Bachelor Thesis

Memristor-based Echo State Networks at the Edge of Chaos

Paul Stapel

June 22, 2026

Supervisors

Dr. C. Spitoni
Mathematical Institute - Utrecht University

Prof. Dr. R.H.H.G. van Roij
Institute for Theoretical Physics - Utrecht University

Abstract

In this thesis, we iterate on literature on memristor-based Echo State Networks (ESNs) to find a network that behaves at the Edge of Chaos. Through means of the theoretical model of the Edge of Stability Echo State Network (ES^2N), an improvement is proposed meant to lift the ESN into this chaotic regime, characterized by an annular region of Jacobian eigenvalues all near 1 in magnitude. A physical model of the ES^2N based on conical memristors is proposed, mathematically shown to correspond to the theoretical model of the ES^2N. Through simulation on the chaotic Mackey-Glass timeseries, an increase in Valid Prediction Time (VPT) of 59.0% has been established, with increased efficiency (VPT per Watt) ranging between 35.9% and 59.0%. Moreover, it is shown that the new physical ES^2N performs at least as well as the theoretical model when studying this benchmark. Finally, it is shown that the physical ES^2N is able to perform well at the MSO8 benchmark, an input sequence containing multiple timescales which is impossible for the standard ESN.

Contents

1. Introduction

In recent years, the use of Artificial Intelligence (AI) has seen a worldwide surge as technology advanced to allow Large Language Models to become relevant. Throughout this recent development, increase in global energy consumption has become a concern among researchers. Global datacenter energy consumption is projected to double by 2030, and it will account for 3% of total global energy consumption [1]. With this rising demand in AI, green energy solutions have become more and more sought after.

The human brain offers a compelling source of inspiration when it comes to efficient computation. With just 20 Watts [2], a power consumption comparable to household lightbulbs, it is able to perform learning tasks and reason about the world around us. Because of this efficiency, research has been done into neuromorphic (brain-inspired) devices, such as fluidic iontronic memristors that mimic the brains salt-and-water based approach to computation [3–8]. Benefits to these types of devices include compatibility for brain-computer interfaces, low electromagnetic interference, low energy consumption and tunable ion transport enabled by the variability of ion types and sizes [3].

In this thesis, we will use memristors as neuromorphic building blocks for implementing so-called Echo State Networks for machine learning tasks involving temporal data.

1.1 Neuromorphic components

Traditionally, three main electrical components were known, namely the resistor, inductor and capacitor. Through symmetry arguments on their relations between Current, Potential, Flux and Charge, a 4th element was also hypothesized in 1971: the memristor [9]. In the early 2000's, physical memristors have been realized in many different ways [10–16], though it is argued none of these are an actual memristor as per the original definition [17]. Nonetheless, it has been shown these physical implementations of a memristor component have characteristics useful in the field of neuromorphic computing [18]. We will go into further depth on these components in Chapter 2.

1.2 Memristor based machine learning

Recently, a correspondence between memristor based circuits and Leaky Echo State Networks (Li-ESNs, see Chapter 3) has been formally established [19]. This makes memristor-based ESNs useful for a variety of tasks related to temporal data, such as time-series forecasting (e.g. energy, weather, financial markets), medical signal processing, speech recognition, computer vision, robotics and real-time anomaly detection in industrial systems [20–32].

However, for the memristor-based implementation of such networks to be useful in practice, they must still have sufficient performance compared to the digital counterpart. For recurrent neural networks (RNNs, see chapter 3) like ESNs to be efficient and most theoretically tractable, they must operate at the so-called Edge of Chaos [33–36]. Current implementations of memristor based ESNs have not yet been able to reach this regime [19, 37]. This forms the main motivation of this thesis.

1.3 Thesis outline

Because current memristor-based Echo State Networks do not operate at the Edge of Chaos, this thesis will aim to find an answer to the following research questions:

RQ1. *How can Edge of Chaos dynamics be introduced into a memristor-based Echo State Network?*

RQ2. *To what extent does the introduction of Edge of Chaos dynamics improve the predictive performance of the memristor-based Echo State Network, and how does this affect the energy efficiency of the physical circuit?*

In section 2 and 3 of this thesis, we will review some elementary theory on memristors and Echo State Networks respectively. The aim of these sections is to lay the foundation for the actual results and new material in this thesis. From there, we will go over the definitions of chaos in section 4, some useful theories about chaos and how it relates to machine learning.

Having covered all theory and stated our problem, we turn to solutions in the Edge of Stability Echo State Networks and the mathematical theory around them in section 5. In doing so, we look into potential benefits such a network has over the standard Echo State Network. Next, we explore the physical aspects of such a network in section 6. Specifically, we wish explore what additions we would need to make to our circuit in order to have it be mathematically equivalent to an ES^2N. From here, we dedicate section 7 to the modelling of this physical implementation, comparing benchmarks with the original ESN to quantify the effect Edge of Chaos computing has using simulations. The results of these simulations will be laid out in section 8 and discussed in section 9. Finally, we reach a conclusion on our research questions in section 10.

2. Memristors

In electrical circuits, we are usually concerned with three main electrical components, namely the resistor, inductor and capacitor. These govern the relations between voltage (v), current (i), flux (Φ) and charge (q). Writing down the constitutive equations these components follow, we can see that:

$$\begin{cases} \text{Resistor:} & dV := R\,dI \\ \text{Capacitor:} & dq := C\,dV \\ \text{Inductor:} & d\Phi := L\,dI \end{cases}$$

and by definition of flux and charge:

$$\begin{cases} \text{Flux:} & d\Phi := V\,dt \\ \text{Charge:} & dq := I\,dt \end{cases}$$

Looking at these equations more carefully, we can see that out of the ways the 4 quantities can relate, a relation between flux and charge is missing. Because of this missing relation, Leon Chua proposed a new basic circuit element in 1971, which he dubbed the memristor [9]. The namings of this element will become clear in the next section.

2.1 Definition of a memristor

The constitutive equation that describes the memristor relates flux to charge using some value M, called the memristance:

$$d\Phi := M\big(q(t)\big)dq \tag{2.1}$$

Using the definition of flux and charge, we can rewrite this:

$$M\big(q(t)\big) = M\Big(\int_{-\infty}^{t} I(t')dt'\Big) = \frac{\frac{d\Phi}{dt}}{\frac{dq}{dt}} = \frac{V(t)}{I(t)} \tag{2.2}$$

Note here that this expression will not hold when $I(t) = 0$. In this case, we must understand the memristance in terms of the definition (2.1). Then, we can see that memristance has a clear resemblance to resistance, both having units Ohm (Ω). The only difference is that the memristance depends on the entire history of the charge, as opposed to just some momentary voltage value. This is the reasoning behind the name memristor; we are dealing with a resistive device that has memory of the charge at previous times.

When the current is set to zero, the charge must be constant, and so $M(q)$ also becomes a constant value. In this case, we recover exactly the constitutive equation for the resistor, further validating the similarity between memristors and resistors. From hereon out, we shall use the reciprocal of the memristance — which we shall call the conductance, $g\big(q(t)\big)$ or simply $g(t)$ — because it is easier to work with in parallel circuits.

2.2 Conical memristors

The specific memristors we will be studying in this thesis are conical fluidic iontronic memristors. Throughout this thesis, we shall refer to these simply as conical memristors. An example of such a conical memristor is displayed in figure 1.

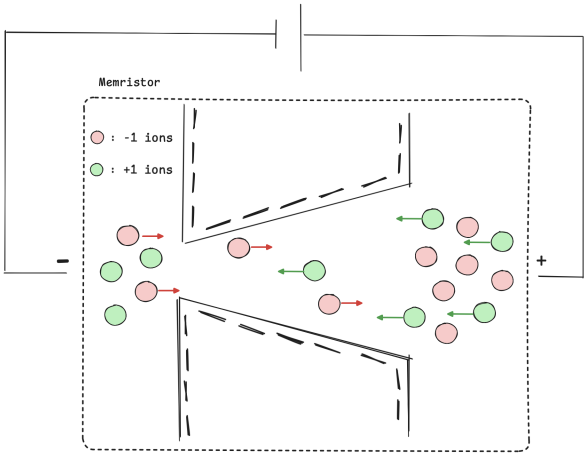

Figure 1: *Internals of a conical fluidic iontronic memristor. When a voltage is applied over the memristor, ions will move through the channel based on the existing concentration gradient of the ions, magnitude of the applied voltage, surface charge of the channel and the dimensions of the channel.*

In this conical channel, the ion concentration gradient and surface charge on the channel create a resistance to ion flow. As the ions move from one side to another, it becomes more and more difficult to move ions into that side, increasing electrical resistance in the channel. Because the ionic conductance of the channel depends on the instantaneous concentration profile, and because that profile reflects the entire history of prior voltage inputs, the channel conductance depends on the cumulative history of the applied voltages. When the voltage is kept at a constant value, the conductance will tend to some equilibrium value. We say that the memristor is volatile; the state will change even when there is no change in voltage.

To state this tendance to equilibrium more generally, we can note that our steady state conductance depends only on the applied voltage V, so that we can name it $g_\infty(V)$. Then, the dynamics of the memristor can be written as

$$\dot{g} = f(g, V). \tag{2.3}$$

Here, we reach equilibrium, where $g \to g_\infty(V)$ and $f(g, V) \to 0$, at some constant relaxation timescale τ intrinsic to the device. With this we can proceed to further refine our mathematical model of this conical memristor.

2.3 Simple Volatile Memristors

Knowing that a conical memristor obeys Eq. (2.3), we can try to narrow down an actual equation using some simplifications, following the 2024 paper by Kamsma et al. [18]. A natural assumption would be that the function $f(g, V)$ depends on the distance of the current conductance to steady state. Then, we could write

$$\dot{g} = f(g_\infty(V) - g(t)).$$

As we would reach a steady state, we would off course have that $f(0) = 0$. The full form of f, however, remains unknown as of now. Let us then expand f to first order to find:

$$\dot{g} = f(0) + \big(g_\infty(V) - g(t)\big) f'(0) + O\Big(\big(g_\infty(V) - g(t)\big)^2\Big).$$

Finally, due to dimensional analysis, we conclude $f'(0)$ has units time^{-1}, so that we can match it with the constant relaxation timescale τ that we previously mentioned. With this, all our simplifications have been done, and we are finally ready to define the class of memristors our conical memristor falls into: the simple volatile memristor (SVM).

Definition 1 (Simple Volatile Memristor)**.** *A Simple Volatile Memristor (SVM) is a circuit component with a dynamic conductance $g(t)$ that tends to some voltage-dependent steady state conductance $g_\infty(V)$ such that this dynamic conductance can be described by*

$$\begin{cases} I(t) = g(t)V(t) & (a), \\ \dot{g}(t) = \dfrac{g_\infty(V) - g(t)}{\tau} & (b) \end{cases} \tag{2.4}$$

For the memristor as in Fig. 1, the $g_\infty(V)$ can be described by the following equation [38]:

$$\frac{g_\infty(V)}{g_0} = 1 + \Delta g \int_0^L \left[\frac{x}{L} \frac{R_t}{R(x)} - \frac{e^{\mathrm{Pe}(V)\,\frac{x}{L}\,\frac{R_t^2}{R_b R(x)}} - 1}{e^{\mathrm{Pe}(V)\,\frac{R_t}{R_b}} - 1} \right] \frac{dx}{L} \tag{2.5}$$

where the quantities used are derived from the Poisson-Nernst-Planck-Stokes equations [39]. The value $g_0 = g_\infty(0)$ denotes the steady-state conductance at $V = 0$, and $\Delta g = 2w\frac{R_b - R_t}{R_b} D_u$, with $D_u \approx -0.25$. The channel length is given by L, and the channel radius is described by $R(x)$, with R_t and R_b denoting the radii at the tip and base respectively. Furthermore, $Pe(V) = \frac{Q(V)L}{D\pi R_t^2}$ is the Péclet number, where $Q(V) = \frac{R_t R_b \epsilon \psi_0}{\eta L} V$ represents the electro-osmotic flow and D is the diffusion coefficient of the ions. Finally, $w = \frac{eD\eta}{k_b T \epsilon \psi_0}$ denotes the ratio of ionic to electro-osmotic mobility, where ϵ is the permittivity, ψ_0 is the surface potential and η the viscocity.

When plotting Eq. (2.5) as a function of V, using the values as proposed in [38] (see Appendix A), we find the graph in figure 2.

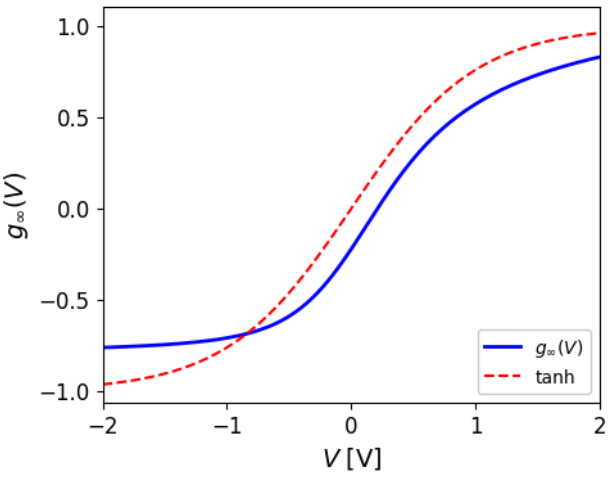

Figure 2: *The steady state conductance as per Eq.* (2.5) *for different voltages. When plotting it together with the* tanh *function, we can see that it matches its nonlinear behavior.*

Because $g_\infty(V)$ roughly approximates a hyperbolic tangent function, and *tanh* is widely used as an activation function in machine learning [40], we conclude that g_∞ can take its place in machine learning tasks. We will be able to exploit these nonlinear properties later when we talk about Echo State Networks in the next chapter.

2.4 Linear SVM

Other SVMs also exist. One specific example of note is the linear SVM. This SVM still obeys the dynamics as described by Eq. (2.4), but unlike the nonlinear steady state conductance as described earlier, $g_\infty(V)$ takes on a linear form [41]. In this case, we may write $g_\infty(V) = g_0^l + bV$, where we let b be some coupling constant with units conductance per volt. The dynamics of this memristor then become:

$$\begin{cases} I(t) = g^l(t)V(t) & \text{(a)}, \\ \dot{g}^l(t) = \dfrac{g_0^l + bV - g^l(t)}{\tau} & \text{(b)} \end{cases} \tag{2.6}$$

This concludes all prerequisite knowledge on memristors. In the next section, we shall start working towards a network using these SVMs as building blocks.

