reservoir. The results for the one-to-one experiments are for the most part very poor, with the exception of the single material, single timescale experiment. Again, we find that the heterogeneity, with this task, worsens

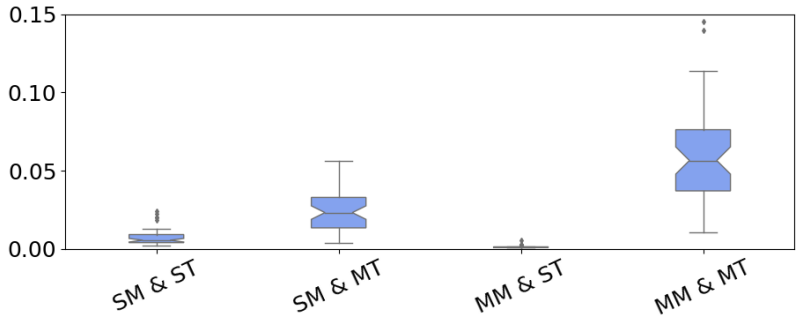


Figure 8.4: MSO all-to-all experiments without the base cases. For abbr., see table 8.1

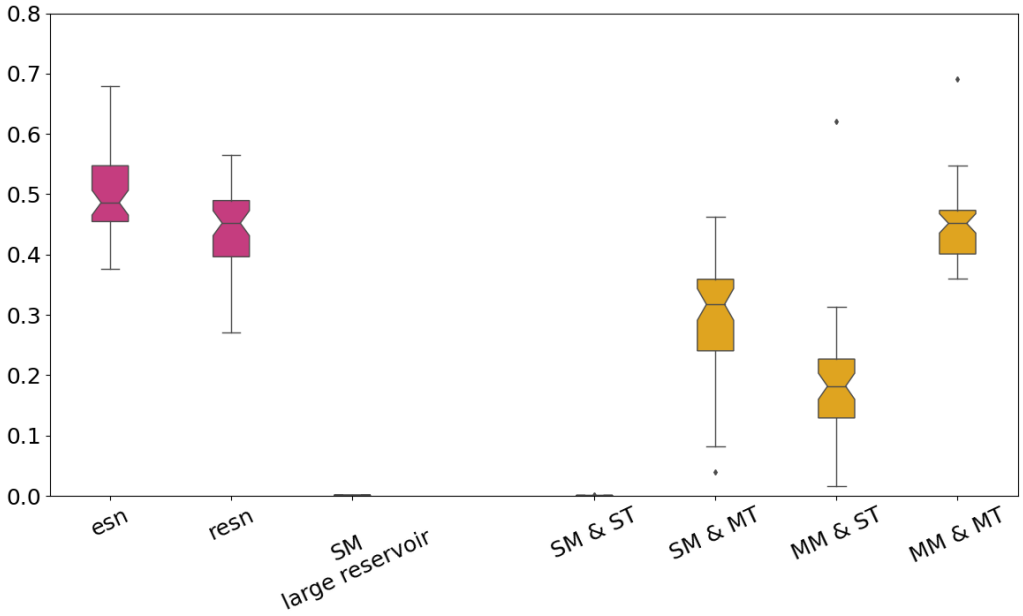


Figure 8.5: MSO one-to-one experiments. For abbr., see table 8.1

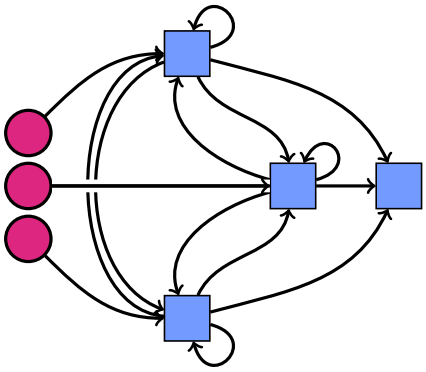


Figure 8.6: an alternate configuration proposed for the MSO*3 task. An additional subreservoir is added so that the outputs can be combined outside of the trained output layer.

Reservoir type	heart	chest	blood
Standard ESN	0	50	0
Restricted ESN	0	5	0
SM large reservoir	3	25	1
SM & ST (A)	5	21	0
SM & MT (A)	0	17	0
MM & ST (A)	0	23	0
MM & MT (A)	0	28	0

Table 8.3: Number of divergent results ($NRMSE > 1$) over 50 runs of the reservoirs in the all-to-all sleep apnea experiments. For abbr., see table 8.1

the results rather than improves them.

Alternate restricted reservoir topologies may lead to better results for the one-to-one multi-input MSO* task, such as adding an additional subreservoir for the purpose of summing the outputs (fig. 8.6).

8.4.2 Sleep Apnea

The results for the Sleep Apnea task are reported in figures 8.7 and 8.8. Where the NRMSE of the results is divergent, these have been set to 1 and reported in tables 8.3 and 8.4.

The results of the all-to-all experiments are very poor, with a high number of divergences for the heart dataset. This is in keeping with our prior results, but not in keeping with the MSO* results.

Unlike the MSO results, however, this task has distinct outputs, and it is likely that

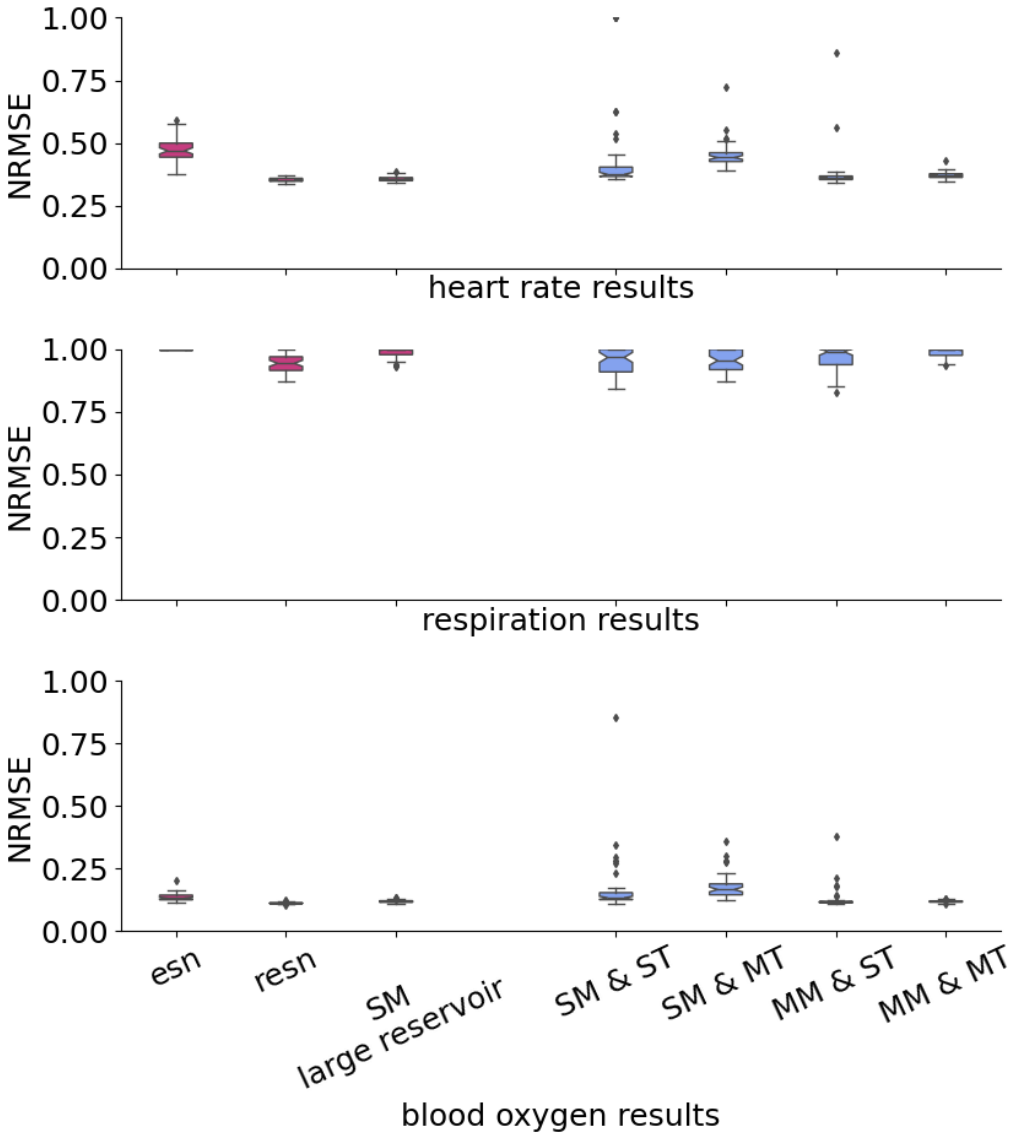


Figure 8.7: Sleep Apnea, all-to-all experiments. For abbr., see table 8.1

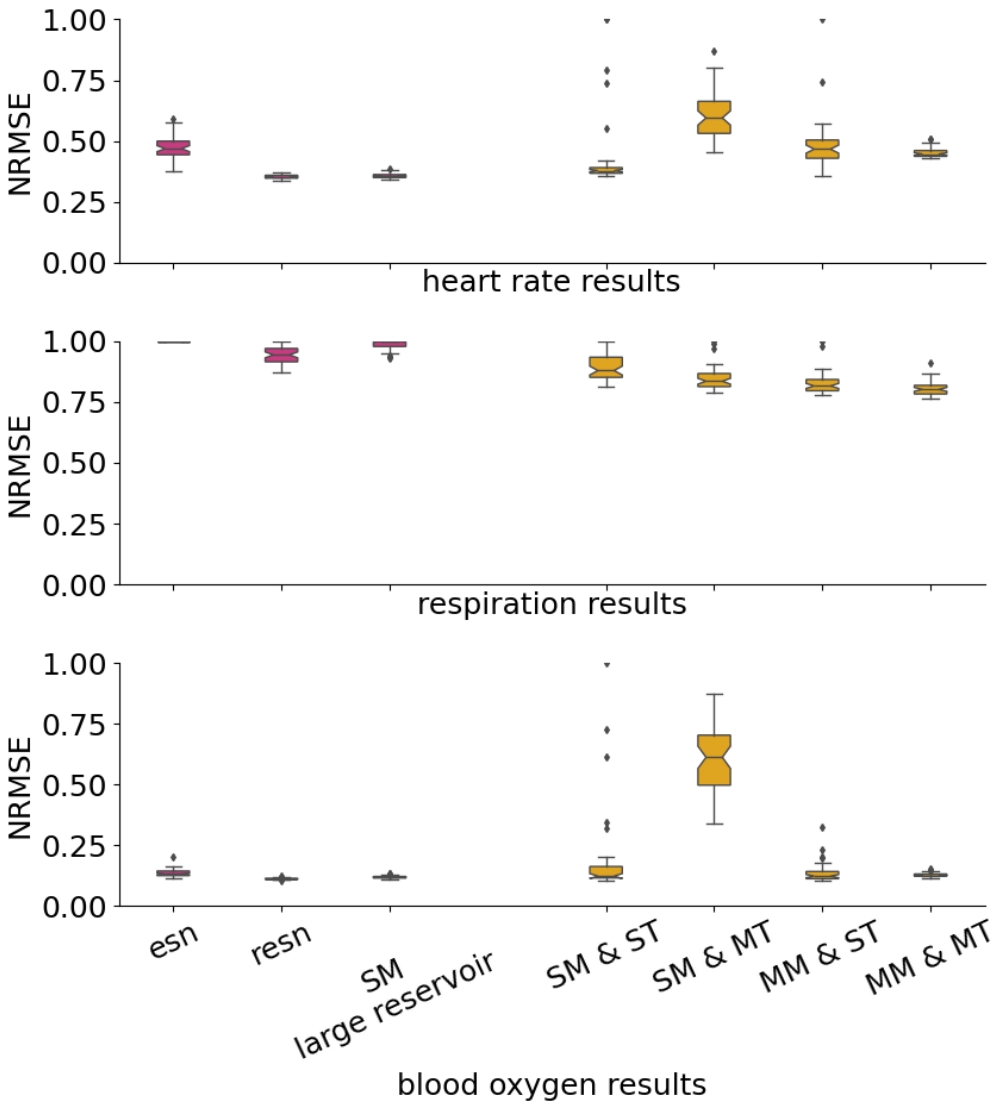


Figure 8.8: Sleep Apnea one-to-one experiments. For abbr., see table 8.1

when the inputs are too tightly coupled, they act as noise to each other.

The results from the sleep apnea one-to-one experiments are far more promising. In the multi-material and multi-timescale experiments both, we have a much lower number of divergences overall. In the multi-timescale, single material case, we have no divergences at all, it also has the worst results for blood oxygenation and heart rate over all experiments.

The multi-material, single timescale reservoir blood oxygenation results are better than the results for the multi-timescale, single material reservoir. However, this experiment still has a small number of divergent results.

The multi-material, multi-timescale experiment not only has no divergences at

Reservoir type	heart	chest	blood
Standard ESN	0	50	0
Restricted ESN	0	5	0
SM large reservoir	3	25	1
SM & ST (1)	2	8	1
SM & MT (1)	0	0	0
MM & ST (1)	1	1	0
MM & MT (1)	0	0	0

Table 8.4: Number of divergent results ($NRMSE > 1$) over 50 runs of the reservoirs in the one-to-one sleep apnea experiments. For abbr., see table 8.1

all, it also has lower (and thus better) NRMSEs in all three datasets. As such, we conclude that both the temporal and the material heterogeneity contribute to these better results.

The better results we find in the multi-timescale experiments may indicate that by using multiple timescales, we increase the memory of the system. A system with a longer memory would be more suitable for the longer timescales of the sleep apnea dataset, leading to these better results.

8.5 Conclusions

The use of heterogeneous reservoirs has allowed us to get acceptable results on a task which is otherwise very difficult for a reservoir computer, the sleep apnea benchmark. We see in this case that both the temporal and material heterogeneity are necessary for these results.

We find, however, that these results can not be generalised across every reservoir computing task. Some tasks, even those that involve multiple different parts, need less heterogeneity, and more communication between the subreservoirs. This can be achieved, to an extent, with an all-to-all input mapping, but more research is needed to see if there are better ways to design heterogeneous reservoirs for this style of task.

We find that heterogeneous reservoirs where multiple inputs are fed in to specific subreservoirs using one-to-one input mapping is helpful for decoupling inputs from one another.

In tasks where heterogeneity may improve the results, the process we develop in

this chapter can be used in order to best choose which subreservoir is best suited to which input. We find that we can improve the results even with predetermined parameters and timescales, as long as the mapping of these to the inputs is judiciously chosen.

The sleep apnea task is an ideal choice for heterogeneous reservoirs, as it consists of three datasets that are related but distinct, and that have dynamics on different timescales. It is also notable for being a particularly difficult task.

Future work will focus on finding other tasks that are better suited to heterogeneous reservoirs than standard ones, such as audio classification with inputs at different frequencies, control tasks that rely on positional input from sensors, or other tasks based on multiple channels of physiological data.

9 Conclusions

9.1 Summary of Findings

In this thesis, we study how we can scale up Reservoir Computers by combining multiple smaller reservoirs together. In doing this, we aim to tackle more complex tasks by combining reservoirs with heterogeneous properties. These might be reservoirs that operate on multiple distinct timescales, or reservoirs made by different materials.

In chapter 5, we find that restricted reservoirs are as effective as larger reservoirs with similar properties, for simple tasks. This supports the hypothesis that we can combine multiple smaller reservoirs to create larger reservoirs. Unfortunately, this does not necessarily hold as tasks get more complex.

In chapter 6, we introduce and study multi-timescale reservoirs. These multi-timescale reservoirs raise a number of questions, but we find that we cannot replace a single-timescale reservoir with a multi-timescale one without consideration to task design, or to how the timescales might change the behaviour of the reservoir.

In chapter 7, we look at multi-material reservoirs using different combinations of "mock materials", a concept which we introduce in chapter 6. We use this multi-material reservoir to work with a complex, multi-input task that cannot be solved with a standard ESN: the sleep apnea benchmark. We find that multi-material reservoirs are a promising avenue for solving tasks with multiple distinct parts. That said, we also find that without an additional layer of separation between our inputs, different parts of a task can act as additional noise to each other.

In chapter 8, we combine the work of the previous two chapters to build a multi-timescale, multi-material reservoir. We compare the performance of our benchmark tasks, MSO* and sleep apnea, across all of the models studied in the thesis, and

Infobox 9.1: Research Questions

How can we scale up reservoirs by combining multiple smaller reservoirs together?

1. How can we combine reservoirs that operate on different timescales in order to improve performance on multi-timescale tasks?
2. How can we combine reservoirs with different qualities in order to explore more complex tasks?
3. How can we design a heterogeneous reservoir that is specifically adapted to a difficult task, using multiple timescales and materials?

work towards designing a heterogeneous reservoir that can get the best results.

Our findings in this final technical chapter are twofold: We are able to tailor a multi-timescale, multi-material reservoir such that we are able to get promising results for the sleep apnea task. Even with the constraints we place on the materials and timescales used, our use of heterogeneity leads to far better results for this task than we get out of a standard ESN.

We also find, however, that the MSO* task performs poorly on our heterogeneous reservoirs, as the task itself is not well suited to heterogeneity. We propose alternate reservoir designs which may be more effective for solving this task (fig. 8.6).

9.2 Thesis Contributions

The findings of this thesis allow us to answer the research questions posed in chapter 4, which we repeat in infobox 9.1.

With our restricted reservoirs, we introduce a way to describe combinations of reservoirs, which allows us to study how we scale up reservoirs with combinations of subreservoirs.

With the mock materials that we introduce, we are able to simulate constraints we may have when working with existing physical materials, which cannot always be tuned as an ESN might. As such, we are able to bridge the gap between ESNs and physical instantiations of reservoirs, and answer question (1).

The restricted reservoir model also allows us to formalise how we describe the structure of combinations of reservoirs. With this, we are able to design a model for multi-timescale ESNs, where each subreservoir operates at a different rhythm. This allows us to answer question (2).

We use both material and temporal heterogeneity to work towards solving the complex sleep apnea benchmark, by using different arts of the reservoir to work on different parts of the task.

In doing this, we outline a process for designing reservoirs for these more complex tasks. However, we find that different styles of task such as MSO* may need differently structured reservoirs in order to properly exploit heterogeneity.

We also find that using one-to-one mapping to different subreservoirs allows us to break up complex tasks to different parts of a reservoir.

By defining this process and proposing extensions to it, we are able to propose an answer to question (3).

This thesis introduces multiple concepts of heterogeneous reservoirs that can further the development of multi-reservoir systems. We are able to use these concepts to answer our research questions, and through this, we gain an answer to our overarching question, and are able to use our techniques to scale up a reservoir computer. We are thus able to get closer to solving a complex task, the sleep apnea benchmark.

We additionally contribute to the field through an extensive review of the benchmarks used in Reservoir Computing. In doing so, we introduce a paradigm through which to view existing benchmarks, as well as some guidelines for creating new ones.

9.3 Future Work

Our findings open the door to a number of avenues for future research. While our work helps bridge the gap between simulation and the physical, our processes still rely strongly on the ability to run a high number of experiments in a short amount of time. The technique needs refining before being fully applicable to physical reservoirs.

Once we have this extension of this technique, future research should focus on whether we can replicate our results using physical reservoir instantiations.

More research is also needed on how we might use this technique for other multi-input tasks, such as classification tasks like speech recognition or sleep apnea detection, or control tasks such as the Van der Pol oscillator task (section 3.4.4, [45, 46]).

This work may also tie in to designing the architecture of a heterogeneous reservoir to best suit the task. Along with experimenting on which materials and timescales are used and how the inputs are mapped to subreservoirs, further research may consider:

- how the connections between the subreservoirs may affect different parts of a task.
- whether we can mix one-to-one mapping with all-to-all mapping by having more subreservoirs than we have inputs.
- how distinct timescales may affect the connections between subreservoirs.

9.4 Multi-timescale Reservoirs and Spectral Radii

Another topic that we touch on in this thesis is the effect of spectral radius normalisation has on the performance of a reservoir overall.

The spectral radius of an ESN's weight matrix has been established to be an important factor of reservoir computing since the earliest work in the field [63]. The common wisdom [87] holds that a spectral radius that is normalised to less than 1 ensures the Echo State Property, essential for Reservoir Computing.

While this is an oversimplification that is studied at greater length by Jaeger [66], the value to which the spectral radius is normalised certainly has an effect on the behaviour of a Reservoir Computer.

This becomes particularly relevant to our work due to some properties of block diagonal matrices, which our restricted reservoir resembles greatly. The spectral radius of a matrix is equal to its largest absolute eigenvalue. In a block diagonal matrix, the eigenvalues of the matrix are the set of eigenvalues of each individual block. As such, the spectral radius of the matrix overall is the largest of the eigenvalues of the blocks.

This being the case, our multi-timescale reservoirs open the possibility of having different spectral radii at different timesteps:

Our bucket material has a spectral radius set to 0.8, so a restricted reservoir composed only of buckets in parallel will have an overall spectral radius of 0.8. However, if one subreservoir of several is asleep, the weight matrix of that subreservoir is the identity matrix, with a spectral radius of 1. As such, the spectral radius of the reservoir as a whole is raised to 1.

Of course, when the subreservoirs are not run in parallel, and are instead connected to each other, the off-diagonal values of the edges between subreservoirs may have an effect on the reservoir's spectral radius overall.

Again, however, we can use multiple timescales to manipulate the changes to the spectral radius. Off-diagonal edges only contribute to changes to the spectral radius if they interact with each other or the blocks in certain ways. With sleep states, we can ensure that these off-diagonals do not always exist at the same time, and limit these interactions. As such, we can exploit the multiple timescales in order to stabilise the spectral radius over time.

More research should be conducted on how one might best exploit the interaction of multi-timescale reservoirs with the spectral radius of a restricted reservoir.

References

[1] Dan A. Allwood et al. “A perspective on physical reservoir computing with nanomagnetic devices”. In: *Applied Physics Letters* 122.4 (Jan. 2023), p. 040501. DOI: 10.1063/5.0119040. URL: https://doi.org/10.1063%5C%2F5.0119040.

[2] AMS. “Persistence forecast”. In: *Glossary of Meteorology* (2012). https://glossary.ametsoc.org/wiki/Persistence_forecast, accessed 2024-03-20.

[3] Antoine Dejonkheere, Francois Duport, Anteo Smerieri, Li Fang, Jean-Louis Oudar, Marc Haelterman, Serge Massar. “All-optical reservoir computer based on saturation of absorbtion”. In: *Optical Express* 22.9 (2014), pp. 10868–10881.

[4] L Appeltant et al. “Information processing using a single dynamical node as complex system”. en. In: *Nat. Commun.* 2 (Sept. 2011), p. 468.

[5] A F Atiya and A G Parlos. “New results on recurrent network training: unifying the algorithms and accelerating convergence”. en. In: *IEEE TNN* 11.3 (2000), pp. 697–709.