3. Echo State Networks

In this section, we will be working towards a physical memristive device designed to process temporal data. The device we will be turning physical to predict this temporal data is a recurrent neural network called an Echo State Network. For this goal, we will first review some basic machine learning.

3.1 Recurrent Neural Networks

In the study of neural networks, a recurrent neural network (RNN) is a network of nodes (neurons) designed to process sequential data and discover patterns within said data [42]. An RNN consists of three layers: the input layer, hidden layer and output layer. We call such an RNN deep if the hidden layer consists of multiple layers. To start, an input is passed into the input layer. Some function then maps the value at each node of the input to a value on some node in the first hidden layer. Then, each layer maps to the next using some function, until the output layer is reached. The results are then read out, and the model is trained to read out this output and convert it into a result. The state of each layer changes as the input propagates through the layers. The term recurrent in recurrent neural network stems from the fact that the previous state of a hidden layer is also passed into the new state of that same layer in some arbitrary way. The structure of such a network can be seen in figure 3.

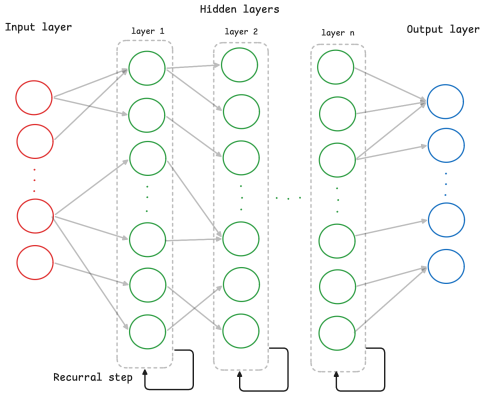

Figure 3: *An example network topology for a Recurrent Neural Network. Hidden layers are represented with dotted lines, and black arrows are used to represent the recurrent step.*

RNNs are well suited to processing timeseries, as they do not only observe the current state, but also know about every previous state through the recurrence. To better illustrate this fact, let us first state a general mathematical formulation of such an RNN [43].

Definition 2 (Recurrent Neural Network)**.** *Let $U_t \in \mathbb{R}^p$ and $O_t \in \mathbb{R}^q$ be some p-dimensional input and q-dimensional output respectively, both at time t. Furthermore, let $(H_t^1, ...H_t^L)$ be a tuple of $L \in \mathbb{N}$ hidden layers, with $H_t^{(l)} \in \mathbb{R}^{n_l}$ and n_l the number of nodes in the l-th hidden layer. Finally, let θ, ϕ be sets of arbitrary parameters. Then, a recurrent neural network with L hidden layers is a dynamical system of the form:*

$$\begin{cases} H_t^{(1)} = f^{(1)}\big(H_{t-1}^{(1)}, U_t; \theta^{(1)}\big) \\ H_t^{(l)} = f^{(l)}\big(H_{t-1}^{(l)}, H_t^{(l-1)}; \theta^{(l)}\big) \\ O_t = G\big(H_t^{(L)}; \phi\big) \end{cases} \tag{3.1}$$

with f^l and G being arbitrary functions with domains given by the Cartesian product of the domains of the inputs, that define the connections between nodes in the network.

As we can see, the hidden layer state acts like a summary of all previous states. In that way, our model can more easily remember the entire timeseries.

The parameters θ and ϕ are learnable. Through learning these parameters, the network will adapt its internal dynamics so that the output layer matches more and more closely with the desired targets. However, training these parameters can be computationally demanding and suffers from the vanishing and the exploding gradient problem [44–46]. This makes it nearly impossible to implement an RNN in physical hardware. A solution to these problems comes from a specific type of Recurrent Neural Network, called an Echo State Network.

3.2 The Echo State Network

As training the parameters of the hidden layers is an expensive operation that is prone to problems, a good simplification would be to fix the hidden layer, and instead learn to interpret the output layer. Figure 4 gives a visual on what such a network would look like.

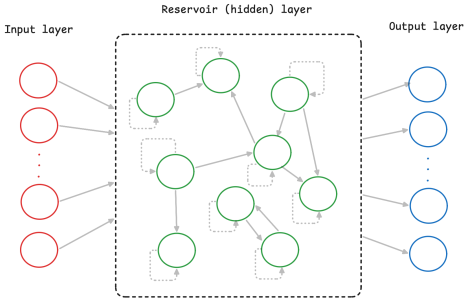

Figure 4: *An echo state network. Here, the recurrent relationship is encoded by the edges between nodes in the reservoir layer. Note that each node will connect to itself, representing the linear leaking term in Eq.* (3.3)*.*

Here, the hidden layer is called the reservoir and has

only one layer. If we mathematically view the edges (inter)between layers as matrices, we would have three matrices W to deal with: W^{in}, the edges between input layer and reservoir nodes; W^r, the edges between nodes in the reservoir layer; and W^o, the edges between the reservoir nodes and the output nodes. For a small example, let us look at the following toy ESN and define what these matrices would look like:

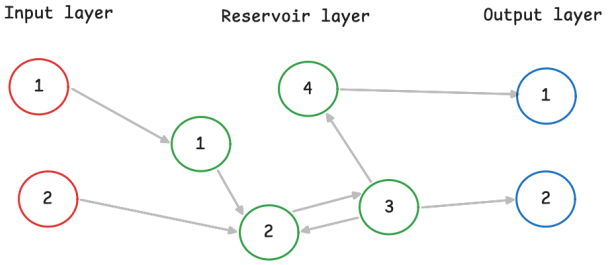

Figure 5: *A simple ESN with small input, output and reservoir layers. The self edges representing the leaking term are ignored.*

For simplicity, let us assume each of the edges is weighted with 1. Then, the matrix is simply the connectivity matrix of the network. We find

$$W^{in} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}; \qquad W^r = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Then, we would train the output matrix W^o for our specific task. In an ESN, only W^o is trainable, and the rest are fixed properties of the system. This greatly reduces computational complexity of this machine learning method. Besides this, having a fixed internal weight matrix also makes it easier to realize ESNs in physical circuits.

The final component we need to mention before moving on to the mathematical definition of an ESN is the Echo State Property (ESP). Heuristically, the ESP tells us that an ESN is able to forget its initial conditions, such that the internal state of the reservoir will entirely depend on the past history of the input signal after a certain amount of time has elapsed.

Definition 3 (Echo State Property)**.** *An ESN with reservoir states $x(t)$ has the Echo State Property if for any compact $C \subset \mathbb{R}^K$ and any two starting states $x(0)$ and $x'(0)$, there exists a sequence $(\delta_h)_{h=0,1,2,\ldots}$ that converges to 0 such that for any input sequence $(u(t))_{t=0,1,2,\ldots} \subseteq C$ it holds that*

$$\|x(h) - x'(h)\| \leq \delta_h. \tag{3.2}$$

From this, we can state a formal definition of an ESN. Here, we shall consider the discrete case, where we evaluate the dynamics of the ESN at discrete timesteps. Furthermore, we shall use lowercase letters to distinguish the ESN from the more general RNN.

Definition 4 (Echo State Network)**.** *Let f be some non-linear function $f : \mathbb{R}^n \to \mathbb{R}^n$, and let $x \in \mathbb{R}^n$ and $o, u \in R^d$ be states at discrete time $t \in \mathbb{N}$. Furthermore, let $\alpha \in [0, 1]$. A Leaky Echo State Network is an RNN with 1 hidden layer, which we call the internal reservoir, having the ESP and obeying the following dynamical equations:*

$$\begin{cases} x[t] = \alpha f\big(W^r x[t\text{-}1] + W^{in} u[t\text{-}1]\big) + (1\text{-}\alpha) x[t\text{-}1] \\ o[t] = W^o x[t] \end{cases} \tag{3.3}$$

where $W^r \in \mathbb{R}^{n \times d}$ (a matrix with sparsity s) and $W^{in} \in \mathbb{R}^{n \times n}$ are fixed, and $W_o \in \mathbb{R}^{d \times n}$ is to be trained using target samples.

Here, we call α the leaking rate, and observe that setting $\alpha = 1$ leaves us with an ESN without any leaking. In literature, both leaking and non-leaking ESNs are used [47, 48], so we have decided to provide a general definition covering both these cases.

To complete our model of the ESN, let us look at how to extract the output matrix W_o from this model. The easiest way to do this is to optimize our output matrix using ridge regression. This provides a combination of a least squares metric that tries to optimize correctness of the model and a quadratic ridge term meant to prevent overfitting, tuned by $\lambda \approx 10^{-6}$. We get the following equation, which we can easily solve numerically:

$$\hat{W}_o = \arg\min_{W_o} \Big(||W_o x[t] - u[t]|| + \lambda ||W_o||_F^2 \Big),$$

where $\|\cdot\|$ is the usual Euclidean norm, and $\|\cdot\|_F$ is the Frobenius norm on matrices:

$$\|A\|_F = \sqrt{\sum_i \sum_j |A_{ij}|^2}.$$

3.3 Memristor-based ESN

Having created a model of ESNs, let us finally look at how such a network can be achieved using iontronic memristors. For this physical implementation, we will be using the conical SVM as introduced in section 2.3, while following the outline of [19]. To guide this section, let us first introduce what an ESN would look like using SVM's, to then derive that such a circuit indeed obeys Eq. (3.3).

From Fig. 6, we can see what components make up a single reservoir node in our ESN. Each node has an input terminal and an output terminal, where in the middle a parallel resistor and memristor make sure that the nonlinear update behavior at each node is captured. Not shown here are the connections from the input and output layer. We can imagine that at each input terminal of a reservoir node, an edge from the input layer may exist, and similarly to the output layer from the output terminals.

Now, between each of the terminals, we will have a current flowing in parallel. The value of the resistor at each node

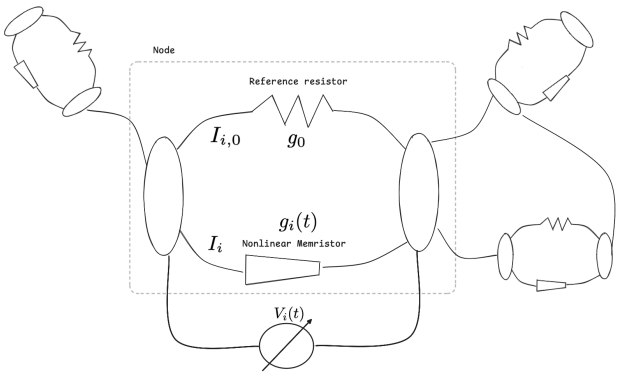

Figure 6: *A slice of the topology of a physical ESN. At each node, we have a resistor and memristor that define the state of the node (right ellipse) based on the current that is put into the node (left ellipse).*

is chosen specifically to be the steady state conductance at zero volts, g_0. This is done to make sure that the peripheral circuit that encodes the voltage update rules has a reference current from which it can update the voltage without needing to know the exact voltage value at each terminal. These current-to-voltage exchanges will thus be handled by the peripheral circuit, which is a quite standard procedure in the world of neuromorphic computing [49]. By the voltage over the node, a current is created. The current through the reference resistor of the i-th node will then be of value $I_{i,0} = g_0 V(t)$, while the current through the memristor will be the standard memristor value $I_i = g_i(t)V_i(t)$ as per Eq. (2.4)a.

The peripheral circuit is set up in such a way that for a network as in Def. 4, the voltage over node i at time t is described by

$$V_i(t) = \sum_j W^r_{ij}\Big(\frac{I_j}{I_{j,0}} - 1\Big) + \sum_j W^{in}_{ij} u_j(t). \tag{3.4}$$

Finally, we conduct some simplifications on the conical SVMs to make the correspondence with an ESN work. Firstly, we simplify the circuit by making all SVMs physically identical. The proof that will follow also extends to when this is not the case, but for this thesis, that case is irrelevant. Secondly, we convert the memristor conductance to a normalized dimensionless conductance to better match the state variable of the ESN model to this conductance. The dimensionless and shifted conductance $\tilde{g}(t)$ will be defined as

$$\begin{aligned} \tilde{g}_i(t) &= \frac{g_i(t) - g_0}{g_0}, \\ \tilde{g}_{i,\infty} &= \frac{g_{i,\infty}(V_i) - g_0}{g_0}. \end{aligned} \tag{3.5}$$

We are now ready to prove our correspondence.

Theorem 1 (Correspondence to ESN)**.** *Let our circuit be as in Fig. 6, and let the memristors be identical with conductance normalized as in Eq.* (3.5)*. Furthermore, let some peripheral circuit be introduced so that voltage updates based on current as in Eq.* (3.4)*. Then, there is a correspondence between this circuit and the mathematical definition of an ESN (4).*

Proof. First, observe that Eq. (2.4)b turns into the dimensionless equation

$$\begin{aligned} \dot{\tilde{g}}(t) &= \frac{1}{g_0}\frac{\big(g_\infty(V_i) - g_0\big) - \big(g_i(t) - g_0\big)}{\tau} \\ &= \frac{\tilde{g}_\infty(V_i) - \tilde{g}_i(t)}{\tau}. \end{aligned}$$

Then, we can see from Eq. (3.4) that

$$\begin{aligned} V_i(t) &= \sum_j W^r_{ij}\Big(\frac{I_j}{I_{j,0}} - 1\Big) + \sum_j W^{in}_{ij} u_j(t) \\ &= \sum_j W^r_{ij}\Big(\frac{g_j(t)V_j(t)}{g_0 V_j(t)} - 1\Big) + \sum_j W^{in}_{ij} u_j(t) \\ &= \sum_j W^r_{ij}\Big(\frac{g_j(t) - g_0}{g_0}\Big) + \sum_j W^{in}_{ij} u_j(t) \\ &= \sum_j W^r_{ij}\tilde{g}_j + \sum_j W^{in}_{ij} u_j(t). \end{aligned}$$

where we used the definition of the dimensionless conductance as per Eq. (3.5). Note that the second line may have division by zero. However, as the same term appears in the numerator and denominator, we assume for $V_j(t) = 0$ that we may take the limit of this division, which is 1. From here, we can cast this into a vector form to find:

$$V(t) = W^r\tilde{\boldsymbol{g}} + W^{in}\boldsymbol{u}.$$

Inserting this equation for $V(t)$ into our SVM equation of motion, we find:

$$\dot{\tilde{\boldsymbol{g}}} = \frac{\tilde{g}_\infty(W^r\tilde{\boldsymbol{g}} + W^{in}\boldsymbol{u}) - \tilde{\boldsymbol{g}}(t)}{\tau}.$$

Finally, taking the Euler Discretization of this equation of motion, now simplifying our vector notation back to the usage of g with this representing the dimensionless vector, we have:

$$\frac{g[t+1] - g[t]}{\Delta t} = \frac{dg}{dt}[t]$$

Inserting this into our equation, we finally find

$$g[t] = \alpha g_\infty\big(W_r g[t-1] + W_{in}u[t-1]\big) + (1-\alpha)g[t-1] \tag{3.6}$$

where we have taken $\alpha = \frac{\Delta t}{\tau}$, which we shall call the leaking rate, as it mimics the leaking term in the li-ESN. Note that for α to have domain $[0,1]$, we will be enforcing $\Delta t < \tau$ for our physical circuit.