[6] Levy Boccato et al. “An echo state network architecture based on volterra filtering and PCA with application to the channel equalization problem”. In: *The 2011 International Joint Conference on Neural Networks*. July 2011, pp. 580–587.

[7] Levy Boccato et al. “An extended echo state network using Volterra filtering and principal component analysis”. en. In: *Neural Netw.* 32 (Aug. 2012), pp. 292–302.

[8] Simon Brodeur and Jean Rouat. “Regulation toward Self-organized Criticality in a Recurrent Spiking Neural Reservoir”. In: *Proceedings of the 22nd international conference on Artificial Neural Networks and Machine Learning - Volume Part I*. Vol. 7552. unknown, Sept. 2012, pp. 547–554.

[9] Jason Brownlee. *7 Time Series Datasets for Machine Learning [dataset]*. en. https://machinelearningmastery.com/time-series-datasets-for-machine-learning/. Accessed: 2022-4-6. Nov. 2016.

[10] Daniel Brunner et al. “Parallel photonic information processing at gigabyte per second data rates using transient states”. en. In: *Nat. Commun.* 4 (2013), p. 1364.

[11] Lars Büsing, Benjamin Schrauwen, and Robert Legenstein. “Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons”. In: *Neural Computation* 22.5 (2010), pp. 1272–1311. DOI: 10.1162/neco.2009.01-09-947.

[12] John Butcher et al. *Extending reservoir computing with random static projections: a hybrid between extreme learning and RC*. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.226.6672&rep=rep1&type=pdf. Accessed: 2021-11-19.

[13] John B Butcher et al. "Extending reservoir computing with random static projections". In: *ESANN 2010*. 2010, pp. 303–308.

[14] K. Caluwaerts et al. "Locomotion Without a Brain: Physical Reservoir Computing in Tensegrity Structures". In: *Artificial Life* 19.1 (2013), pp. 35–66.

[15] Daniel Canaday, Andrew Pomerance, and Daniel J Gauthier. "Model-free control of dynamical systems with deep reservoir computing". en. In: *J. Phys. Complex.* 2.3 (2021), p. 035025.

[16] Michal Čerňanský and Peter Tiňo. "Predictive Modeling with Echo State Networks". In: *Artificial Neural Networks - ICANN 2008*. Springer Berlin Heidelberg, 2008, pp. 778–787.

[17] Kyriakos Chatzidimitriou and Pericles Mitkas. "A NEAT way for evolving Echo State Networks". In: vol. 215. Jan. 2010, pp. 909–914. DOI: 10.3233/978-1-60750-606-5-909.

[18] Martin Trefzer Chester Wringe and Susan Stepney. "Reservoir computing benchmarks: a tutorial review and critique". In: *International Journal of Parallel, Emergent and Distributed Systems* 0.0 (2025), pp. 1–39. DOI: 10.1080/17445760.2025.2472211. eprint: https://doi.org/10.1080/17445760.2025.2472211. URL: https://doi.org/10.1080/17445760.2025.2472211.

[19] J. T. Connor, R. D. Martin, and L. E. Atlas. "Recurrent neural networks and robust time series prediction". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 240–254. DOI: 10.1109/72.279188.

[20] Jerome Connor, Les E. Atlas, and Douglas R. Martin. "Recurrent networks and NARMA modeling". In: *Proceedings of the 4th International Conference on Neural Information Processing Systems*. NIPS'91. Morgan Kaufmann, 1991, pp. 301–308.

[21] Matteo Cucchi et al. "Reservoir computing with biocompatible organic electrochemical networks for brain-inspired biosignal classification". en. In: *Sci Adv* 7.34 (2021).

[22] Matteo Cucchi et al. "Hands-on reservoir computing: a tutorial for practical implementation". In: *Neuromorphic Computing and Engineering* 2.3 (2022), p. 032002. DOI: 10.1088/2634-4386/ac7db7.

[23] Matthew Dale. "Neuroevolution of hierarchical reservoir computers". In: *GECCO 2018*. Kyoto, Japan: ACM, 2018, pp. 410–417.

[24] Matthew Dale. "Reservoir Computing in Materio". en. PhD thesis. University of York, 2018.

[25] Matthew Dale et al. "Evolving Carbon Nanotube Reservoir Computers". en. In: *UCNC 2016*. Vol. 9726. LNCS. Springer, 2016, pp. 49–61.

[26] Matthew Dale et al. "A substrate-independent framework to characterize reservoir computers". en. In: *Proc. Math. Phys. Eng. Sci.* 475.2226 (June 2019), p. 20180723.

[27] Matthew Dale et al. “Computing with Magnetic Thin Films: Using Film Geometry to Improve Dynamics”. In: *UCNC 2021*. Vol. 12984. LNCS. Springer, 2021, pp. 19–34.

[28] Matthew Dale et al. “Reservoir computing quality: connectivity and topology”. In: *Nat. Comput.* 20.2 (June 2021), pp. 205–216.

[29] Matthew Dale et al. “Reservoir Computing with Thin-film Ferromagnetic Devices”. In: *CoRR* abs/2101.12700 (2021). arXiv: 2101.12700. URL: https://arxiv.org/abs/2101.12700.

[30] Joni Dambre et al. “Information processing capacity of dynamical systems”. en. In: *Sci. Rep.* 2 (July 2012), p. 514.

[31] Zhidong Deng and Yi Zhang. “Collective behavior of a small-world recurrent neural system with scale-free distribution”. en. In: *IEEE TNN* 18.5 (2007), pp. 1364–1375.

[32] Guillaume Dion, Salim Mejaouri, and Julien Sylvestre. “Reservoir computing with a single delay-coupled non-linear mechanical oscillator”. In: *J. Appl. Phys.* 124.15 (Oct. 2018), p. 152132.

[33] Guillaume Dion et al. “In-sensor human gait analysis with machine learning in a wearable microfabricated accelerometer”. In: *Communications Engineering* (2024).

[34] Elisa Donati et al. “Processing EMG signals using reservoir computing on an event-based neuromorphic system”. In: *BioCAS 2018*. IEEE, 2018, pp. 1–4.

[35] Chao Du et al. “Reservoir computing using dynamic memristors for temporal information processing”. en. In: *Nat. Commun.* 8.1 (Dec. 2017), p. 2204.

[36] François Duport et al. “All-optical reservoir computing”. en. In: *Opt. Express* 20.20 (Sept. 2012), pp. 22783–22795.

[37] Aya N Elbedwehy, Mohy Eldin Abo-Elsoud, and Ahmed Elnakib. “ECG Denoising using a Single-Node Dynamic Reservoir Computing Architecture”. In: *Mansoura Engineering Journal* 46.4 (2021), pp. 47–52.

[38] J. Doyne Farmer. “Chaotic attractors of an infinite-dimensional dynamical system”. In: *Physica D. Nonlinear phenomena* 4.3 (1982), pp. 366–393. DOI: 10.1016/0167-2789(82)90042-2.

[39] Chrisantha Fernando and Sampsa Sojakka. “Pattern Recognition in a Bucket”. In: *Advances in Artificial Life*. Springer, 2003, pp. 588–597.

[40] Pascal Fries. “Rhythms for cognition: communication through coherence”. In: *Neuron* 88.1 (2015), pp. 220–235.

[41] Keisuke Fujii and Kohei Nakajima. “Harnessing Disordered-Ensemble Quantum Dynamics for Machine Learning”. In: *Phys. Rev. Applied* 8.2 (Aug. 2017), p. 024030.

[42] Claudio Gallicchio and Alessio Micheli. “Architectural and Markovian factors of echo state networks”. en. In: *Neural Netw.* 24.5 (2011), pp. 440–456.

[43] Claudio Gallicchio and Alessio Micheli. “Echo State Property of Deep Reservoir Computing Networks”. In: *Cognit. Comput.* 9.3 (2017), pp. 337–350.

[44] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. “Deep reservoir computing: A critical experimental analysis”. In: *Neurocomputing* 268 (2017), pp. 87–99.

[45] Tian Gan. “Breaking Implicit Assumptions of Physical Delay-Feedback Reservoir Computing”. PhD thesis. Physics, Engineering and Technology, 2023.

[46] Tian Gan et al. “Multi-input Multi-output (MIMO) Delay-Feedback Reservoir Computing: Model-free Control of Van Der Pol Oscillator (submitted)”. In: 2024.

[47] Nikhil Garg et al. “Signals to Spikes for Neuromorphic Regulated Reservoir Computing and EMG Hand Gesture Recognition”. In: *ICONS 2021*. Article 29. ACM, 2021, pp. 1–8.

[48] Timnit Gebru et al. “Datasheets for Datasets”. In: *arXiv:1803.09010 [cs.DB]* (2021). arXiv: 1803.09010 [cs.DB].

[49] George R. Doddington and Thomas B. Schalk. “Speech Recognition: Turning Theory into Practice”. In: *IEEE Spectrum* 18.9 (1981), pp. 26–32.

[50] William Gilpin. “Chaos as an interpretable benchmark for forecasting and data-driven modelling”. In: *CoRR* abs/2110.05266 (2021). arXiv: 2110.05266. URL: https://arxiv.org/abs/2110.05266.

[51] Leon Glass and Michael C. Mackey. “Mackey-Glass equation”. In: *Scholarpedia* 5.3 (2010), p. 6908. DOI: 10.4249/scholarpedia.6908.

[52] Ary L. Goldberger et al. “PhysioBank, PhysioToolkit, and PhysioNet [dataset]”. In: *Circulation* 101.23 (2000), e215–e220. DOI: 10.1161/01.CIR.101.23.e215. eprint: https://www.ahajournals.org/doi/pdf/10.1161/01.CIR.101.23.e215. URL: https://www.ahajournals.org/doi/abs/10.1161/01.CIR.101.23.e215.

[53] Alireza Goudarzi, Alireza Shabani, and Darko Stefanovic. “Exploring transfer function nonlinearity in echo state networks”. In: *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*. May 2015, pp. 1–8.

[54] Xing Xing Guo et al. “Four-channels reservoir computing based on polarization dynamics in mutually coupled VCSELs system”. en. In: *Opt. Express* 27.16 (Aug. 2019), pp. 23293–23306.

[55] S. Harding and J. Miller. “Evolution in materio: initial experiments with liquid crystal”. In: *Proceedings. 2004 NASA/DoD Conference on Evolvable Hardware, 2004.* 2004, pp. 298–305. DOI: 10.1109/EH.2004.1310844.

[56] Simon Haykin et al. “What Makes a Dynamical System Computationally Powerful?” In: *New Directions in Statistical Signal Processing: From Systems to Brains*. MIT Press, 2007, pp. 127–154.

[57] Georg Holzmann and Helmut Hauser. “Echo state networks with filter neurons and a delay&sum readout”. en. In: *Neural Netw.* 23.2 (Mar. 2010), pp. 244–256.

[58] Udo Hübner et al. “Lorenz-like chaos in NH_3-FIR lasers (Data Set A)”. In: *Time Series Prediction: Forecasting The Future And Understanding The Past*. Ed. by Andreas S Weigend and Neal A Gershenfeld. Westview Press, 1994, pp. 73–104.

[59] Y Ichimaru and G B Moody. “Development of the polysomnographic database on CD-ROM”. en. In: *Psychiatry Clin. Neurosci.* 53.2 (Apr. 1999), pp. 175–177.

[60] Masanobu Inubushi and Kazuyuki Yoshimura. “Reservoir Computing Beyond Memory-Nonlinearity Trade-off”. en. In: *Sci. Rep.* 7.1 (Aug. 2017), p. 10199.

[61] H Jaeger. “Short term memory in echo state networks”. In: *GMD Report* (2002).