The correspondence is now clearly visible, where we point out the correspondences:

- $g_\infty \sim f$, the nonlinear function as in Fig. 2.

- $g[t] \sim x[t]$, the dimensionless state variable
- $o[t] = W_o g[t]$, which can be achieved by means of the peripheral circuit reading the outputs at each terminal node.

This proves that we have a correspondence between our physical circuit and the ESN. □

3.4 Simulation of the physical ESN

Now that we have shown that a physical ESN network is achievable using memristors, we will look into how well this Echo State Network actually functions when simulated.

The Echo State Network source code functions in a few steps, and requires nothing more than initialization of the hyperparameters of the system and some input sequence that we wish to learn. For the ESN, the hyperparameters of note are laid out in Appendix B, along with the default values used in the simulations

Here, we let W^{in} and W^r be scaled by some scalar, in order to make tuning of these matrices more intuitive in the codebase. Finding a good set of hyperparameters will always depend on the task at hand, and specifically the value of τ will have to be adjusted to fit the internal timescale of the timeseries we are trying to predict.

For us to test out how effective the model is at predicting near-chaotic timeseries, let us introduce the series on which we will be basing must of our metrics.

Definition 5 (Mackey-Glass series)**.** *Let $\zeta, \eta, \theta, n, t \in \mathbb{R}_{>0}$, and let $x : \mathbb{R} \to \mathbb{R}$ be some continuously differentiable function. Furthermore, let $\phi : [-\theta, 0] \to \mathbb{R}$ be some arbitrary function. Then, the Mackey-Glass equation is the differential equation:*

$$\begin{cases} \frac{dx}{dt} = \frac{\zeta \phi(t-\theta)}{1+\phi(t-\theta)^n} - \eta x(t), & t \in [0, \theta] \\ \frac{dx}{dt} = \frac{\zeta x(t-\theta)}{1+x(t-\theta)^n} - \eta x(t), & t > \theta \end{cases} \tag{3.7}$$

This series is a so-called delay differential equation, and its behavior mimics that of certain biological processes like the production of blood cells [51]. Note that we used non-standard symbols for the different parameters to avoid confusion with our main definitions. Standard values to use for ζ, η, θ and n are $0.2, 0.1, 17$ and 10 respectively [19, 52, 53]. Figure 7 shows what such a timeseries may look like using these standard values.

For the simulation, we will try to use the ESN to predict the Mackey-Glass series. More details follow in Section 7.3. To measure accuracy of our prediction, the Valid Prediction Time measure is used [54, 55].

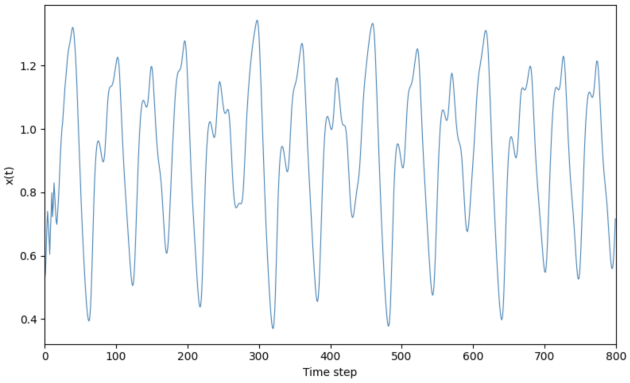

Figure 7: *An example of the Mackey-Glass series for $\theta = 17$, which is chaotic in nature.*

Definition 6 (Valid Prediction Time)**.** *Let $y(t)$ be the timeseries we wish to predict, and $\tilde{y}(t)$ its predicted value. Then, the Valid Prediction Time (VPT) is defined as:*

$$t_{VPT} = min_t \Big\{ \frac{\big(y(t) - \tilde{y}(t)\big)^2}{var(y)} > \delta \Big\} \tag{3.8}$$

Here, the value $\delta = 0.4$ is commonly used in literature [54, 55].

We will return to the specifics on simulating the ESN in Chapter 7, and will look at our results in Chapter 8. However, having some basic knowledge of the simulation helps contextualize the coming chapters. For now, we would first like to focus on the theory of chaos, how to measure it in systems, and why it is desirable in Echo State Networks.

4. Chaos theory

Dynamical systems are a large part of nature, from cellular processes to population dynamics. Classifying these dynamical systems, we can view them as either being chaotic or non-chaotic. In the non-chaotic case, small changes in initial conditions will produce predictable and small changes in time. On the contrary, chaotic systems will see small initial changes blow up in what is called the butterfly effect in mainstream media.

Non-chaotic systems tend to converge to a stable value and lose information on the internal dynamics, while chaotic systems tend to diverge and lose causality in output. Both of these properties are not ideal for systems that need to adapt to their environment. Because of this, most dynamical systems find themselves working a balancing-act, trying to stay as close as possible to the interface between chaotic and non-chaotic [56]. Machine learning systems like our ESN is another example of systems that perform best at the edge of chaos [57]. To better explore this, let us try to understand what it means exactly to be performing at the edge of chaos.

4.1 Maximal Lyapunov Exponent

The most common way to mathematically express the notion of whether small changes in initial conditions also lead to small changes in eventual state through a systems Lyapunov exponents. In particular, the Maximal Lyapunov Exponent (MLE) is commonly used to discern between ordered and chaotic dynamics. It is defined as follows:

Definition 7 (Maximal Lyapunov exponent)**.** *Consider a discrete dynamical system with state $x[t]$ and let x_0 be its initial condition. Now, let a second trajectory start from the perturbed initial condition $x_0 + \delta_0$, where $\|\delta_0\|$ is infinitesimal. Denoting the separation between the two trajectories at time t by $\delta(t)$, we define the Maximal Lyapunov Exponent (MLE) as:*

$$\lambda_{\max} = \lim_{t\to\infty} \lim_{\|\delta_0\|\to 0} \frac{1}{t} \ln \frac{\|\delta(t)\|}{\|\delta_0\|}.$$

where for sufficiently small perturbations, the separation will evolve as:

$$\|\delta(t)\| \approx e^{\lambda_{\max} t} \|\delta_0\|.$$

Here, we can see that $\lambda < 0$ leads the perturbation to converge to zero. Likewise, a $\lambda > 0$ will cause the perturbation to grow exponentially. The edge of chaos we are looking for lies exactly at the interface, at $\lambda \approx 0$.

To compute the Lyapunov exponent for our discrete ESN, we will first be turning to a more theoretical basis to prove that it is actually meaningful to compute Lyapunov exponents for such complex orbits as with an ESN. Here, we will be employing measure theory [58].

Theorem 2 (Oseledets Multiplicative Ergodic Theorem [59])**.** *Let the triple $(M, \mathcal{A}, \rho)$ be a probability space and let $\tau : M \to M$ be a measure-preserving map. Let T be a measurable function from M to the space of all real $m \times m$ matrices such that*

$$\log^+ \|T(\cdot)\| \in L^1(M, \rho)$$

and

$$T_x^n = T_{\tau^{n-1}(x)} \cdots T_{\tau(x)} T_x.$$

Then there exists a $\Gamma \subseteq M$ with $\rho(\Gamma) = 1$ and $\tau(\Gamma) \subseteq \Gamma$, such that the following holds for all $x \in \Gamma$:

1. *$\Lambda_x := \lim_{n\to\infty}((T_x^n)^* T_x^n)^{1/2n}$ exists.*
2. *Let $\exp \lambda_x^{(1)} < \cdots < \exp \lambda_x^{(s)}$ be the distinct eigenvalues of Λ_x, where $s = s(x)$, the $\lambda_x^{(r)}$ are real, and $\lambda_x^{(1)}$ may equal $-\infty$. Let $U_x^{(1)}, \ldots, U_x^{(s)}$ be the corresponding eigenspaces and set $m_x^{(r)} = \dim U_x^{(r)}$. The functions $x \mapsto \lambda_x^{(r)}$ and $x \mapsto m_x^{(r)}$ are τ-invariant. Define $V_x^{(0)} = \{0\}$ and*

$$V_x^{(r)} = U_x^{(1)} \oplus \cdots \oplus U_x^{(r)}, \quad r = 1, \ldots, s.$$

Then for all $u \in V_x^{(r)} \setminus V_x^{(r-1)}$, $1 \le r \le s$,

$$\lim_{n\to\infty} \frac{1}{n} \log \|T_x^n u\| = \lambda_x^{(r)}.$$

From this theorem, we are able to identify the phase space of the dynamic system with M, and we can see that τ corresponds to the dynamical equation that drives the conductance $g \sim x$ forward. Then, what the first part of the theorem tells us is that the Lyapunov exponents exists. The second part of the theorem tells us that the Lyapunov exponents are τ invariant, i.e. unique to the system and independent of the trajectory through phase space.

Furthermore, from theorem 2, we can identify the Jacobian matrix J as a measurable function from M to the space of real $m \times m$ matrices where the chain rule of the Jacobian gives us that

$$J_x^n = J_{\tau^{n-1}(x)} \ldots J_{\tau(x)} J_x \tag{4.1}$$

With the Lyapunov exponents thus being given by

$$\lambda^{(r)} = \lim_{n\to\infty} \frac{1}{n} \log \|J_x^n u\|, \quad u \in V_x^{(r)} \setminus V_x^{(r-1)}, \tag{4.2}$$

where $\setminus$ represents the set difference. For the chaotic behavior of the system, we are interested in the maximal Lyapunov exponent $\lambda^{(s)}$, which dominates the long-term growth of perturbations. By the Oseledets theorem, this is given by the largest eigenvalue of the limit matrix Λ_x, and is again τ-invariant along the orbit.

To compute this in practice for the ESN, we note that directly evaluating J_x^n is computationally difficult for large n, as the matrix product grows with the trajectory. Instead, we will work by tracking a vector $u \in V_x^s / V_x^{s-1}$ forward under the action of the Jacobian at each step, normalizing it to take care of numerical overflow:

$$\lambda \approx \frac{1}{n} \sum_{t=0}^{n-1} \log \|J_{f^t(x)} \hat{u}_t\| \tag{4.3}$$

where $\hat{u}_t$ is the normalized tangent vector at step t. This sum converges to the maximal Lyapunov exponent as $n \to \infty$, as guaranteed by Oseledets theorem.

Besides this theoretical method, one can convince themselves that a good way to measure the chaotic behavior heuristically is to look at eigenvalues of J directly at different times. The closer they remain to the unit circle, the more the system tends to the edge of chaos. For eigenvalues of low magnitude, we can presume the system behaves non-chaotically. If the eigenvalues remain above 1 in magnitude, the system will have chaotic behavior.

Then, instead of going off of equation (4.3) directly, we will be looking at the Jacobian eigenspectrum to see how the vector u is perturbed by this Jacobian matrix through time, a measure that we will further validate in the next chapter when we talk about the Edge of Stability Echo State Network.

4.2 Disclaimer

Now, we have a method of computing the Lyapunov exponents, using Oseledets theorem to give us a theoretical baseline to go off on. The problem, however, is that we have completely ignored the input sequence on which our dynamical system depends. The literature on chaos theory is mostly based on autonomous dynamical systems of the form $x[t] = f(x[t-1])$ with f some function. For our Echo State Network, however, we are dealing with a non-autonomous dynamical system. For such systems, we have that $x[t] = f(x[t-1], u[t-1])$, i.e. the dynamics are not only driven by how the internal state evolves, but also by some external input. This is a problem often ignored in machine learning research [60], and much research on these systems is still open [61].

Because of this, we must note that the general notions of ergodicity on which modern chaos theory relies do not necessarily apply to our specific case of the Echo State Network. However, it has recently been shown that the Mackey-Glass system for $\zeta > \eta$ admits a non-trivial invariant measure [61]. In this same paper, numerical evidence strongly suggests it is ergodic, so that we will assume this is the case for the sake of simplicity. With this the case, we are still allowed to use Oseledets theorem [62], so that our initial methods are still suggested to be valid.

Going into more depth on this topic goes beyond the scope of this thesis, but it is good to know that even though usually non-autonomous dynamics are ignored for machine learning research, there is a strong basis based on [61] to assume that our methods of calculating the Lyapunov exponents still hold.

4.3 Chaos analysis of the ESN

Returning briefly to the heuristic evaluation of chaos of a dynamical system based on the instantaneous Jacobian eigenspectrum of the dynamic equation, we can look at this spectrum for our ESN to get a feel for how chaotic the system is. For different measurements in time and for different leaking rates, the result of this eigenspectrum are overlaid in Fig. 8.