[62] H Jaeger. *Supporting Online Material to Harnessing nonlinearity: predicting chaotic sys- tems and saving energy in wireless communication*. Feb. 2004.

[63] Herbert Jaeger. “The “echo state” approach to analysing and training recurrent neural networks – with an erratum note”. In: *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148.34 (2001), p. 13.

[64] Herbert Jaeger. *The echo state approach to recurrent neural networks (presentation)*. accessed 29 November 2023. 2004. URL: https://www.ai.rug.nl/minds/uploads/ESNStandardSlides.pdf.

[65] Herbert Jaeger. *Discovering multiscale dynamical features with hierarchical Echo State Networks*. Tech. rep. TR-10. Jacobs University Bremen, 2007.

[66] Herbert Jaeger. “Long short-term memory in echo State Networks: Details of a simulation study”. In: (Feb. 2012).

[67] Herbert Jaeger and Harald Haas. “Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication”. en. In: *Science* 304.5667 (Apr. 2004), pp. 78–80.

[68] Herbert Jaeger, Wolfgang Maass, and Jose Principe. “Special issue on echo state networks and liquid state machines”. In: *Neural Netw.* 20.3 (2007), pp. 287–289.

[69] Herbert Jaeger et al. “Optimization and applications of echo state networks with leaky-integrator neurons”. en. In: *Neural Netw.* 20.3 (Apr. 2007), pp. 335–352.

[70] Sarah Jarvis, Stefan Rotter, and Ulrich Egert. “Extending stability through hierarchical clusters in echo state networks”. en. In: *Front. Neuroinform.* 4 (2010).

[71] Fei Jiang, Hugues Berry, and Marc Schoenauer. “Supervised and Evolutionary Learning of Echo State Networks”. In: *Parallel Problem Solving from Nature – PPSN X*. Springer Berlin Heidelberg, 2008, pp. 215–224.

[72] Fei Jiang, Hugues Berry, and Marc Schoenauer. “Unsupervised Learning of Echo State Networks: Balancing the Double Pole”. In: *Genetic and Evolutionary Computation Conference, GECCO 2008, Proceedings, Atlanta, GA, USA, July 12-16, 2008*. unknown, June 2008, pp. 869–870.

[73] Prashant Joshi and Wolfgang Maass. “Movement Generation and Control with Generic Neural Microcircuits”. In: *Biologically Inspired Approaches to Advanced Information Technology, First International Workshop, BioADIT 2004, Lausanne, Switzerland, January 29-30, 2004. Revised Selected Papers*. Vol. 3141. Springer Verlag, Jan. 2004, pp. 258–273.

[74] Prashant Joshi and Wolfgang Maass. “Movement generation with circuits of spiking neurons”. en. In: *Neural Comput.* 17.8 (Aug. 2005), pp. 1715–1738.

[75] Denis Kleyko et al. “Modality classification of medical images with distributed representations based on cellular automata reservoir computing”. In: *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*. Apr. 2017, pp. 1053–1056.

[76] Danil Koryakin, Johannes Lohmann, and Martin V Butz. “Balanced echo state networks”. en. In: *Neural Netw.* 36 (Dec. 2012), pp. 35–45.

[77] Tomoyuki Kubota, Hirokazu Takahashi, and Kohei Nakajima. “Unifying framework for information processing in stochastically driven dynamical systems”. In: *Physical Review Research* 3.4 (2021), p. 043135. DOI: 10.1103/PhysRevResearch.3.043135.

[78] Dhireesha Kudithipudi et al. “Design and Analysis of a Neuromemristive Reservoir Computing Architecture for Biosignal Processing”. en. In: *Front. Neurosci.* 9 (2015), p. 502.

[79] Yoshiki Kuramoto. “Diffusion-Induced Chaos in Reaction Systems”. In: *Progress of Theoretical Physics Supplement* 64 (1978), pp. 346–367.

[80] Floris Laporte, Joni Dambre, and Peter Bienstman. “Simulating self-learning in photorefractive optical reservoir computers”. en. In: *Sci. Rep.* 11.1 (Jan. 2021), p. 2701.

[81] L Larger et al. “Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing”. en. In: *Opt. Express* 20.3 (Jan. 2012), pp. 3241–3249.

[82] Robert Legenstein and Wolfgang Maass. “Edge of chaos and prediction of computational performance for neural circuit models”. In: *Neural Networks* 20.3 (2007), pp. 323–334. DOI: 10.1016/j.neunet.2007.04.017.

[83] E.N. Lorenz. “Predictability: a problem partly solved”. In: *Seminar on Predictability, 4-8 September 1995*. Vol. 1. ECMWF. Shinfield Park, Reading: ECMWF, 1995, pp. 1–18. URL: https://www.ecmwf.int/node/10829.

[84] Edward N. Lorenz. “Deterministic Nonperiodic Flow”. In: *Journal of Atmospheric Sciences* 20.2 (1963), pp. 130–141. DOI: 10.1175/1520-0469(1963)020⟨0130:DNF⟩2.0.CO;2.

[85] Jake Love et al. “Task Agnostic Metrics for Reservoir Computing”. In: (Aug. 2021). arXiv: 2108.01512v1 [cs.LG].

[86] Zhixin Lu et al. “Reservoir observers: Model-free inference of unmeasured variables in chaotic systems”. en. In: *Chaos* 27.4 (Apr. 2017), p. 041102.

[87] Mantas Lukoševičius. “A practical guide to applying echo state networks”. In: *LNCS*. LNCS. Springer, 2012, pp. 659–686.

[88] Mantas Lukoševičius and Herbert Jaeger. “Reservoir computing approaches to recurrent neural network training”. In: *Computer science review* 3.3 (2009), pp. 127–149.

[89] Thomas Lymburn et al. “Reservoir computing with swarms”. en. In: *Chaos* 31.3 (Mar. 2021), p. 033121.

[90] R Lyon. “A computational model of filtering, detection, and compression in the cochlea”. In: *ICASSP ’82. IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 7. May 1982, pp. 1282–1285.

[91] Qianli Ma, Lifeng Shen, and Garrison W Cottrell. “Deep-ESN: A Multiple Projection-encoding Hierarchical Reservoir Computing Framework”. In: *arXiv:1711.05255 [cs.LG]* (2017). arXiv: 1711.05255 [cs.LG].

[92] Qianli Ma et al. “Decouple Adversarial Capacities with Dual-Reservoir Network”. In: *ICONIP 2017*. Springer, 2017, pp. 475–483.

[93] Qianli Ma et al. “Convolutional Multitimescale Echo State Network”. en. In: *IEEE Trans Cybern* 51.3 (2021), pp. 1613–1625.

[94] Wolfgang Maass, Thomas Natschläger, and Henry Markram. “Real-time computing without stable states”. en. In: *Neural Comput.* 14.11 (2002), pp. 2531–2560.

[95] Zeeshan Khawar Malik, Amir Hussain, and Qingming Jonathan Wu. “Multilayered Echo State Machine: A Novel Architecture and Algorithm”. en. In: *IEEE Trans Cybern* 47.4 (2017), pp. 946–959.

[96] Joshua Mallinson et al. “Experimental Demonstration of Reservoir Computing with Self-Assembled Percolating Networks of Nanoparticles”. In: *Advanced Materials* 36 (May 2024), e2402319. DOI: 10.1002/adma.202402319.

[97] Luca Manneschi, Andrew C Lin, and Eleni Vasilaki. “SpaRCe: Improved Learning of Reservoir Computing Systems Through Sparse Representations”. en. In: *IEEE Trans Neural Netw Learn Syst* PP (Aug. 2021).

[98] Luca Manneschi et al. “Exploiting Multiple Timescales in Hierarchical Echo State Networks”. In: *Frontiers in Applied Mathematics and Statistics* 6 (2021).

[99] Luca Manneschi et al. “Optimising network interactions through device agnostic models”. In: *arXiv preprint arXiv:2401.07387* (2024).

[100] Franklin T. Luk, ed. *Adaptive algorithms for bilinear filtering*. Vol. 2296. International Society for Optics and Photonics. SPIE, 1994, pp. 317–327. DOI: 10.1117/12.190846. URL: https://doi.org/10.1117/12.190846.

[101] *McMaster IPIX Radar [Dataset]*. http://soma.mcmaster.ca/ipix/index.html. Accessed: 2022-4-7.

[102] Leon Glass Micheal C. Mackey. “Oscillations and Chaos in Physiological Control Systems”. In: *Science* 197.4300 (July 1977), pp. 287–289.

[103] *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges [dataset]*. http://yann.lecun.com/exdb/mnist/. Accessed: 2022-4-5.

[104] Grégoire Montavon, Geneviève Orr, and Klaus-Robert Müller. *Neural Networks: Tricks of the Trade*. en. Springer Berlin Heidelberg, Nov. 2012.

[105] John Moon et al. “Temporal data classification and forecasting using a memristor-based reservoir computing system”. en. In: *Nature Electronics* 2.10 (Oct. 2019), pp. 480–487.

[106] Mushegh Rafayelyan, Jonathan Dong, Yongqi Tan, Florent Krzakala, Sylvain Gigan. “Large-Scale Optical Reservoir Computing for Spatiotemporal Chaotic Systems Prediction”. In: *Physical Review* 10.041037 (Nov. 2020).

[107] Kohei Nakajima et al. “A soft body as a reservoir: case studies in a dynamic model of octopus-inspired soft robotic arm”. en. In: *Front. Comput. Neurosci.* 7 (July 2013), p. 91.

[108] *NASA Greenwich Sunspot Numbers [dataset]*. https://solarscience.msfc.nasa.gov/greenwch/SN_m_tot_V2.0.txt. Accessed: 2022-6-4.

[109] NGDC. *NGDC sunspot numbers [dataset]*. https://www.ngdc.noaa.gov/stp/space-weather/solar-data/solar-indices/sunspot-numbers/group/daily-values-and-means/group-sunspot-numbers_monthly-means(monthrg).txt. Accessed: 2022-6-4.

[110] Wilten Nicola and Claudia Clopath. “Supervised learning in spiking neural networks with FORCE training”. In: *Nature communications* 8.1 (2017), p. 2208.

[111] A. Otto, W. Just, and G. Radons. “Nonlinear dynamics of delay systems: an overview”. In: *Philosophical Transactions A* 377.2153 (2019), p. 20180389. DOI: 10.1098/rsta.2018.0389.

[112] Y Paquot et al. “Optoelectronic reservoir computing”. en. In: *Sci. Rep.* 2 (Feb. 2012), p. 287.

[113] Jaideep Pathak et al. “Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data”. en. In: *Chaos* 27.12 (Dec. 2017), p. 121102.

[114] Jaideep Pathak et al. “Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach”. en. In: *Phys. Rev. Lett.* 120.2 (Jan. 2018), p. 024102.

[115] Paul Gerhard Plöger et al. “Echo State Networks for Mobile Robot Modeling and Control”. In: *RoboCup 2003: Robot Soccer World Cup VII*. unknown, July 2003, pp. 157–168.

[116] Balth van der Pol. “LXXXV. On oscillation hysteresis in a triode generator with two degrees of freedom”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 43.256 (1922), pp. 700–719.

[117] *Raw Sunspot data - Carrington [dataset]*. en. https://wwwbis.sidc.be/silso/carrington. Accessed: 2022-4-6.

[118] Ali Rodan and Peter Tino. “Minimum complexity echo state network”. en. In: *IEEE TNN* 22.1 (2011), pp. 131–144.

[119] Ali Rodan and Peter Tino. “Minimum complexity echo state network”. en. In: *IEEE Trans. Neural Netw.* 22.1 (Jan. 2011), pp. 131–144.