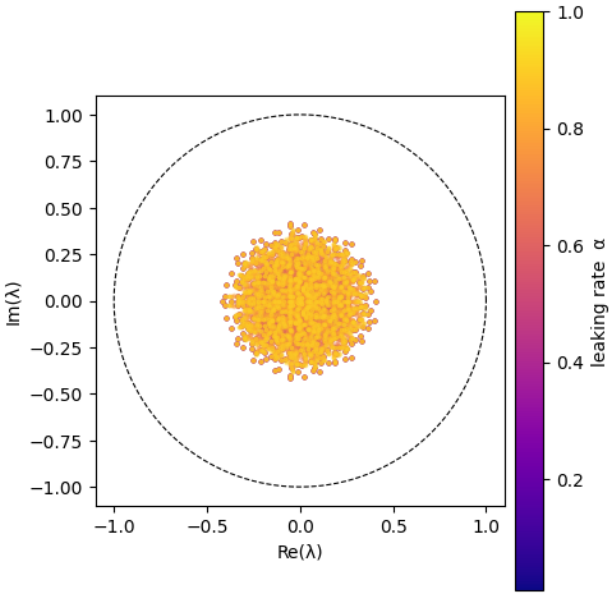

Figure 8: *The Jacobian Eigenspectrum of the ESN for initial values as in [50] and increasing values of the leaking rate α. As we can see, the leaking rate does not influence the spectrum much, and most values are clustered near the center of the graph.*

This heuristic is exactly the motivation for this study. With the Jacobian eigenvalues at individual times being this low, we can conclude that this physical implementation of an ESN does not operate near the edge of chaos regime.

The rest of this research will be structured around finding a solution to this problem, studying a new circuit topology in the hopes of showing that this does a better job at edge of chaos behavior, so that we may improve on the results of this current ESN.

5. Edge of Stability ESN

The solution we propose is based on the theoretical model of Edge of Stability Echo State Networks, which from now on will be abbreviated as ES^{2}Ns. The model of these ES^{2}Ns was proposed by Ceni and Gallicchio in 2023 [63].

The main idea of these ES^{2}Ns is that we introduce an orthogonal matrix to the linear part of the dynamic equation. In the original ESN as described by Eq. (3.3), the linear leaking term $(1-\alpha)g[t-1]$ looks a bit like a decay matrix where each eigenvector of the system has eigenvalue with value $(1-\alpha)$. The main addition that the ES^{2}N model introduces is that these singular eigenvalues will be distributed over a circle, effectively decoupling the memory timescales of the system and adding favorable theoretical properties in the process.

5.1 Definition of the ES^{2}N

To get a feeling for what an ES^{2}N actually entails, we will start by stating its definition and building intuition on what such a network would look like in general.

Definition 8 (Edge of Stability Echo State Network)**.** *Let f be some non-linear function $f : \mathbb{R}^n \to \mathbb{R}^n$. Furthermore, let $\beta \in [0,1]$. Finally, let O be some arbitrary orthogonal matrix. An Edge of Stability Echo State Network is an RNN with 1 hidden layer, which we call the internal reservoir, obeying the following dynamical equations:*

$$\begin{cases} x[t] &= \beta f\big(W^r x[t-1] + W^{in} u[t-1] + c\big) \\ &\quad +(1-\beta)Ox[t-1] + g(u[t-1]) \\ o[t] &= W^o x[t], \end{cases} \tag{5.1}$$

where W^r and W^{in} are fixed and W^o is to be trained using target samples. Furthermore, c is a general constant that may be zero, and g is some function that does not depend on the state x.

As we can see, the definition closely mirrors that of Def. 4, only differing in the addition of an orthogonal matrix and some extra generalizations. Nonetheless, this additional matrix provides exactly what we need from the network in order to lift the standard ESN to the edge of chaos regime.

In this model, we randomly select some arbitrary orthogonal matrix, but are allowed to constrain it beyond the necessary constraint of it being orthogonal. This allows us to, for example, have its non-zero entries correspond to the non-zero entries of the internal matrix W^{in}. Theoretically, the matrix is thus also allowed to be sparse, although it is difficult to uniformly sample a random orthogonal sparse matrix. Nonetheless, this allows some extra flexibility in this matrix.

For an example of an ES^{2}N, we can look at Fig. 9

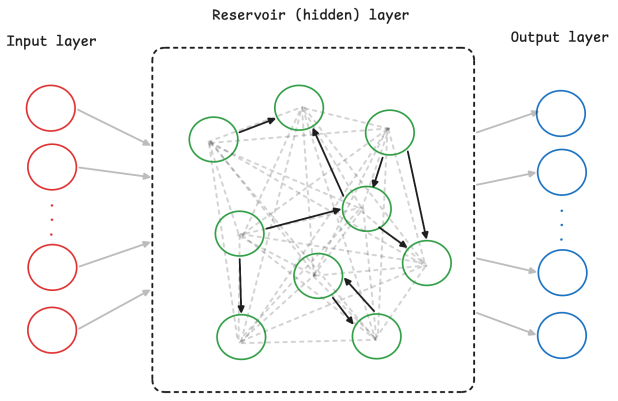

Figure 9: *An example scheme of the ES^{2}N. Here, the self edges of Fig. 4 are removed, and replaced by a dense array of edges representing the Orthogonal matrix of Def. 8.*

Of course, Fig. 9 is not the only topology possible, and we can convince ourselves that it is not a trivial extension of the physical ESN as per Fig. 6. Fortunately, we only need the dynamic equation to match to make use of all the interesting properties we will discuss and prove in the next subsection, and we will see that there is a way to make a circuit match these dynamics in section 6.

5.2 Mathematical properties

Let us build up the mathematical properties of the ES^{2}N from the ground up. For this, we will first be defining some constants.

Let us start by denoting the continuous ES^{2}N reservoir update rule as $X(u,x) = \beta f\big(W^r x + W^{in} u + c\big) + (1-\beta)Ox + g(u)$. Its derivative can then be written as:

$$\frac{\partial X}{\partial x}(u,x) = \beta D(u,x)\rho W^r + (1-\beta)O \tag{5.2}$$

$$D(u,x) := diag\Big(f'(\rho W^r x + \omega W^{in} u + c)\Big) \tag{5.3}$$

where we have taken the prefactors ρ and ω out of the matrices W^r and W^{in}, respectively. These will aid us in our future proofs. The diagonal matrix D has a supremum, which we shall denote by $\gamma := \sup_{u,x} \|D\|$.
We denote for a specific input-driven trajectory $\{u[t], x[t]\}$ for $0 < t < T$ the jacobian map of (5.2) to be:

$$J[t] = \frac{\partial X}{\partial x}(u[t], x[t]) \tag{5.4}$$

Lastly, we denote the maximum singular value of the matrix ρW^r to be $\sigma := \|\rho W^r\|$.

We are now ready to prove that our system has the Echo State Property as per Def. 3, and that it is just a system that does not depend on its initial condition in the limiting case.

5.2.1 ES^{2}N ESP

For this proof, we will be following in the footsteps of [63, 64]. This is the most important result we will prove, as it

proves that our new network is of the same class and can solve the same class of problems as our original ESNs.

Theorem 3 (ES^{2}N Echo State Property)**.** *Let our ES^{2}N be defined as in Def. 8, and let $|f'| \leq 1$, $\sigma < 1$ and $\beta > 0$. Then, the ES^{2}N has the ESP for all inputs.*

Proof. Following the proof structure of ([64], Proposition 3), proving this is equivalent to proving that

$$\|\frac{\partial X}{\partial x}\| < 1$$

for all x, u. From the Triangle Inequality and Cauchy Schwarz inequality, we can see that:

$$\begin{aligned}\|\frac{\partial X}{\partial x}\| &= \|\beta D(u,x)\rho W^r + (1-\beta)O\| \\ &\leq \|\beta D(u,x)\rho W^r\| + \|(1-\beta)O\| \\ &\leq \beta\|D(u,x)\|\sigma + (1-\beta)\|O\|\end{aligned}$$

Where we can finally use that we presumed $|f'| \leq 1$ and $\sigma < 1$ combined with the fact that $\|O\| \leq 1$ for orthogonal matrices to arrive at the conclusion that

$$\begin{aligned}\|\frac{\partial X}{\partial x}\| &\leq \beta\|D(u,x)\|\sigma + (1-\beta)\|O\| \\ &< \beta + (1-\beta) = 1\end{aligned}$$

Thus, we can conclude that the ESP holds. □

Next, we will prove two results related to the eigenvalue spectrum of the Jacobian of this new system.

5.2.2 Annular Eigenvalue Spectrum

Next, we prove the fact that the eigenvalue spectrum of the ES2N is annular, even when the orthogonal matrix O is sparse. This result will show us that the eigenvalues will always be spread across a circle (decoupling them) and are restricted to some controllable region.

Theorem 4. *Let us consider an ES2N model whose state update equation is given by eq. (5.1), and recall the definitions of σ and γ. Then, the eigenspectrum of the Jacobian of the ES2N map is confined in the annular neighbourhood of radius $\beta\gamma\sigma$ of the circle centered in the origin of radius $1-\beta$. In formulas, for each eigenvalue μ of the Jacobian matrix there exists a $\theta \in [0, 2\pi)$ such that*

$$\left|(1-\beta)e^{i\theta} - \mu\right| \leq \beta\gamma\sigma. \tag{5.5}$$

We will prove this using the Bauer-Fike theorem [65].

Proof. Define $A = (1-\beta)O$ and $E = \beta D(u,x)\rho W_r$, so that the Jacobian of the ES2N model is $\frac{\partial X}{\partial x}(u,x) = A + E$. The matrix O is orthogonal, hence there exists a unitary matrix V such that $O = V\Lambda V^{-1}$, where Λ is the diagonal matrix of the eigenvalues of O. In particular, each eigenvalue of O is of the form $e^{i\theta}$, for some argument $\theta \in [0, 2\pi)$, due to the orthogonality of O. Therefore, $A = (1-\beta)V\Lambda V^{-1}$, and all eigenvalues of A have the form $(1-\beta)e^{i\theta}$, for some argument $\theta \in [0, 2\pi)$. In other words, all the eigenvalues of A lie on the complex circle centered in the origin with radius $(1-\beta)$. Now, since V is unitary, we have that $\|V\| = \|V^{-1}\| = 1$. Therefore, the Bauer-Fike theorem tells us that for each μ eigenvalue of $\frac{\partial X}{\partial x}(u,x)$ there exists a complex number $\lambda = (1-\beta)e^{i\theta}$ such that

$$\left|(1-\beta)e^{i\theta} - \mu\right| \leq \|\beta D(u,x)\rho W_r\| \leq \beta\gamma\,\|\rho W_r\| = \beta\gamma\sigma. \tag{5.6}$$

This effectively proves our result when we take into consideration the definition of the Jacobian of our system. □

Besides showing the annular region, we can see that the tightening is controlled in part by hyperparameter β. The annulus can be tightened to make the actual eigenvalue spectrum more stable and controllable. This is the final property on which we would like to expand.

5.2.3 Tunable Lyapunov exponent

Finally, we wish to state a property using a heuristic argument. We know that all eigenvalues are bounded to a tight annulus. From there, we can specifically denote that

$$1 - \beta(1+\gamma\sigma) \leq |\mu_n| \leq 1 - \beta(1-\gamma\sigma)$$

Now, taking logarithms and applying first order approximation $\log(1-\beta) \approx -\beta$ for small β, we can see that

$$-\beta(1+\gamma\sigma) \leq \log(|\mu_n|) \leq -\beta(1-\gamma\sigma)$$

Assuming that the eigenvalues of the Jacobian are distributed approximately uniformly across the annulus, we estimate the maximal Lyapunov exponent to then be the average of the logs of these eigenvalues (4.3), i.e.

$$\lambda_{\max} \approx \frac{1}{2}[-\beta(1+\gamma\sigma) + (-\beta(1-\gamma\sigma))] = -\beta \tag{5.7}$$

From this, we conclude that for whatever arbitrarily small β we choose, we can get an Maximal Lyapunov Exponent that approaches zero from below. That is, for small β, our system approaches the Edge of Stability from stability.

From these properties, we conclude it would be fruitful to look for a physical way to extend our original circuit to an ES^{2}N circuit.

6. Physical ES^{2}N

In this section, we will aim to create a physical implementation of the ES^{2}N. To get back at our original goal of lifting the existing circuit to the edge of chaos regime, we will try to build on top of the original model introduced in section 3 by adding minimal changes, so that it is a logical step forward that does not become much more difficult to realize.

The main problem that we are facing is that we are trying to create orthogonal coupling between nodes in the circuit outside of the non-linear update rule. Because of this, it is not as easy as adding connections of memristors between the different nodes, as this will just result in more non-linear complexity that does not translate to the non-linear part. With the circuit implementation that we had, there were three main roads that would lead to some linear response: changing either the device dynamics or the peripheral circuit, or introducing new circuit elements.

In terms of the device dynamics, this would not be a great solution in our case. Conical fluidic iontronic devices already live at the interface of what experimentally realizable, and any more complexity would cause delay in experimentation. If this were not a problem, however, the way forward would be to have ions in channels share a larger pool of ions from which ions could flow between the different memristors, thus effectively bypassing the nonlinear update rules. The theory on this has, however, not been researched yet, and wether this road is worth exploring is yet unknown.

The second approach would be to change the peripheral circuit. We could take the nonlinear input and linearize it, for example. This, however, is not a trivial matter [66], and will cause the peripheral circuit to consume much more energy. This is undesirable, and not a satisfactory solution to our problem.

Finally, we could try to add new circuit elements. The nice part about working with conductance is that Kirchhoffs laws tell us that they are additive in parallel. As it turns out, using the Linear Conical Memristors as introduced in Def. (2.6), we are able to achieve exactly our goal. In this section, we will be exploring the resulting physical circuit.

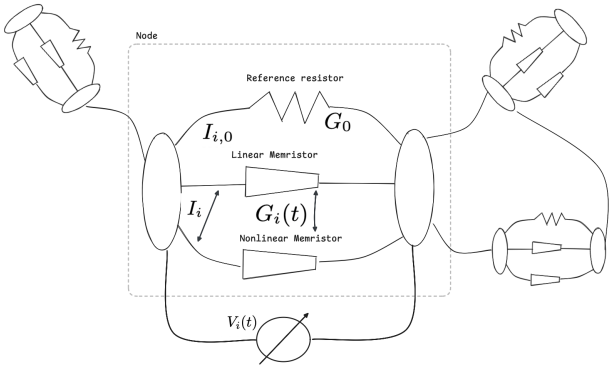

Figure 10: *A slice of an ES^{2}N. Here we have at each node a linear and nonlinear memristor, and a resistor, all in parallel. The total conductance and current here are taken over both pathways that contain memristors. We can then view this as essentially one larger memristor with a given total conductance $G_i(t)$.*

6.1 ES^{2}N circuit

The ES^{2}N circuit we propose is formed by the addition of a singular linear conical memristor element at each node, in parallel to the already existing circuit elements of Fig. 6. We have updated that circuit to match our new topology in Fig. 10.

We immediately stumble onto one limitation on the orthogonal matrix from this topology; the orthogonal matrix must encode the same edges as the internal reservoir of the system. We could theoretically work around this constraint, for example by branching the connection at the nonlinear memristor directly into different nodes, but the mathematics become more messy this way, and we do not truly gain anything from doing this besides added complexity. The main reason why we can still have this work is that both the internal reservoir and orthogonal matrix are allowed to be both sparse and randomly instantiated. Thus, this added constraint does not bother us.

From this point forward, we will be using tES^{2}N when referring to the theoretical ES^{2}N, and pES^{2}N when referring to the physical circuit in 6. Specifically:

Definition 9. *(tES^{2}N and pES^{2}N) We define the tES^{2}N as the theoretical model of an Edge of Stability Echo State Network, obeying the general definition of Definition 8.*

We define the pES^{2}N as the physical memristor-based implementation of such a tES^{2}N, with internal state obeying Eq. (6.3) *and generation of the orthogonal matrix being constrained as per Section 7.2.*

For both systems, default hyperparameters are given by Appendix B.

6.2 Mathematical pES^{2}N

We will start of our model in much the same way we did for the theoretical ESN: by proving that there is indeed a correspondence between the pES^{2}N and the ES^{2}N. To start this off, we will first need to look at voltage and dimensionless total conductance of the system.

We can imagine at each node that there is a current flowing in parallel between the two terminals. The value of the memristor will once again be the steady state conductance, where we must now define this steady state conductance as the combined one for the linear and non-linear memristor. As we have a linear and nonlinear memristor in parallel, the total conductance is additive, and we can denote it by

$G = g^l + g$. Then, we can update the equation of motion of this total conductance to be

$$\begin{cases} I(t) = G(t)V(t) & \text{(a)}, \\ \dot{G}(t) = \dfrac{g_0^l + bV + g_\infty(V) - G(t)}{\tau} & \text{(b)}. \end{cases}$$

Now, we can make these equations dimensionless in the same way we did for Eq. (6.1). Then, we find that:

$$\begin{aligned} \tilde{G}_i(t) &= \frac{G_i(t) - G_0}{G_0}, \quad \tilde{g}_{i,\infty} = \frac{g_{i,\infty}(V_i) - G_0}{G_0}, \\ \tilde{b} &= \frac{bV_c}{G_0}, \qquad\qquad \tilde{g}_0^l = \frac{g_0^l - G_0}{G_0} \end{aligned} \tag{6.1}$$

Here, the value of G_0 is now the steady state conductance at zero volts for both memristors combined, so $G_0 = g_\infty(0) + g_0^l$, and V_c represents some reference voltage (set to 1 Volts) to make the coupling dimensionless. Having dimensionless values now makes it easy for us to use literature values for those circuits for this circuit as well. Now, we set up our peripheral circuit in the same way as for the ESN, so that our voltage update rule matches Eq. (3.4). Now, everything is set up for us to prove the following correspondence.

Theorem 5 (Correspondence to ES^{2}N)**.** *Let our circuit be as in Fig. 10, and let all memristors of the same type be identical with conductance normalized as in Eq.* (6.1)*. Furthermore, let some peripheral circuit be introduced so that voltage updates based on current as in Eq.* (3.4)*. Then, there is a correspondence between this circuit and the mathematical definition of an ES^{2}N (8).*

Proof. First, we note the dimensionless update rule for the total conductance:

$$\begin{aligned} \dot{\tilde{G}} &= \frac{1}{G_0\tau}\Big(\big(g_0^l - G_0\big) + \big(bV - G_0\big) \\ &\quad + \big(g_\infty(V_i) - G_0\big) - \big(G_i(t) - G_0\big)\Big) \\ &= \frac{\tilde{g}_0^l + \tilde{b}V + \tilde{g}_\infty(V_i) - \tilde{G}_i(t)}{\tau}. \end{aligned}$$

Then, we can see from (3.4) that

$$\begin{aligned} V_i(t) &= \sum_j W_{ij}^r\Big(\frac{I_j}{I_{j,0}} - 1\Big) + \sum_j W_{ij}^{in} u_j(t) \\ &= \sum_j W_{ij}^r\Big(\frac{G_j(t)V_j(t)}{G_0 V_j(t)} - 1\Big) + \sum_j W_{ij}^{in} u_j(t) \\ &= \sum_j W_{ij}^r\Big(\frac{G_j(t) - G_0}{G_0}\Big) + \sum_j W_{ij}^{in} u_j(t) \\ &= \sum_j W_{ij}^r \tilde{G}_j + \sum_j W_{ij}^{in} u_j(t), \end{aligned}$$

where we used the definition of the dimensionless conductance of Eq. (3.5). From here, we can cast this into a vector form to find:

$$V(t) = W^r\tilde{\boldsymbol{g}} + W^{in}\boldsymbol{u}.$$

Inserting this equation for $V(t)$ into our SVM equation of motion, we find:

$$\dot{\tilde{\boldsymbol{G}}} = \frac{\tilde{g}_\infty(W^r\tilde{\boldsymbol{G}} + W^{in}\boldsymbol{u}) + \tilde{b}W^r\tilde{\boldsymbol{G}} + \tilde{b}W^{in}\boldsymbol{u} + \tilde{g}_0^l - \tilde{\boldsymbol{g}}(t)}{\tau}$$

Then, taking the Euler Discretization of this equation of motion, now simplifying our vector notation back to the usage of G to represent the dimensionless vector, we have:

$$\frac{G[t+1] - G[t]}{\Delta t} = \frac{dG}{dt}[t]$$

Inserting this into our equation, we finally find:

$$\begin{aligned} G[t] =& \beta g_\infty\big(W^r G[t-1] + W^{in}u[t-1]\big) \\ &+ \big(I - \beta(I - bW^r)\big)G[t-1] + \beta(bW^{in}u[t-1] + g_0^l), \end{aligned}$$

where I represents the identity matrix and $\beta := \frac{\Delta t}{\tau}$ is a parameter we call the proximity. We will simplify this further by letting

$$\beta(bW^{in}u[t-1] + g_0^l) = C\big(u[t-1]\big).$$

Furthermore, we define

$$O := \frac{1}{1-\beta}\Big((1-\beta)I + \beta bW^r\Big). \tag{6.2}$$

Then, we can easily see that

$$(1-\beta)O = \Big((1-\beta)I + \beta bW^r\Big) = I - \beta\big(I - bW^r\big).$$

Finally, we arrive at our equation for $G[t]$:

$$\begin{aligned} G[t] =& \beta g_\infty\big(W^r G[t-1] + W^{in}u[t-1]\big) \\ &+ (1-\beta)OG[t-1] + C\big(u[t-1]\big) \end{aligned} \tag{6.3}$$

The correspondence is now clearly visible, where we point out the following correspondences:

- $g_\infty \sim f$, the nonlinear function as in Fig. 2.
- $G[t] \sim x[t]$, the dimensionless state variable.
- $o[t] = W_o g[t]$, which can be achieved by a peripheral circuit.
- O a matrix we can constrain to be orthogonal by placing constraints on W^r.
- $C\big(u[t-1]\big) \sim g(u[t-1])$ some state-independent additional term.

This proves that we have a correspondence between our physical circuit and the ES^{2}N as described in Def. 8. □

With this, we have proven that it is indeed possible to lift our initial circuit as per Fig. 6 to an ES^{2}N circuit by means of some minimal adjustments to the circuits. Namely, we require only that W^r is constrained in a way so that O as per Eq. 6.2 is orthogonal, and that we add one additional linear memristor at each node. Because our pES^{2}N is of the same class as the tES^{2}N, all of the mathematical properties as proved in Section 5.2 still hold.

Off course, it still remains to be verified that this actually produces the efficiency gain we expect it to. In order to further validate our new circuit, the rest of this thesis will focus on producing results through means of simulation.

7. Modelling an ES^{2}N

To start our simulation based research, we first have to model our pES^{2}N. In doing so, we will discuss the general scheme for modelling any Echo State Network, and how we train the output matrix W^o to correctly translate internal state to the next input state. Then, we pay attention to modelling the defining characteristic of the ES^{2}N in the orthogonal matrix. Finally, we look at the tests and measures we will use to compare our different models.

7.1 Code overview

The code used for the simulations is written in Python, and the codebase taken as inspiration is public on github [50]. The main program of running an Echo State Network can be divided into 5 parts, as illustrated in Fig. 14.

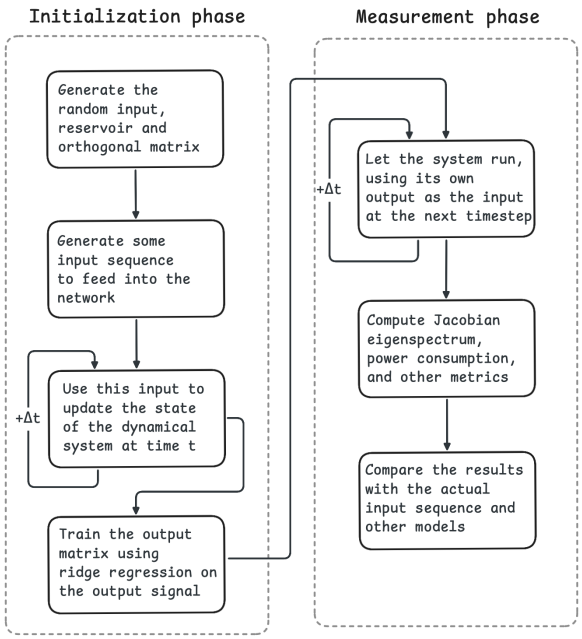

Figure 11: *An overview of the main steps our program takes in computing results for our physical networks.*

Let us quickly go over each of these steps. First, we need to initialize our model by generating the random matrices. For this, we will be employing pseudo-random number generators. The main reason we wish to randomly generate these matrices is so that we can draw conclusions about our networks not just for specific configurations, but in general for all configurations. For this to hold, we will be looking at many random networks when doing comparisons, so that we can assume the results we find tend to their expected values. There is some nuance when it comes to generating the orthogonal matrix, but we will dedicate an entire section to that problem.

Next, we have our input series. For the main results in this thesis, the Mackey Glass series will be used, as described by Eq. (3.7). For such a series, it is important that the hyperparameters of the network match the specific problem and internal frequencies for optimal predictive power. Because of this, hyperparameters like internal memristor timescales have to be carefully selected to give a fair comparison. These optimal scales are not always obvious, and we will be employing the python hyperparameter optimization framework "fortuna" to aid us in this regard.

After this, we are ready to have dynamical system run, using the input sequence to feed its input. We define a set time length for which we will be training the model on this input sequence, and we also define a washout period that we ignore for training purposes, as the system will still need to calibrate to match the input sequence well. After the training time has been completed, we use ridge regression to train our output model based on the input sequence. For this, we remember that $o[t] = W^o x[t]$, and remember the definition of ridge regression:

Definition 10 (Ridge regression)**.** *Let the tuple $(x[t], o[t]) \in \mathbb{R}^n \times \mathbb{R}^m$ be a training sample at timestep $t \in [0, T]$, where we define $t = 0$ as the period after washout. Furthermore, let us define $W^o \in \mathbb{R}^{m \times n}$ such that*

$$\hat{o}[t] = W^o x[t]$$

Let $\lambda \in \mathbb{R}$. Then, ridge regression is the method of finding some W^o such that the function

$$\sum_{t=0}^{T} \|W^o x[t] - o[t]\|^2 + \lambda \|W^o\|_F^2 \qquad (7.1)$$

is minimized. Here, the first term is the Mean Square Error function, measuring correctness of the matrix in matching the label $o[t]$. The second term is called the Ridge term, which prevents overfitting and reduces variance of $\hat{W}^0$ over different training samples.

The ridge regression as in Eq. (7.1) can be solved easily by a numerical solver. Then, we are left with choosing a suitable value of λ. To do this, the Optuna python package [67] will be swept over some values on a logarithmic scale, so that we can decide which ridge term works best for our model.

After letting this new model run on its own, we will be able to compute its Jacobian and Power consumption. For the power consumption, we have a known voltage, so that it is as simple as computing

$$P = \sum_i G_i V_i^2. \qquad (7.2)$$

Finally, some data analysis will be done, comparing the generated output with the actual timeseries and trying to understand how "well" the system adapts.

7.2 Orthogonal Matrix

Let us now pay some extra attention to the way the orthogonal matrix is constructed. Recall that Eq. (6.2) gives us an exact constraint on what our physical orthogonal matrix will look like. As the reservoir matrix W^r is arbitrary, this

also means that we can simply generate some matrix meeting our requirements of orthogonality, and then have W^r follow from that random matrix. In that case, we invert the equation to find:

$$W^r = \frac{(1-\beta)(O-I)}{\beta b} \tag{7.3}$$

Now, we are left with engineering O to meet our requirements of an orthogonal matrix. To start, we look at our requirements. Our matrix O must be:

- Sparse, where we will define its sparsity to be of value s. We want sparsity to be arbitrary and tunable, as higher sparsity results in lower computational complexity and cheaper builds.
- Orthogonal, as per the requirements of the ES^{2}N.
- Randomly generated, as stated previously.
- Local, in so far that localization supplies the system with a richer variation in dynamics. [68, 69].

Then, the first problem we will tackle is that of generating a random orthogonal matrix with sparsity s. An orthogonal matrix is a square matrix whose column vectors form an orthonormal set, which is equivalent to saying that it is a matrix Q for which:

$$Q^TQ = QQ^T = I$$

In literature, there are two main approaches to initializing a sparse orthogonal matrix: Exact Orthogonal Initialization (EOI) [70] and Sparsity Aware Orthogonal Initialization (SAO) [71].