[120] Ali Rodan and Peter Tiňo. “Simple Deterministically Constructed Recurrent Neural Networks”. In: *Intelligent Data Engineering and Automated Learning – IDEAL 2010*. Springer Berlin Heidelberg, 2010, pp. 267–274.

[121] Benjamin Roeschies and Christian Igel. “Structure optimization of reservoir networks”. In: *Log. J. IGPL* 18.5 (Oct. 2010), pp. 635–669.

[122] Benjamin Roeschies and Christian Igel. “Structure optimization of reservoir networks”. In: *Logic journal of the IGPL* 18.5 (2010), pp. 635–669. ISSN: 1367-0751. DOI: 10.1093/jigpal/jzp043.

[123] Olivier Roy and Martin Vetterli. *The effective rank: A measure of effective dimensionality*. https://core.ac.uk/download/pdf/147929764.pdf. Accessed: 2022-3-28.

[124] Sadman Sakib et al. “Noise-Removal from Spectrally-Similar Signals Using Reservoir Computing for MCG Monitoring”. In: *ICC 2021 - IEEE International Conference on Communications*. June 2021, pp. 1–6.

[125] Rebecca Pontes Salles. *SantaFe.A: Time series A of the Santa Fe Time Series Competition in TSPred: Functions for Benchmarking Time Series Prediction [dataset]*. en. https://rdrr.io/cran/TSPred/man/SantaFe.A.html. Accessed: 2022-4-7. Jan. 2021.

[126] *Santa Fe Time Series Competition Data Set B [dataset]*. en. https://archive.physionet.org/physiobank/database/santa-fe/. Accessed: 2022-6-1.

[127] Nils Schaetti, Michel Salomon, and Raphael Couturier. “Echo state networks-based reservoir computing for MNIST handwritten digits recognition”. In: *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*. Paris: IEEE, Aug. 2016.

[128] Jürgen Schmidhuber et al. “Training recurrent networks by Evolino”. en. In: *Neural Comput.* 19.3 (Mar. 2007), pp. 757–779.

[129] B Schrauwen and J Van Campenhout. “BSA, a fast and accurate spike train encoding scheme”. In: *Proceedings of the International Joint Conference on Neural Networks, 2003.* Vol. 4. July 2003, 2825–2830 vol.4.

[130] Benjamin Schrauwen, Lars Buesing, and Robert Legenstein. “On Computational Power and the Order-Chaos Phase Transition in Reservoir Computing”. In: *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*. unknown, Jan. 2008, pp. 1425–1432.

[131] Benjamin Schrauwen et al. “Improving reservoirs using intrinsic plasticity”. In: *Neurocomputing* 71.7-9 (2008), pp. 1159–1171.

[132] Friedhelm Schwenker and Amr Labib. “Echo State networks and Neural network Ensembles to predict Sunspots activity”. In: *ESANN 2009*. 2009.

[133] Biraj Shakya et al. “A Circuit-embedded Reservoir Computer for Smart Noise Reduction of MCG Signals”. In: *2021 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)*. Nov. 2021, pp. 56–61.

[134] Yi-Wei Shen et al. “Deep photonic reservoir computing recurrent network”. In: *Optica* 10 (Dec. 2023). DOI: 10.1364/OPTICA.506635.

[135] Md Raf E Ul Shougat and Edmon Perkins. “The van der Pol physical reservoir computer”. In: *Neuromorphic Computing and Engineering* 3.2 (2023), p. 024004.

[136] Md Raf E Ul Shougat et al. “A Hopf physical reservoir computer”. en. In: *Sci. Rep.* 11.1 (Sept. 2021), p. 19465.

[137] G I Sivashinsky. “Nonlinear analysis of hydrodynamic instability in laminar flames—I. Derivation of basic equations”. In: *Acta Astronaut.* 4.11 (Nov. 1977), pp. 1177–1206.

[138] G I Sivashinsky. “On Flame Propagation Under Conditions of Stoichiometry”. In: *SIAM J. Appl. Math.* 39.1 (1980), pp. 67–82.

[139] J.J.E. Slotine and W. Li. “Applied Nonlinear Control”. In: Prentice-Hall International Editions (1991). URL: https://books.google.co.uk/books?id=HddxQgAACAAJ.

[140] Miguel C Soriano et al. “Delay-based reservoir computing: noise effects in a combined analog and digital implementation”. en. In: *IEEE Trans Neural Netw Learn Syst* 26.2 (Feb. 2015), pp. 388–393.

[141] Susan Stepney. “Non-instantaneous Information Transfer in Physical Reservoir Computing”. In: *UCNC 2021 Espoo, Finland, October 2021*. Vol. 12984. LNCS. Springer, 2021, pp. 164–176. DOI: 10.1007/978-3-030-87993-8_11.

[142] Susan Stepney. “Non-instantaneous Information Transfer in Physical Reservoir Computing”. In: *UCNC 2021*. Vol. 12984. LNCS. Springer, 2021, pp. 164–176.

[143] Susan Stepney. “Physical Reservoir Computing: a tutorial”. In: *Natural Computing (in press)* (2024).

[144] Charles Swindells. “Nailed it: Reservoir computing with a rusty iron nail”. Submitted 2024, presented at Magnetism, Institute of Physics, University of Manchester, 2023.

[145] Gouhei Tanaka et al. “Recent advances in physical reservoir computing: A review”. en. In: *Neural Netw.* 115 (July 2019), pp. 100–123.

[146] Adrian Thompson and Paul Layzell. “Analysis of Unconventional Evolved Electronics”. In: *Commun. ACM* 42 (Apr. 1999), pp. 71–79. DOI: 10.1145/299157.299174.

[147] *TI 46-Word [dataset]*. en. https://catalog.ldc.upenn.edu/LDC93S9. Accessed: 2022-4-7.

[148] Jacob Torrejon et al. “Neuromorphic computing with nanoscale spintronic oscillators”. In: *Nature* 547.7664 (2017), pp. 428–431.

[149] F. Triefenbach et al. “Phoneme recognition with large hierarchical reservoirs”. In: *Adv. Neural Inf. Process. Syst.* 23 (2010), pp. 2307–2315.

[150] Fabian Triefenbach et al. “Acoustic Modeling With Hierarchical Reservoirs”. In: *IEEE TASLP* 21.11 (2013), pp. 2439–2450.

[151] Balth Van Der Pol and Jan Van Der Mark. “LXXII. The heartbeat considered as a relaxation oscillation, and an electrical model of the heart”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 6.38 (1928), pp. 763–775.

[152] Kristof Vandoorne et al. “Parallel reservoir computing using optical amplifiers”. In: *IEEE transactions on neural networks* 22.9 (2011), pp. 1469–1481.

[153] Kristof Vandoorne et al. “Experimental demonstration of reservoir computing on a silicon photonics chip”. In: *Nature communications* 5.1 (2014), p. 3541.

[154] Andrew Varga and Herman J M Steeneken. “Assessment for automatic speech recognition: II. NOISEX-92: A database and an experiment to study the effect of additive noise on speech recognition systems”. In: *Speech Commun.* 12.3 (July 1993), pp. 247–251.

[155] D Verstraeten, B Schrauwen, and D Stroobandt. “Reservoir-based techniques for speech recognition”. In: *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. July 2006, pp. 1050–1053.

[156] D Verstraeten et al. “Isolated word recognition with the Liquid State Machine: a case study”. In: *Inf. Process. Lett.* 95.6 (Sept. 2005), pp. 521–528.

[157] D Verstraeten et al. “An experimental unification of reservoir computing methods”. en. In: *Neural Netw.* 20.3 (Apr. 2007), pp. 391–403.

[158] David Verstraeten et al. “Memory versus non-linearity in reservoirs”. In: *The 2010 International Joint Conference on Neural Networks (IJCNN)*. July 2010, pp. 1–8.

[159] I T Vidamour et al. “Quantifying the computational capability of a nanomagnetic reservoir computing platform with emergent magnetisation dynamics”. In: *Nanotechnology* 33.48 (2022). DOI: 10.1088/1361-6528/ac87b5.

[160] Quentin Vinckier et al. “High-performance photonic reservoir computer based on a coherently driven passive cavity”. en. In: *Optica, OPTICA* 2.5 (May 2015), pp. 438–446.

[161] Pantelis R Vlachas et al. “Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks”. en. In: *Proc. Math. Phys. Eng. Sci.* 474.2213 (May 2018), p. 20170844.

[162] Pantelis R Vlachas et al. “Forecasting of Spatio-temporal Chaotic Dynamics with Recurrent Neural Networks: a comparative study of Reservoir Computing and Backpropagation Algorithms”. In: *ArXiv* (2019).

[163] Andreas S Weigend. *Time Series Prediction: Forecasting The Future And Understanding The Past*. en. Routledge, May 2018.

[164] Andreas S Weigend and Neal A Gershenfeld, eds. *Time Series Prediction: Forecasting The Future And Understanding The Past*. Westview Press, 1994.

[165] Daan Wierstra, Faustino J. Gomez, and Jürgen Schmidhuber. “Modeling systems with internal state using evolino”. In: *GECCO 2025*. ACM, 2005, pp. 1795–1802. DOI: 10.1145/1068009.1068315.

[166] Ronald J Williams and David Zipser. “Gradient-based learning algorithms for recurrent networks and their computational complexity”. In: *Backpropagation*. Psychology Press, 2013, pp. 433–486.

[167] Chester Wringe, Susan Stepney, and Martin Trefzer. “Modelling and Evaluating Restricted ESNs on Single-and Multi-Timescale Problems”. In: *Natural Computing* (2023). DOI: https://doi.org/10.21203/rs.3.rs-3758288/v1.

[168] Chester Wringe, Susan Stepney, and Martin A Trefzer. “Restricted reservoirs on heterogeneous timescales”. en. In: *Artificial Neural Networks and Machine Learning*. Vol. 15025. LNCS. ICANN 2024, Lugano, Switzerland: Springer 2024, Sept. 2024, pp. 168–183.

[169] Chester Wringe, Susan Stepney, and Martin A. Trefzer. “Heterogeneous Reservoirs for Predicting Multivariate Physiological Data”. In: 2024.

[170] Chester Wringe, Susan Stepney, and Martin Albrecht Trefzer. “Modelling and Evaluating Restricted ESNs”. In: *Lecture Notes in Computer Science*. Vol. 14003. UCNC 2023, Jacksonville FL, USA, 2023.

[171] Yanbo Xue, Le Yang, and Simon Haykin. “Decoupled echo state networks with lateral inhibition”. en. In: *Neural Netw.* 20.3 (2007), pp. 365–376.

[172] Kyongmin Yeo. “Data-driven reconstruction of nonlinear dynamics from sparse observation”. In: *Journal of Computational Physics* 395 (2019), pp. 671–689.

[173] Izzet B. Yildiz, Herbert Jaeger, and Stefan J. Kiebel. “Re-visiting the echo state property”. In: *Neural Networks* 35 (2012), pp. 1–9. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2012.07.005. URL: https://www.sciencedirect.com/science/article/pii/S0893608012001852.

[174] George Undy Yule. “On a method of investigating periodicities in disturbed series, with special reference to Wolfer’s sunspot numbers”. In: *Phil. Trans. Roy. Soc. A* 226.636-646 (1927), pp. 267–298.