The EOI method is the truest form of random orthogonal initialization, using Givens rotations [72] on an initial identity matrix, where each iteration reduces the sparsity. However, a limitation of this method is that sparsity is hard to control, as the amount of rotations necessary to reach a sparsity s depends on the randomly selected indices (i, j) which are rotated, and if these rotations overlap. Besides this, it is also difficult to enforce locality using this method, as the rotations are randomly selected and may be long-ranging. We will therefore not be using this method, although this could be an interesting path to go down if these issues can be overcome in some clever way.

The SAO method uses generates dense blocks along the diagonal of the matrix, and afterwards permutes the rows and columns. It is easy to see that permutations are orthogonal matrices, as they are simply matrices with unit column vectors, permuted in position. Furthermore, a block-diagonal matrix whose diagonal blocks are each orthogonal will also be orthogonal. This is a result of the block structure, as the columns belonging to distinct blocks are are parts of disjoint index sets, and therefore orthogonal to one another, while columns within the same block are orthonormal by assumption. Composition of orthogonal matrices is also orthogonal, and thus we are allowed to let these permutations act on this initial block-diagonal matrix to form a random, sparse orthogonal matrix.

The steps of our matrix initialization scheme will be:

1. Create random and dense blocks of size $k \times k$ along the diagonal of the matrix.
2. For each of these blocks, make them orthogonal using QR decomposition.
3. Permute the rows and columns of this matrix in some locality-preserving way.

1) Formally, we let our orthogonal matrix have dimension $n \times n$, and we will that we have $k \times k$ sized blocks. Then, we can fit $\lfloor \frac{n}{k} \rfloor$ of these squares, leaving one square of size $n - k\lfloor \frac{n}{k} \rfloor$. Then, the final resulting sparsity becomes

$$s = 1 - \frac{\lfloor \frac{n}{k} \rfloor k^2 + \left(n - k\lfloor \frac{n}{k} \rfloor\right)^2}{n^2}$$

This is not nicely invertible, though we can assume k divides n to find the much more reasonable relation between k, n and s, which is of the form

$$k = n(1-s) \tag{7.4}$$

For our simulations, we will thus constrain our sparsity so that k divides n, although we could also accept the more complicated solution in case we do not mind having a slight error in our sparsity. For a specific configuration, we will then get a matrix that looks like this:

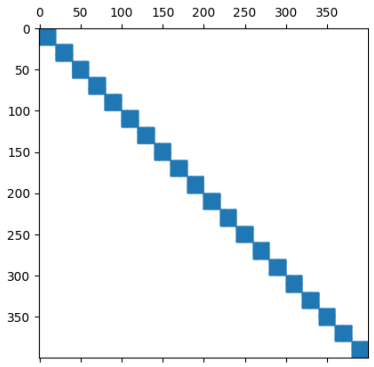

Figure 12: *The block-diagonal matrix for* $n = 400$ *and* $s = 0.95$.

2) The process of QR decomposition to generate random orthogonal matrices is already well-known [73], resulting in a matrix R and orthogonal matrix Q. One important thing to note is that the resulting orthogonal matrix Q will not be unique, as there exists a sign ambiguity. The fix to this would be to normalize the matrix by the sign of the diagonal of R, as discussed by [73]. Then, this matrix is, as we say, Haar-distributed, i.e. uniformly distributed over the group of all Orthogonal matrices of size n. If we then randomly permute our initial block-diagonal matrix, we find Fig. 13.

3) Lastly, looking at localization, we extend our permutations with a weighted probability function, so that not all indices are chosen with equal probability for permuting. Specifically, we let our probability of moving index $i \to j$ be decided by a decay function

$$p(i \to j) \propto e^{\frac{\min(|i-j|, n-|i-j|)}{\kappa}}$$

Where κ is the decay factor and we take the minimum to let boundary conditions wrap so that the matrix behaves nicely at the boundaries. With this, the resulting matrix becomes

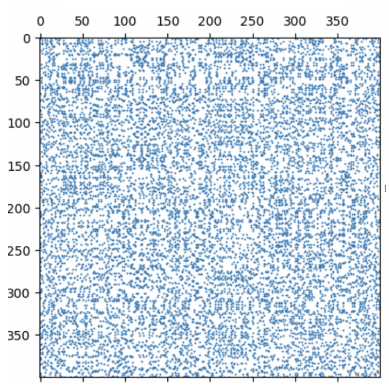

Figure 13: *The orthogonal matrix after random permutations for* $n = 400$ *and* $s = 0.95$

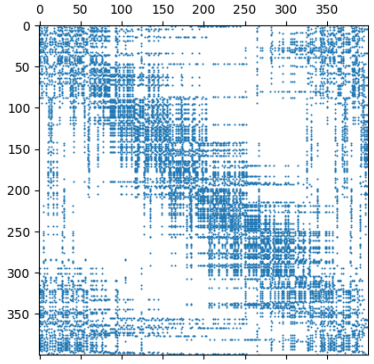

Figure 14: *The final orthogonal matrix after weighted permutations for* $n = 400$ *and* $s = 0.95$

Having done this, we have managed to create a random, sparse, local, orthogonal matrix.

7.3 Methods

Finally, let us discuss the methods of measurement and what we will be measuring in our simulations.

For our measurements, we will be comparing the original ESN, the tES^{2}N from [63], and our pES^{2}N as per Eq. (6.3). Each of these will be given the same (physical) activation function g_∞ as per Fig. 2, and sparsity, reservoir size and training time will also be kept constant between them.

As discussed in Section 3, our main measure for effectiveness will be the Valid Prediction Time as per Eq. (3.8). We will be comparing the VPT on the Mackey Glass series (3.7) for $\tau = 17$. We will generate 100 different networks of each type, and then compare the average VPT and its variance. A higher VPT is generally better, and a lower variance implies that the system is less dependent on the initialization.

The next measure we wish to compute is the Jacobian eigenspectrum per Eq. (5.4). The main complication comes in the form of our activation function, whose derivative is non-trivial. Therefore, we use the method of central differences to take the numerical derivative of this function.

We also wish to look at the power consumption. For this, we will be using equation (7.2). This can be done for each iteration, to give us a graph of the total power consumption. Here, we will also be showing the normalized power usage, so that we can compare the different graphs in terms of shape (e.g. are the peaks positioned at the same points, does the power usage follow the input sequence?). Here, it is important to note that we will only be computing the power usage of the reservoir. Any external circuitry is thus ignored for the purpose of this comparison. This will later be mentioned in the discussion part of this thesis.

Regarding tests beyond the Mackey-Glass series, there are naturally quite a lot of tests developed in the field of Reservoir computing [74]. Using such standard tests will make it easier to compare our extension of the ESN to its original and to compare our results with literature.

One specific Prediction task which we would like to mention is the Multiple Superimposed Oscillators task, or MSO for short. The input sequence for this prediction task takes the form of superimposed sine-waves, all normalized to a value between -1 and 1. In literature, an MSO with 8 frequencies (MSO8) is often used as a test that is intentionally 'impossible' to perform well at for the standard ESN [74, 75]. Let us define this input more formally

Definition 11. *(MSO8 timeseries) Let discrete time be given by* $t \in \mathbb{N}$*, and let* $w = [0.2, 0.311, 0.42, 0.51, 0.63, 0.74, 0.85, 0.97]$*. Then, we define the MSO8 timeseries at time* t *as*

$$x(t) = \sum_i \sin(w_i t). \tag{7.5}$$

We will be using the MSO8 benchmark as an additional test on the ES^{2}N, so that we can then observe if these types of multi-frequency input sequences that are difficult for standard ESNs can be learned by the new ES^{2}N model.

Finally, we note that other tests exist that go beyond the scope of this thesis. For future research specifically, we would be interested to see how the physical ES^{2}N performs at predicting the Hénon map [76], computing the systems Memory Capacity [77], imitating the Van der Pol oscillator [78] and predicting the Larma 2 input sequence.

Having gotten an overview of some of the main benchmarks and methods we will be employing, let us now get into the results obtained by our simulations.

8. Results

Let us now look at the results of our experiments. For this section, we will be comparing the physical ESN with the tES^{2}N and pES^{2}N. For all models, the hyperparameters used are displayed in Appendix B. In case a certain hyperparameter is specifically important for a given observation, we will note on this.

8.1 Jacobian Eigenspectrum

The Jacobian Eigenspectrum for each model is computed by running a model for 4000 timesteps, and then computing 4 times throughout its run, at 1000 step intervals. This way, we can be determine if the Jacobian changes through time. These eigenvalues are then computed for different values of α (for ESN) or β (for ES^{2}N).

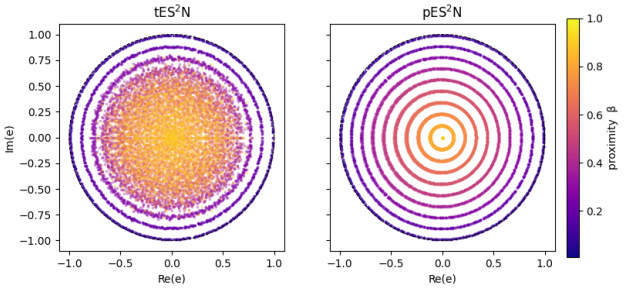

Figure 15: *The Jacobian Eigenspectrum of the tES^{2}N and pES^{2}N for different values of β (proximity).*

When putting Fig. 15 into perspective with Fig. 8, we can indeed see that the original ESN has eigenvalues all clustered near the center of the unit circle, whereas the new ES^{2}Ns display the annular eigenvalue regions as predicted by in Section 5.2. From this, we can also see that we can indeed assume that as $\beta \to 0$ the eigenvalues e tend to move towards $|e| = 1$.

8.2 Valid Prediction Time

Next, we compute the Valid Prediction Time on the Mackey-Glass series, with a total training time of 3000 timesteps for the output matrix. The models are allowed to run for 4000 timesteps, and for the first 2000 timesteps, the input of the mackey glass series is used to drive the dynamics of the internal reservoir. After those 2000 timesteps, the model feeds its own feedback into itself. From this setup, we simulate 100 randomly configured networks for each of the three models, and compute the average and relative standard error of the VPT (Eq. (3.8)). From this, we obtain Table 1.

It is interesting to see how the VPT evolves as we change the leaking rate or proximity of the system. Specifically for the ES^{2}N, we would expect that there is a trade-off between edge of chaos behavior and stability of the system, and thus we might be able to observe some optimality for a certain range of proximities. We plot these data along with the standard error of the mean at each point in Fig. 16.

Table 1: Valid Prediction Time (VPT) and Standard Error of the Mean (SEM) across reservoir models after 100 runs. The pES^{2}N displays a 57.0% increase in average VPT compared to the ESN.

Model	Mean	SEM
ESN	456.98	2.3%
tES^{2}N	637.51	2.2%
pES^{2}N	717.25	2.3%

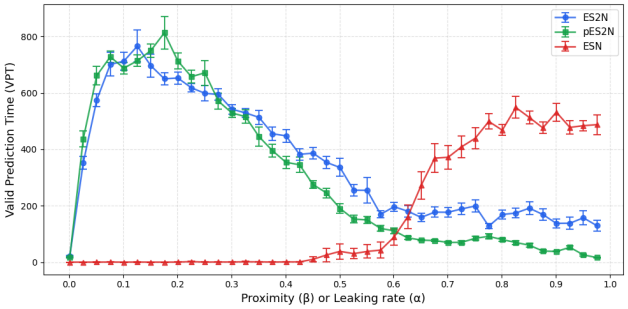

Figure 16: *The Valid Prediction Time (VPT) for three different models as a function of either their Proximity β or Leaking Rate α. Each datum represents the mean of 20 runs.*

Next, it is interesting to see how the models scale for the number of nodes in the internal reservoir. Both good performance for smaller reservoirs and scaling performance with increasing reservoir size are desirable properties, so we wish to see if the systems display these. We graph these results in Fig. 17.

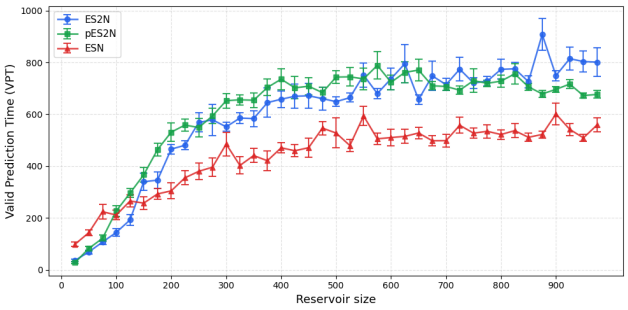

Figure 17: *The Valid Prediction Time (VPT) for three different models as a function of their reservoir size. Each datum represents the mean of 20 runs.*

Similarly, we wish to observe how the sparsity effects the models, to see at what values the system performs best. This directly relates to the number of memristors needed to build the circuit. These results are displayed in Fig. 18.

Finally, to visualise an actual prediction, let us compare how well the predictions of the free-running models match the Mackey-Glass series by overlaying the predicted and actual graph in Fig. 19.

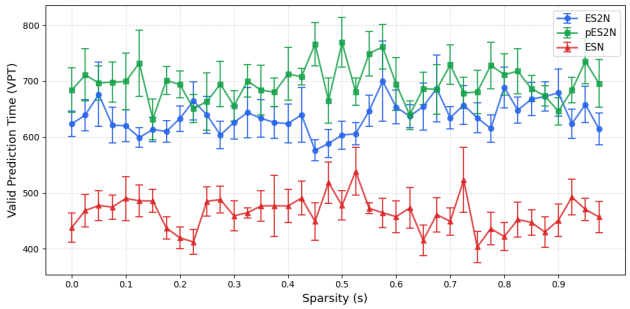

Figure 18: *The Valid Prediction Time (VPT) for three different models as a function of their sparsity s. Each datum represents the mean of 20 runs.*

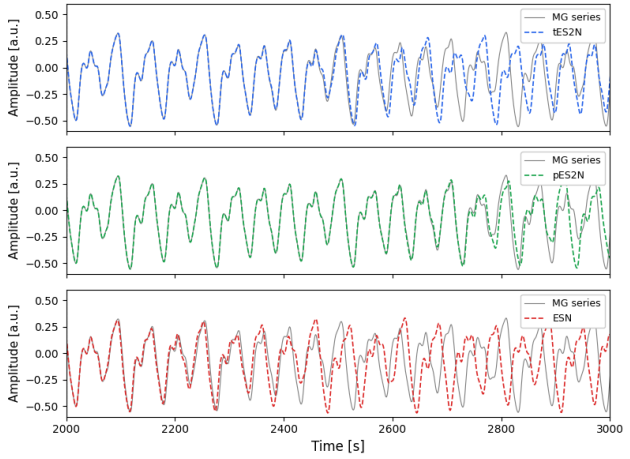

Figure 19: *The predictions for the Mackey-Glass series, starting from $t = 2000$ (the moment the system becomes free-running).*

8.3 Power Consumption

Next, we look at the power consumption of the physical networks on the Mackey-Glass series, from the moment the network starts running freely on its own output. For this computation, we convert back the dimensionless conductance to its original quantity, using a steady state conductance value of 1pS per channel as in Ref. [37]. In doing so, the graph of Fig. 20 was achieved:

To better illustrate whether these graphs align, we also plot the normalized values (to a range between 0 and 1) in Fig. 21. From this, we can observe whether specifically the peaking behavior as displayed in Fig. 20 is inherent to the ES^2N or if it also exists in the ESN.
Now that we can compute the mean power usage for different systems, we are able to turn this value into an actual result by taking the average across 100 systems. This, we can compare with the mean VPT of the system to find the a measure of efficiency, in VPT seconds per pW.
It is interesting to see how different hyperparameters influence this efficiency metric, as optimizing the systems for this efficiency would be most desirable in achieving the goal of low-power physical neural networks. In doing so, we will again be altering the proximity (or leaking rate), reservoir size and sparsity of the system, to see how the efficiency

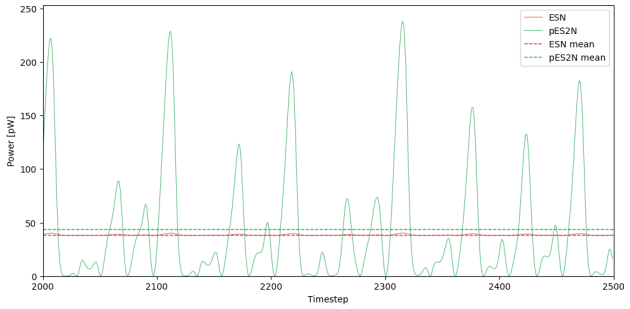

Figure 20: *The power usage of the physical networks starting from the first autonomous timestep. The ES^2N displays a much higher variance, and marginally higher mean value.*

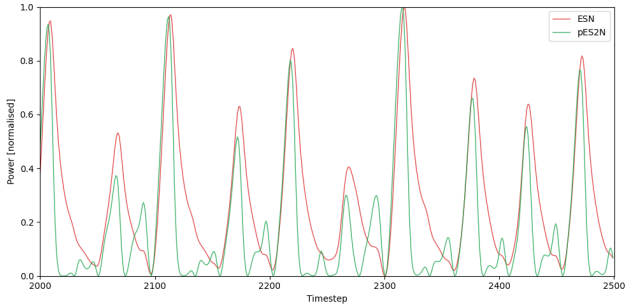

Figure 21: *The normalized power usage of the physical networks starting from the first autonomous timestep. Here, the peaks seem to be occurring at the same timesteps for both networks.*

Table 2: Power usage and efficiency across reservoir models after 100 runs.

Model	Power (pW)	Efficiency (VPT/pW)
ESN	38.37	11.91
pES^2N	44.32	16.18
% diff	+15.5%	+35.9%

maps to these parameters.

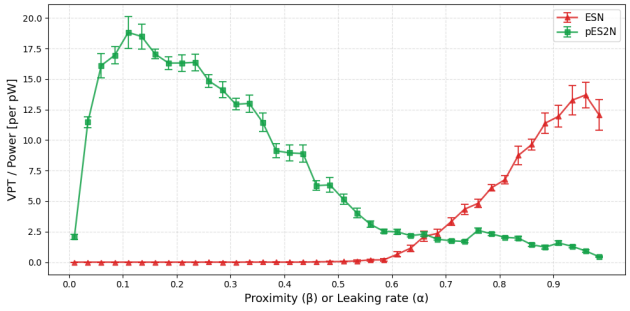

Figure 22: *The efficiency of the network as a function of either their Proximity β or Leaking Rate α. Each datum represents the mean of 20 runs.*

Next, the reservoir size.

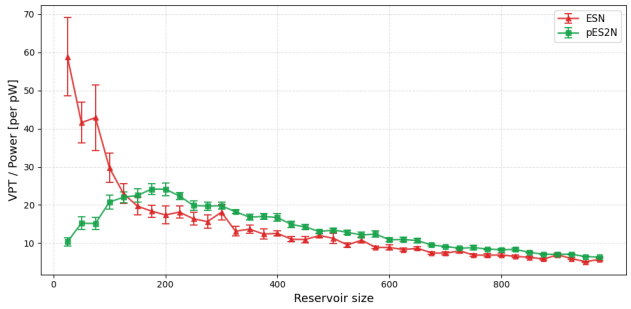

Figure 23: *The efficiency for three different models as a function of their reservoir size N. Each datum represents the mean of 20 runs.*

And finally, the sparsity of the reservoir (and orthogonal) matrix.

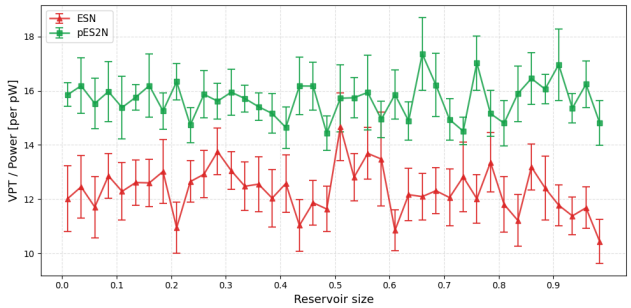

Figure 24: *The efficiency for three different models as a function of their sparsity s. Each datum represents the mean of 20 runs.*

8.4 MSO8 predictions

As mentioned in section 7, we would also like to see how our models perform at the MSO8 prediction benchmark for 18,000 timesteps. This will tell us much about the flexibility of the network, and how well it can deal with sequences having multiple internal timescales (frequencies). For this test, we will still be working with the parameters as per Appendix B.

Firstly, let us take a look at the immediate results of the VPT of the models on the MSO8 series. We find the results of Table 3. Here, we can see that the ES^{2}N models strongly outperform the standard ESN.

Table 3: Valid Prediction Time (VPT) and Standard error of the Mean (SEM) across reservoir models for the MSO8 benchmark for 100 runs. Note here that a VPT of 18,000 is the maximal value.

Model	Mean	SEM
ESN	14.77	1.9%
tES^{2}N	17,790.90	0.4%
pES^{2}N	4,222.24	2.4%

Unexpectedly, the power usage of pES^{2}N circuit on the MSO8 input sequence is actually lower than that of the standard ESN, as can be seen in Fig. 25.

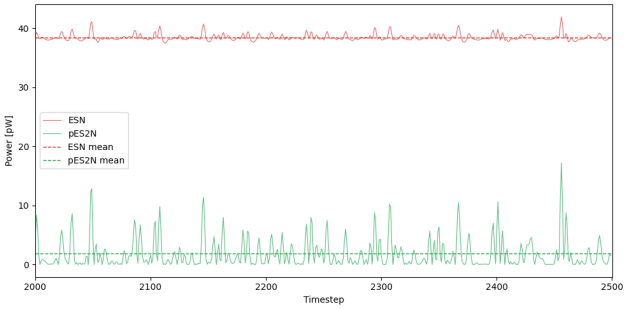

Figure 25: *The power usage of the ESN and pES^{2}N on the MSO8 input sequence.*

The efficiency of the pES^{2}N circuit will then be much better than the ESN for this specific benchmark, as we can see in Table 4. Remember that this will not be a fair comparison, as this is a test specifically designed for the ESN to fail at. Still, it would be illuminating to see if the ES^{2}N is able to succeed in terms of power as well.

Table 4: Valid Prediction Time (VPT) and Standard error of the Mean (SEM) across reservoir models for the MSO8 benchmark for 100 runs. Note here that a VPT of 2000 is the maximal value. Each of the 100 runs for the pES^{2}N reaches this value.

Model	Power (pW)	Efficiency (VPT/pW)
ESN	38.08	0.40
pES^{2}N	1.88	9,407.02
% diff	-95.1%	+2,351,655.0%

Lastly, it would be interesting to see how the performance of these models scales with the number of nodes in the network. We finally display these results in Fig. 26.

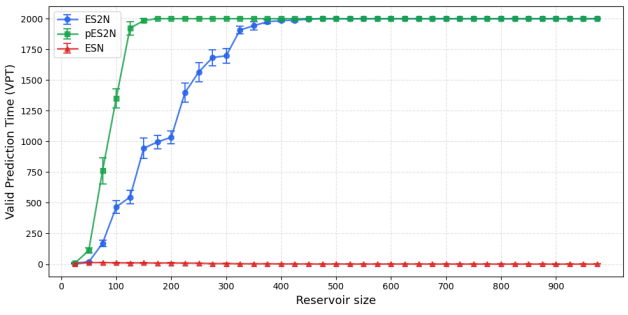

Figure 26: *The performance (VPT) as a function of the reservoir size (N). Here, each modem represents the average of 20 runs. Note that 2000 is the maximum VPT that is being captured with these measurements.*

Having gathered all of these results, we are now ready to discuss their implications and how we could improve these measurements with an eye on future research.

9. Discussion

Now that we have obtained our results, we would like to interpret them in this discussion section. Besides this, we will also be discussing the limitations of our results and methods, and how these measurements can be improved. Finally, we will be going over the next steps that can be taken for research in this field.

In section 8, we introduced the results without much consideration as to how they should be interpreted. These results are meant to showcase the increase in performance (VPT) and Efficiency (VPT/Watt) of the (physical) ES^2N compared to the older ESN model, while also showing that our model behaves as we would expect it to. In doing so, we hope to answer our primary and secondary research question as posed in section 1.

9.1 tES^2N and pES^2N

First, we would like to discuss the differences between the theoretical and physical model of the ES^2N. To do this, we first observe Fig. 15. Here, we clearly see that there is an annular region to which the eigenvalues are confined for each value of β. This result experimentally verifies theorem 4, which stated that the eigenvalues would be confined to some annular region of width $\beta\gamma\sigma$. This result matches most closely to the tES^2N, where we can see the annulus tightening for decreasing values of β. For the pES^2N, however, there seems to be no such tightening behavior, and all annuli already show a small width. To explain this behavior, we know that all hyperparameters are identical for these systems. Because of this, we expect coupled nature of the Orthogonal matrix O and the reservoir matrix W^r to be the cause of this phenomenon.

Besides this, we can see that our third property of a tunable Lyapunov exponent also seems to hold, as the eigenvalue spectrum can be tuned to the Edge of Chaos for arbitrarily low values of β. Do note, however, that both Fig. 16 and Fig. 22 show that as β tends to 0, the VPT and efficiency tend to fall off rapidly. This can be explained by the fact that the simulations become unstable for these regimes. Future research would have to be done to determine whether the instability of the Echo State Network here is an artefact of the discretization, or an actual property of the physical circuit.

Lastly, it is interesting to note that the performance of the pES^2N tends to be similar to or even better than that of the tES^2N, while you would expect that the constraints placed on the physical circuit would also constrain the performance possible. A possible way to explain this is by the fact that the orthogonal matrix of the physical network is more carefully generated to match well with the reservoir matrix when compared to the theoretical network. Considering this, together with the fact that the mathematical properties still hold, we would have no real reason to suspect the circuit to underperform, and the more optimal generation of orthogonal matrices could actually cause the physical network to outperform the theoretical model in expectation.

9.2 Valid Prediction Time - Analysis

Now, we will discuss the results of the Valid Prediction Time measurements taken in section 8.2. Having compared the 3 different models for the hyperparameters as in Appendix B, we can see that the mean VPT for 100 runs is greatest for the pES^2N, showing a 57.0% increase compared to the ESN. With a Standard Error of the Mean (SEM) of 2.3%, this is a significant result, and we can indeed conclude that our new physical ES^2N outperforms the old ESN. Note that for these simulations, all the same hyperparameter values have been used as in the Pyontronics python codebase [19, 50].

When comparing the VPT for different values of the parameters α (for ESN) and β (for ES^2Ns), we can see that the models at the optimal range of β for the ES^2Ns outperform the ESN at its optimal range for α. Interesting to note once again is the rapid fall-off as β goes to zero, which does not occur for the ESN at the opposite end as α tends to 1. The fact that this issue happens only at the $\beta \approx 0$ regime of values suggests once again that this is caused by an instability at the Edge of Chaos.

Another interesting observation is that the ESN network drops to a near-zero VPT quite fast ($\alpha \approx 0.4$), compared to the ES^2N models, which retain some VPT even at their worst regimes. As we do not generally want to operate at these β-values, and hardware will be fixed to work at the optimal values of β, we will not go much further into trying to explain this property.

We have also graphed the VPT for increasing reservoir size in Fig. 17. It is interesting to note that the ESN outperforms the ES^2N models for the lowest number of nodes, where the ES^2N models seem to overtake the performance around a node number of 100. From there, there seems to be a diminishing return on increasing the number of nodes for each of the three models. An interesting follow-up question on these results is whether this flattening behavior also occurs for different input sequences, and to what degree.

Finally, we can come to two interesting conclusions from Fig. 18. Firstly, we can see that there is no meaningful correlation between sparsity and Valid Prediction Time for all three of the models. This suggests that the sparsity does not matter for the performance of the system. As we would like to have our physical circuit to be sparse (so that we don't use too many memristors), this result validates that this is a choice that does not sacrifice on performance. Secondly, the orthogonal matrix for the pES^2N and tES^2N will both be dense for a sparsity of 0. In our graph, we can still see the physical ES^2N outperform its theoretical counterpart, from which we can assume that the difference in performance does not stem from the sparsity of the orthogonal matrix. This in turn lets us conclude that it must be either the additional term that we introduce into the equation of motion, or the method of generating our random matrix that effects our performance thus.