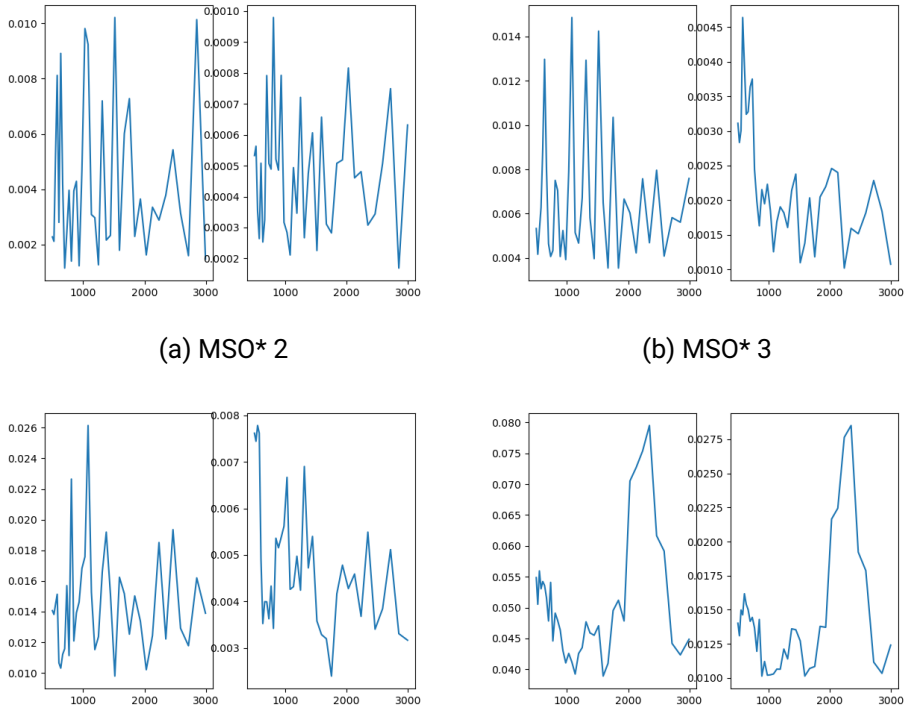
[175] Heng Zhang and Danilo Vasconcellos Vargas. “A Survey on Reservoir Computing and its Interdisciplinary Applications Beyond Traditional Machine Learning”. In: *IEEE Access* 11 (2023), pp. 81033–81070. DOI: 10.1109/access.2023.3299296. URL: https://doi.org/10.1109/access.2023.3299296.

[176] Shaohui Zhang et al. “Deep Fuzzy Echo State Networks for Machinery Fault Diagnosis”. In: *IEEE Transactions on Fuzzy Systems* 28.7 (2020), pp. 1205–1218. DOI: 10.1109/TFUZZ.2019.2914617.

Appendix for Ch. 5: Modelling and Evaluating Restricted ESNs on Single- and Multi-Timescale Problems

A.1 MSO Training Lengths

See fig. A.1.



(a) MSO* 2

(b) MSO* 3

(c) MSO* 4

(d) MSO* 8

Figure A.1: NRMSE mean and standard deviation over 50 runs of the MSO* task, for training lengths between 200 and 3000 datapoints. The washout length used is 100 datapoints, and the testing length is 200 datapoints.

A.2 MSO Densities

See fig. A.2.

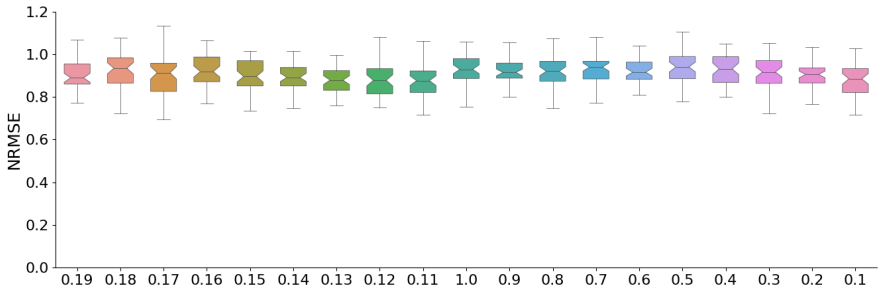


Figure A.2: NRMSEs over 50 runs of different densities for the MSO*8 task

Appendix for Ch. 6: MSO* on Multi-Timescale reservoirs

B.1 Scaling the Weight Matrix

B.1.1 Scaling Off-Diagonals

See fig. B.1

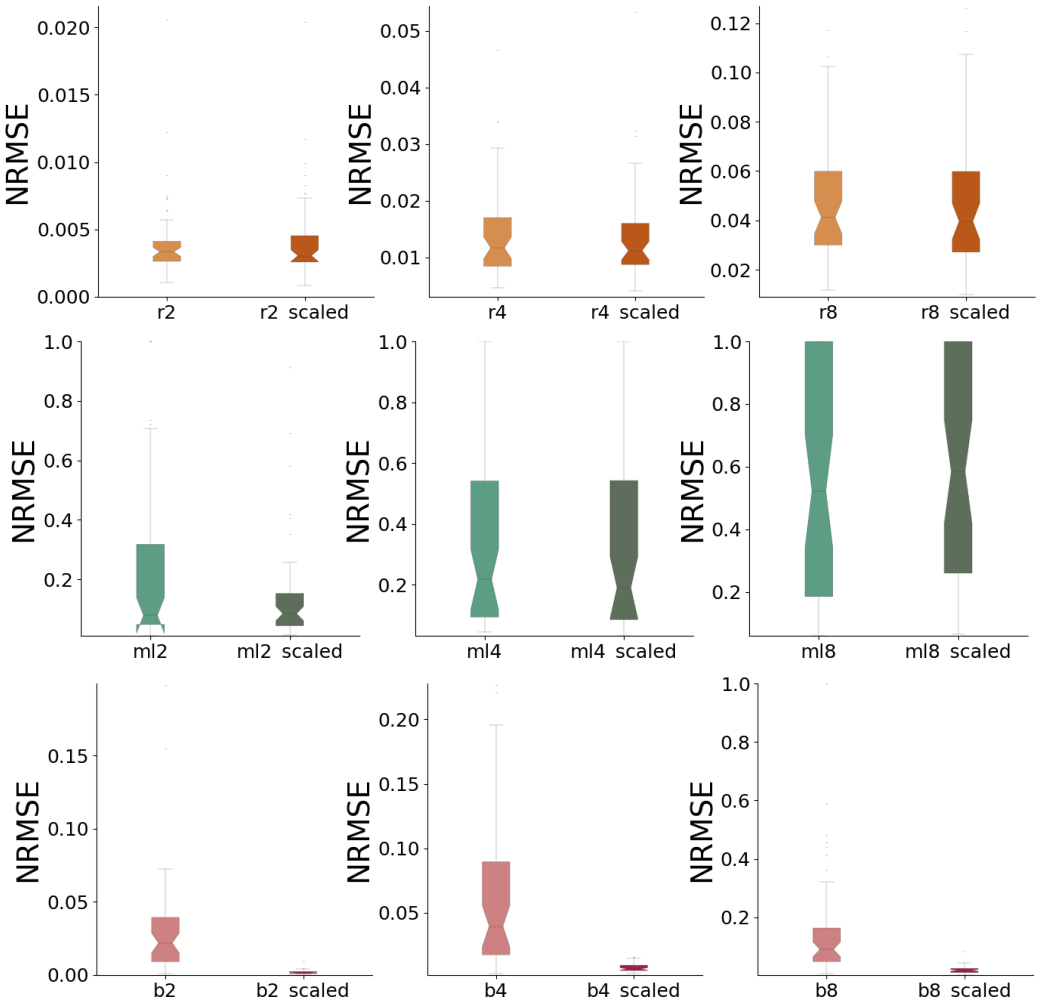


Figure B.1: Results for MSO*2 (left), 4 (center), and 8 (right) with and without scaling of the off-diagonal edges for the three mock materials.

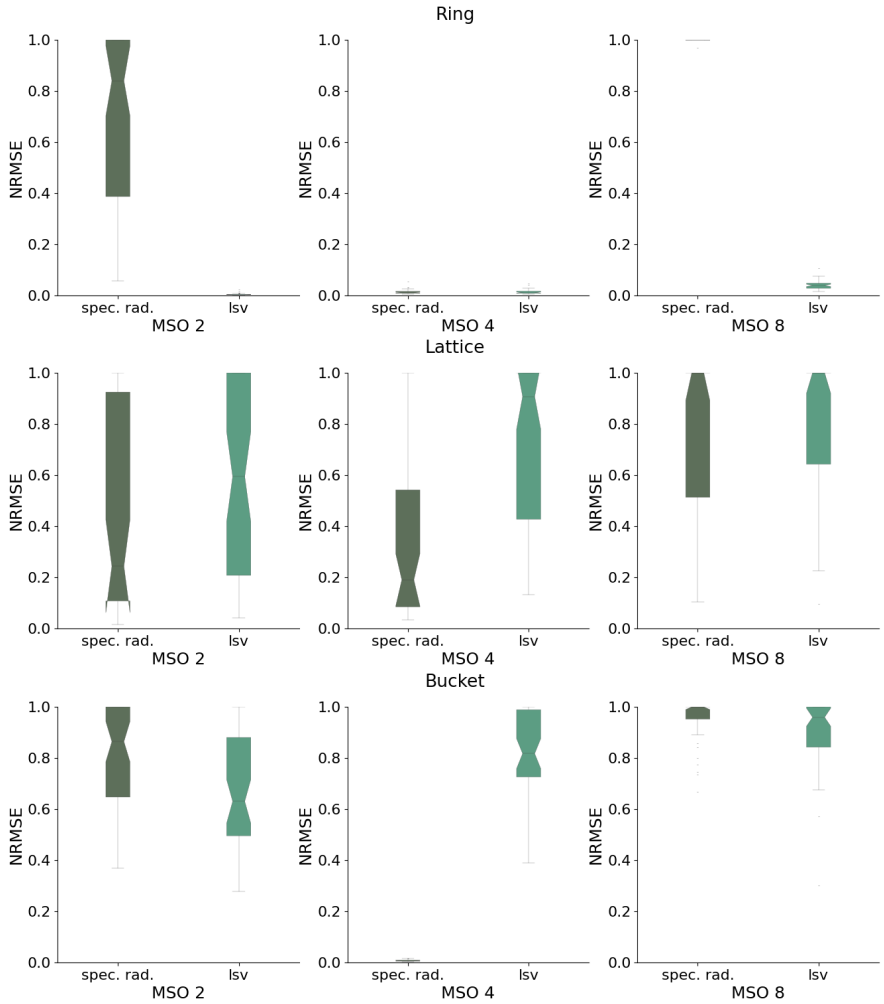


Figure B.2: Results for MSO*2 (left), 4 (center), and 8 (right) scaling the weight matrices by the largest singular value and by the spectral radius.

B.1.2 Spectral Radius v. Largest Singular value

See figure B.2.

C Appendix for Ch. 8: Heterogeneous Timescales and Materials

C.1 MSO Experiments

C.1.1 Material Selection

See figs. C.1 – C.6.

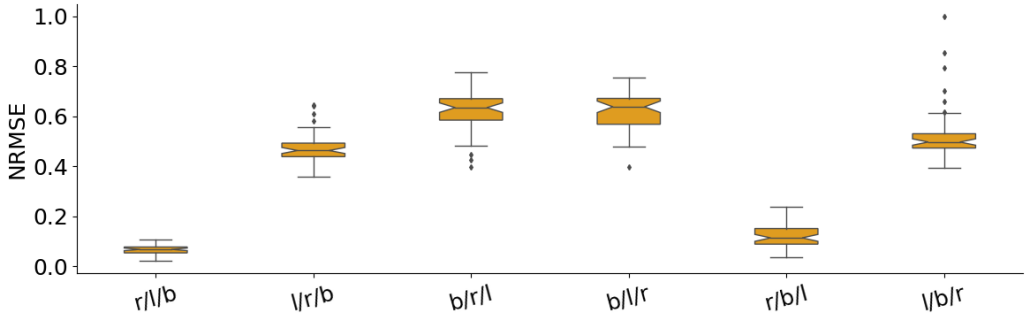


Figure C.1: Selection of materials for multi-material MSO experiments

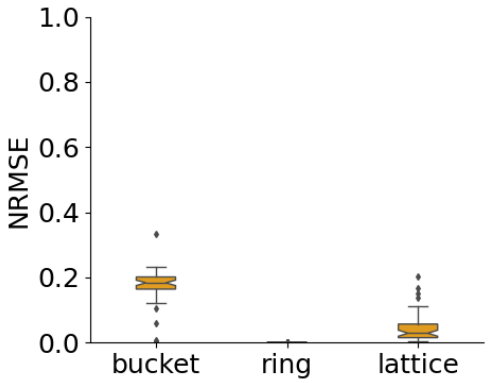
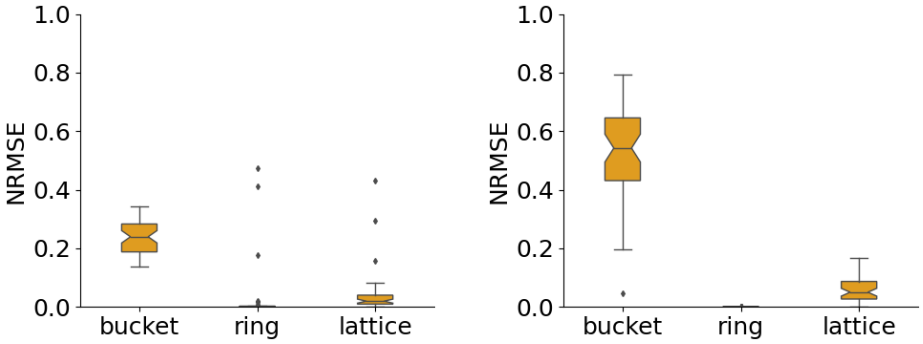


Figure C.2: MSO single reservoir

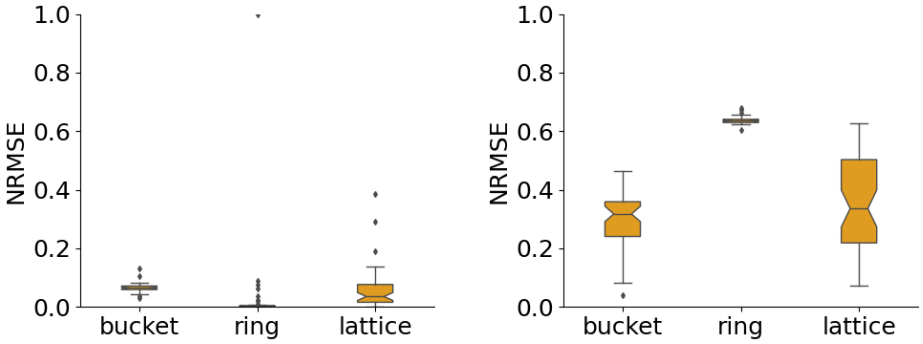
Materials chosen for full comparison:

- SM large reservoir: Ring
- SM & ST (A): Ring



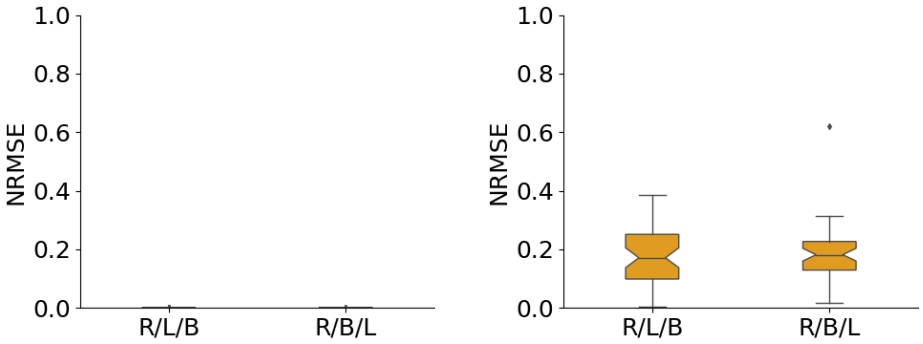
(a) all-to-all (b) one-to-one

Figure C.3: MSO: single material, single timescale



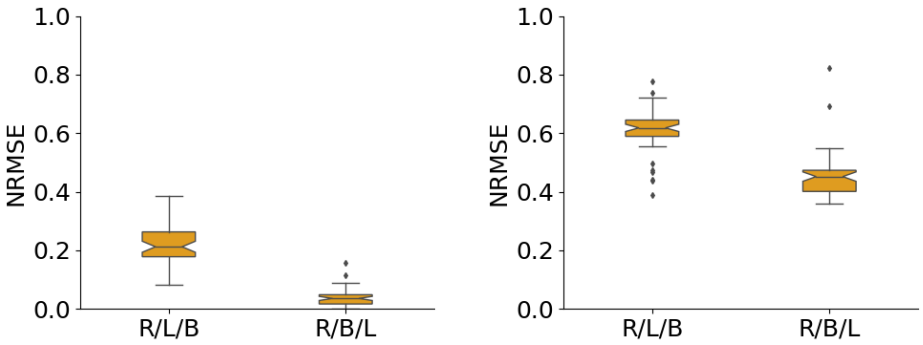
(a) all-to-all (b) one-to-one

Figure C.4: MSO: single material, multi timescale



(a) all-to-all (b) one-to-one

Figure C.5: MSO: multi material, single timescale



(a) all-to-all (b) one-to-one

Figure C.6: MSO: multi material, multi timescale

- SM & ST (1): Ring
- SM & MT (A): Ring
- SM & MT (1): Bucket
- MM & ST (A): R/B/L
- MM & ST (1): R/B/L
- MM & MT (A): R/B/L
- MM & MT (1): R/B/L

C.2 Sleep Apnea Experiments

C.2.1 Material Selection

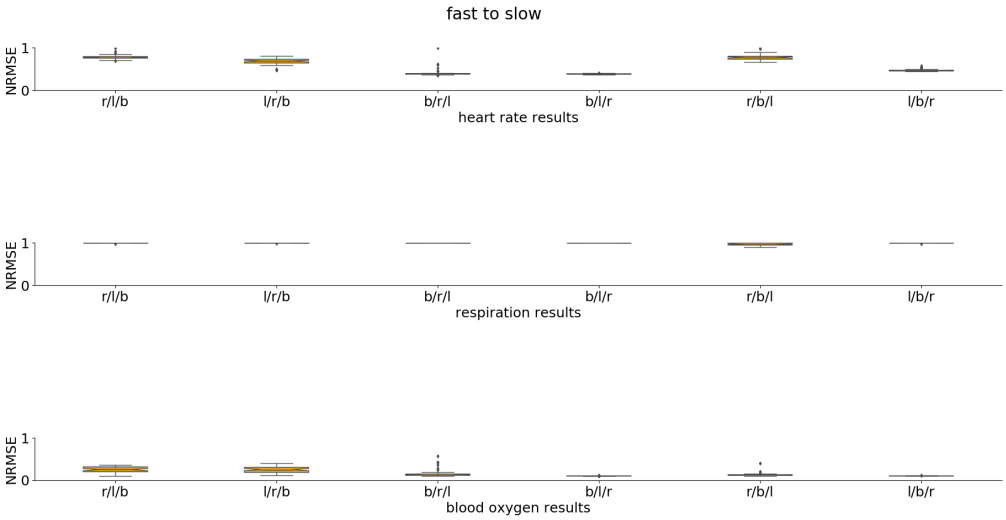
See fig. C.7.

C.2.2 All Divergences

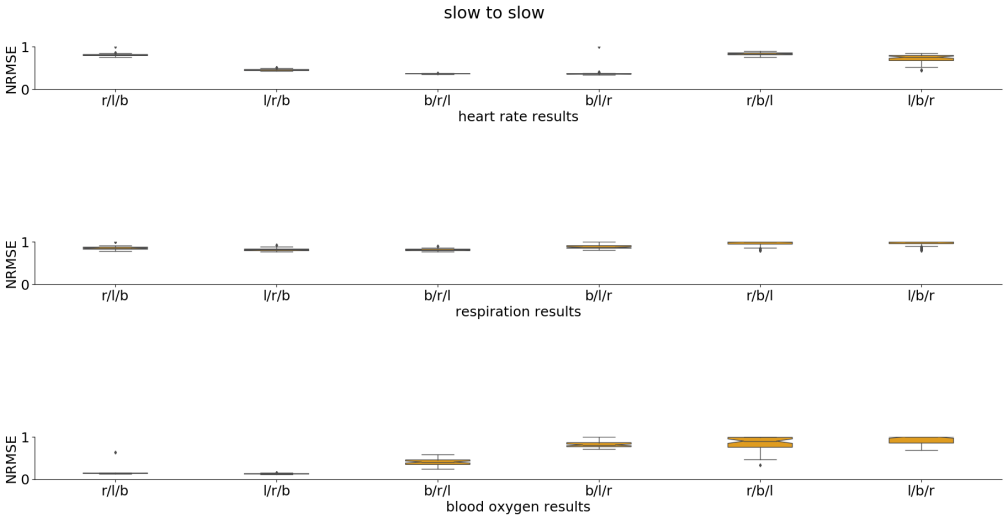
See tab. C.1.

C.2.3 Material Comparison (All Results)

- SM large reservoir: Ring
- SM & ST (A): Lattice
- SM & ST (1): Lattice
- SM, MT (A): Lattice
- SM, MT (1): Lattice
- MM, ST (A): B/R/L



(a) Fast to slow



(b) Fast to fast

Figure C.7: Sleep apnea multi-material material selection

reservoir type	heart	chest	blood
Fast to slow			
Ring/Lattice/Bucket	0	48	0
Lattice/Bucket/Ring	0	48	0
Bucket/Ring/Lattice	2	51	0
Bucket/Lattice/Ring	0	51	0
Ring/Bucket/Lattice	1	13	0
Lattice/Bucket/Ring	0	50	0
Fast to fast			
Ring/Lattice/Bucket	1	3	0
Lattice/Ring/Bucket*	0	0	0
Bucket/Ring/Lattice*	0	0	0
Bucket/Lattice/Ring	1	5	2
Ring/Bucket/Lattice	0	31	12
Lattice/Bucket/Ring	0	32	28

Table C.1: Divergences for all the material combinations for the sleep apnea task.

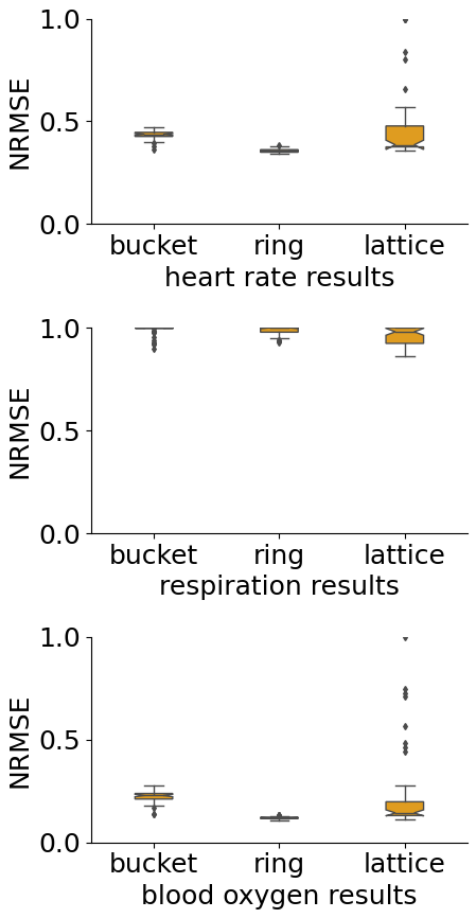
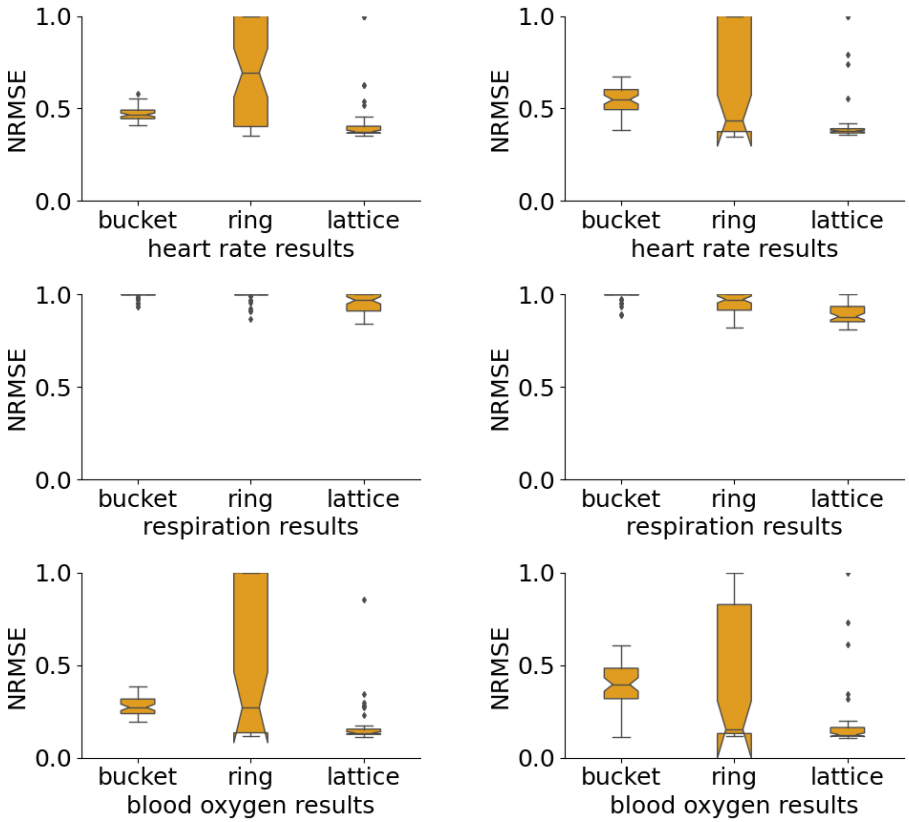


Figure C.8: sleep apnea, single reservoir

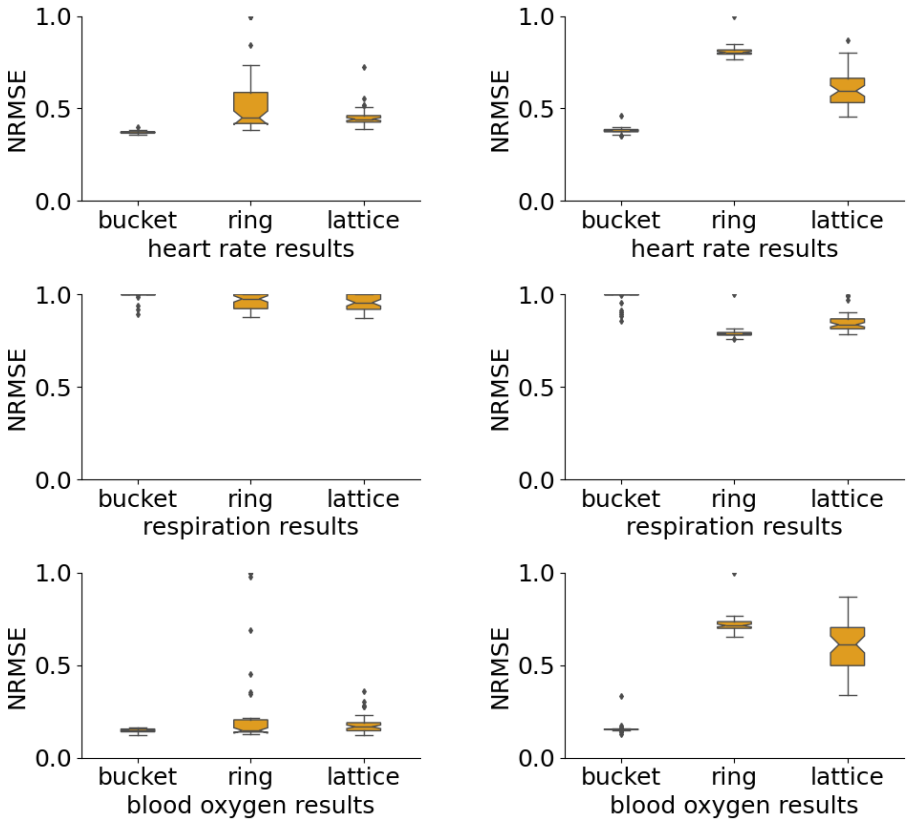
Reservoir type	heart	chest	blood
ESNs			
Standard ESN	0	51	0
Restricted ESN	0	5	0
SM large reservoir			
Bucket	0	38	0
Ring	0	27	0
Lattice*	3	25	1
SM & ST (A)			
Bucket	0	40	0
Ring	21	41	19
Lattice*	5	21	0
SM & ST (1)			
Bucket	0	43	0
Ring	15	25	14
Lattice*	2	8	1
SM & MT (A)			
Bucket	0	45	0
Ring	10	23	17
Lattice*	0	17	0
SM & MT (1)			
Bucket	0	43	0
Ring	1	1	1
Lattice*	0	0	0
MM & ST (A)			
BRL*	0	23	0
LRB	0	25	0
MM & ST (1)			
BRL*	1	1	0
LRB	3	5	2
MM & MT (A)			
BRL*	0	28	0
LRB	0	32	0
MM & MT (1)			
BRL	0	0	0
LRB*	0	0	0

Table C.2: Number of divergent results ($NRMSE > 1$) over 50 runs of the reservoirs in our experiments. The results selected for the full comparison are marked with a *.



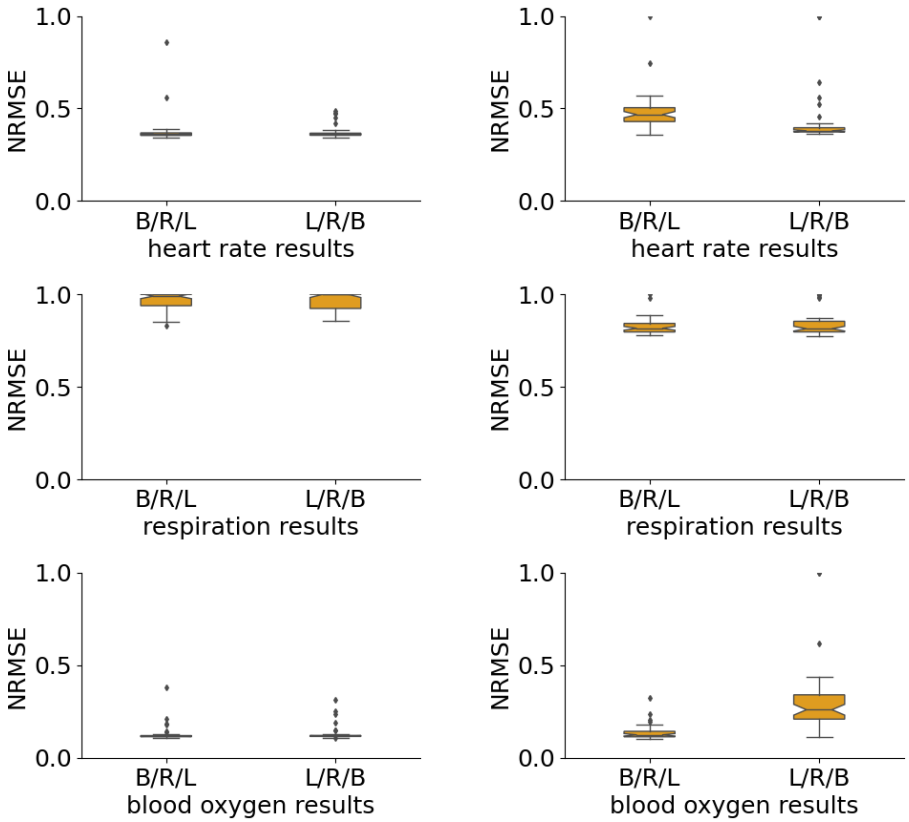
(a) all-to-all (b) one-to-one

Figure C.9: Sleep apnea:Single material, single timescale



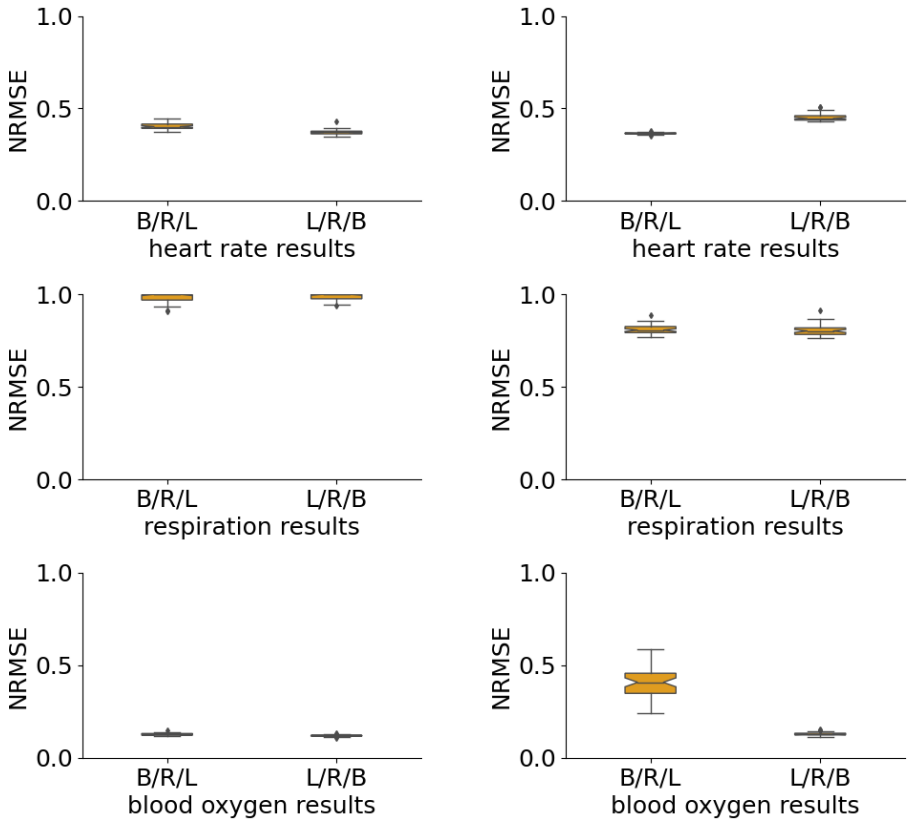
(a) all-to-all (b) one-to-one

Figure C.10: Sleep apnea:Single material, multi timescale



(a) all-to-all (b) one-to-one

Figure C.11: Sleep apnea: Multi material, single timescale



(a) all-to-all (b) one-to-one

Figure C.12: Sleep apnea: Multi material, multi timescale

- MM, ST (1): B/R/L
- MM, MT (A): B/R/L
- MM, MT (1): L/R/B