9.3 Power Consumption - Analysis

For this part, we discuss the results of section 8.3, going over measurements of Power Consumption and efficiency. We would expect the Power usage to increase for the pES^{2}N, as this model introduces a new memristor component at each node. The exact magnitude of this increase is difficult to predict, however, as the power usage of this linear memristor will not be equal to that of the reference resistor and non-linear memristor. Besides this, there is a change of weight between the linear and non-linear contributions of the conductance in the dynamic equation in Eq. (5.1). We hope to see the effect these changes have on the power usage, and to then conclude whether the efficiency of the system (VPT per Watt) is actually increased by our new pES^{2}N model.

Starting with Table 2, we have conducted simulations with 100 randomly configured networks having hyperparameters as in Appendix B. From these simulations, we can observe a 15.5% power increase for the ES^{2}N, which means that the efficiency is increased by a net 35.9%. These results suggest that the gain in Valid Prediction Time for the ES^{2}N is relatively larger than the gain in power consumption. For this result, we will note that the peripheral circuit is not taken into account for these calculations. Though the peripheral circuit has not been defined, it has not been concretely adjusted for the ES^{2}N. If we assume the power usage to be roughly equal between these two, then the found efficiency acts as a lower bound. Calling the power of a network P and of the peripheral circuit P_p, we can note that the increase in efficiency is equal to

$$\frac{\mathrm{VPT}_{\mathrm{ES^2N}}}{\mathrm{VPT}_{\mathrm{ESN}}} \frac{P_{\mathrm{ESN}} + P_p}{P_{\mathrm{ES^2N}} + P_p}.$$

This value tends to $\frac{\mathrm{VPT}_{\mathrm{ES^2N}}}{\mathrm{VPT}_{\mathrm{ESN}}}$ for $P_p \to \infty$, which means that the upper bound of the efficiency increase is simply the performance increase 59.0%. To put it into other words, we argue that the addition of our linear memristor into the nodes of the ESN circuit as per Fig. 6 is at least 35.9% and at most 59.0%.

One problem with the power usage of the ES^{2}N can be found in Fig. 20; there seems to be a large variance in the power usage of the ES^{2}N circuit. A high variance of power on the network can put strain on the circuit, and is generally not a desirable property of the power profile [79]. When looking at the normalized power usage in Fig. 21, we can see that the peaking behavior of this spiking matches closely with that of the ESN, but is simply amplified greatly. This could be explained by the linear term weighing more and including a term linear in the input sequence, thus amplifying the signal that is being put into the sequence and peaking more heavily when this input sequence peaks. Though this would explain the behavior, it is still one of the larger downsides to the ES^{2}N model.

Figures 22 and 24 mimic the behavior of figures 16 and 18, respectively. There is not much new to note here, besides the fact that sparsity does not effect efficiency much. This can be explained by the fact that power usage in the circuit lives in the node, not the edges, of the network. There is an inherent assumption of the edges between nodes being resistance-free, and thus without power consumption. If we where to make this model more accurate, we would predict that efficiency would become better for higher sparsity.

Lastly, Fig. 23 shows us the number of nodes for which the ESN outperforms the ES^{2}N. Specifically, for simple tasks that do not require long-term memory (i.e. a smaller number of nodes), the ESN will perform at the best efficiency out of both models. As the reservoir size increases, the efficiency of both models seem to fall off and converge to near-identical values.

9.4 MSO8 - Analysis

For the final set of results, we turn to the MSO8 and how the different models perform on this benchmark. Remember from section 7 that the MSO8 is specifically chosen as a task the ESN fails at. The reason why the ESN does not perform well at this task is because the MSO8 input sequence operates at 8 different decoupled frequencies. The linear term in Eq. (3.3) decays with a factor $1 - \alpha$ for all eigenvectors, and because of this, the system is unable to learn all 8 frequencies. This is a well-known problem, and has been resolved in other ways, like by having each memristor in a circuit have a unique internal timescale τ (which we call a Bandpass Network) [19].

It is quite an interesting result that the ES^{2}N models both do excel at the MSO8 benchmark (following from Table 3). This would imply that these systems are able to learn different timescales while still having each memristor have identical timescale τ. Because of this, one singular fixed model would be able to tackle problems concerining a myriad of internal timescales, leading to a more flexible design. The reason such ES^{2}N models perform well at such tasks compared to the ESN is because the linear term of the ES^{2}N as per Eq. (5.1) now has an orthogonal matrix, meaning that different eigenvalues now decay at different rates. Because of this, the network will decouple different timescales through time, meaning that different parts of the system are able to remember different timescales of the input sequence.

Another surprising result is the power consumption of the network displayed in Fig. 25. While we still have a large variance similar to with the Mackey-Glass series (Fig. 20), the mean power consumption seems to bit much lower. This may be explained by the fact that the input sequence is less chaotic, meaning that the ES^{2}N is able to learn the profile of the input much easier, requiring less input scaling and thus power consumption. The high variance of the power would still place some strain on the network, but for this specific benchmark, this difference between ESN and ES^{2}N in strain will be less severe. As both performance and power consumption are improved massively for this task, the efficiency increase turns out to be a staggering 2,109,700%. Remember that this improvement is on a test ESNs are not designed for, but this still illustrates the increased flexibility of the (physical) ES^{2}N model.

Finally, it is interesting to see that in Figure 26 and Table 3, the pES^{2}N seems to outperform the tES^{2}N heavily. The physical model seems to scale much faster for an

increasing number of nodes, and the performance tends to be significantly higher. This is a surprising result, as this behavior does not emerge for the Mackey-Glass input sequence. There are two possible places to which we can attribute this behavior. Firstly, it might be because of the extra term linear in the input that the physical model has. This term would more directly input the MSO8 sequence into the circuit, which could cause the system to learn such simple inputs more easily. A second, more likely, hypothesis lies in the constraints we place on our Orthogonal matrix. When we enforce locality of the orthogonal matrix (as is done in the physical circuit), we design the circuit to have its orthogonal matrix more localized for different clusters of nodes. This behavior would explain why the system is able to learn sequences with multiple input frequencies like the MSO8 more easily, as local clusters each predicting a component of the frequencies will emerge more easily for the constrained matrix than for the dense matrix.

We conclude from the MSO8 analysis that the ES^{2}N is much more flexible to different tasks than the standard ESN. Though more research would have to be done on the difference between theoretical and physical ES^{2}N, we conservatively conclude from these results that the constraints placed on the physical design actually extend its better when compared to the theoretical model.

9.5 Limitations to this research

Having created a method and produced results, there is still much that could be improved. Let us discuss some of the limitations of our model and methods, to get a better view of our assumptions, simplifications and potential errors.

Firstly, we note that the Simple Volatile Memristor as per definition 1 is a simplification that is crucial to turning our Echo State Networks physical. The SVM model (and our simulation) hinges on the first-order approximation of the Tayler expansion of f around zero. For large variance and rapidly changing voltage inputs, the first-order approximation may break, which would in turn lead to the breaking of the equivalence between the physical circuit and ESN or ES^{2}N. This might pose a serious problem for actual physical applications, as we have seen in the variance of power in Fig. 20. The problem can be circumvented, however, by reducing the input-scaling ω of the network. Besides this, the peripheral circuit may be able to prevent the network from operating at the regime beyond first-order approximation, further aiding in keeping our correspondence in tact. We suggest that care be taken when implementing the physical ES^{2}N, especially when operating with high-variance and high-frequency input sequences. Once such circuits become physically realizable, we encourage further research into this non-SVM regime to test for which circuits and input sequences the SVM assumption holds.

Secondly, we have assumed the peripheral circuit to be given. Though the tasks we require this peripheral circuit to perform are all quite realizable, further research into the exact topology and power consumption of this circuit would still be needed to completely validate the (physical) ES^{2}N model. For now, the efficiency increase of the pES^{2}N on the Mackey-Glass benchmark can simply be taken as a lower bound, though further research would both refine this claim and make the implementation of such a physical ES^{2}N more realizable.

Thirdly, discretization within our simulations is also an assumption that would work differently once an actual physical circuit will be built. To ensure physical timescales for the simulated memristors, the timesteps in the simulations have been carefully chosen, and this may not be reflected as easily within real-world circuits. The good news is that a correspondence between discrete equations of motion implies a correspondence between continuous equations of motion, as we have derived the discrete equations from their continuous dynamics. Then, the only complication that needs to be studied further is how to translate the hyperparameters used in our simulations to the continuous regime. Specifically for the timestepping, this will require careful engineering of the memristor timescales and circuit clocks.

Besides these points, the noticeable differences between the physical and theoretical ES^{2}N have not been explained to satisfaction yet. Understanding why these produce such different results at times will go a long way towards better understanding the dynamics of the new pES^{2}N, and currently forms a gap of knowledge within the model used.

In terms of methods, the VPT measure is not the only way we can quantify performance of the network on reservoir computing tasks. We have only chosen to perform one test, as it most accurately gives a notion of memory in the system, but there are many other approaches that could be taken. Specifically, the Root Mean Squared Error (RMSE) is a measure often used in the field of reservoir computing [19], which makes comparison to other memristor based networks easier. We have specifically opted not to use this measure, however, as we feel that the RMSE can give a distorted image of the actual effectiveness of a model by potentially rewarding random predictions over phase-shifted predictions. One of our main goals is to assess whether the model is able to learn the (chaotic) dynamics of the input sequence, and as such, a better benchmark that future work could employ is seeing how the phase-space of the prediction and actual sequence differ for example. Using such measures on more different input-sequences will give an even better understanding of the differences between the ESN and pES^{2}N than this thesis has already sketched.

Finally, we would like to note that generation algorithm of the physical Orthogonal matrix is still quite arbitrary and unstudied. The different components mathematically guarantee orthogonality and sparsity, but besides this no real research has been done into how the generation could be improved. Testing out different algorithms for the generation was not within the scope of this thesis, but this has lead to some results regarding the differences between theoretical and physical ES^{2}Ns becoming more difficult to interpret. We therefore remark that the simulations and results could have been improved by means of a preliminary study on different generation methods.

9.6 Future research

For future research, we would of course be interested in seeing these results refined even further. Though the benchmarks and tests conducted in this thesis give a convincing argument for the superiority of the Edge of Chaos regime introduced by the ES^{2}N over the standard ESN circuit, there is still much left unknown. Specifically, the difference between theoretical and physical ES^{2}N can still be expanded upon further, and tests like the Hénon map, Memory Capacity and Larma 2 input sequence (as discussed in section 7.3) would further give insight into the potential of this new model.

Though the internal dynamics, performance and efficiency of the ES^{2}N have been modelled and tested well in this thesis, there is still much unknown about the peripheral circuit necessary to achieve this. Another natural continuation of this research would then be to completely develop the theory around this peripheral circuit and include it into the benchmarks used in this thesis. In doing so, a more complete picture of performance increase and especially efficiency would be sketched. This would also make future implementations of the circuit easier to realize.

As the peripheral circuit is a large unknown that can influence the efficiency of the network greatly, an important direction for this circuitry could be to simplify the role the peripheral circuit plays in its dynamics. If the pES^{2}N can be simplified to still display Edge of Chaos behavior while having a simpler peripheral circuit, this would result in a large gain in efficiency.

Extending on this further, Echo State Networks are still rather difficult to study experimentally due to the use of conical memristors and complex peripheral circuits. Then, studying a simpler model that may have less performance than the ESN but is easier to build would be a good next direction to make the jump between this simple model to ESN to ES^{2}N easier to realize experimentally.

If the physical ES^{2}N where to actually be implemented using conical memristors, another step that would prove useful would be to further research the generation of the random Orthogonal matrix using simulations. The implementation in this thesis serves its purpose well enough, but improvements may still be found in this method, and physical circuits may benefit greatly from some additional research.

In the same vain, studying the limits for where the model holds — both in terms of realizable hyperparameters as well as in terms of the domains for which our assumptions of Simple Volatile Memristors hold — would be valuable research that can further explore the scope of problems this type of physical ES^{2}N is able to solve.

From this, a direction we find important to further deeper understanding of the proximity parameter β is to study the instability that occurs as $\beta \to 0$. We touched on this during this section already, but we have not been able to give a satisfying explanation to this phenomenon yet. Studying this will make it clearer what the origin of instability is, and whether we would be able to find a way to push the circuit even further to the Edge of Chaos then we do now.

Once the model will be made physical, the voltage and current may be subject to noise due (for example) to thermal noise or electromagnetic interference and resistance within the edges of the network. The simulations have not taken these into account, but for testing effectiveness in actual physical implementations, these would be good variables to also include into the model. ESN models have already been studied with noise [80, 81], so we believe this is a natural direction for future research.

Finally, it would be of interest for future research to apply the model of the physical ES^{2}N to physical input data to research future applications of these new models in actual real-world tasks. Candidate tasks would be the respitory data used in [19], ECG and EEG data as in [20, 27, 81], and speech patterns as in [32].

10. Conclusion

We have reviewed literature and theory on Echo State Networks based on Conical Memristors and found that these can be improved by the introduction of Edge of Chaos behavior in the form of the Edge of Stability Echo State Network. We have managed to create a circuit to turn these ES^{2}Ns physical using Conical Memristors, and have proven that this circuit does indeed correspond to the theoretical ES^{2}N. By doing simulations on the Mackey-Glass input series, we have found that performance increases by 59.0%, and the efficiency increase of this new model lies between $35.9\% - 59.0\%$.

We have experimentally verified using the Jacobian Eigenspectrum that the new ES^{2}N obeys the annular eigenvalue property as per Theorem 4 and that the Edge of Chaotic of the physical circuit is tunable. Furthermore, we have established that our new pES^{2}N model performs much better than the ESN at tasks involving multiple frequencies by means of analysing the MSO8 input sequence, a sequence the standard ESN famously fails at.

In conclusion, the proposed physical ES^{2}N is a direct improvement on the physical ESN that went before. Edge of Chaos dynamics can be introduced into the memristor-based Echo State Network by extending it to the Edge of Stability Echo State Network, and this improves both predictive timescales and efficiency in the circuit on standard benchmarks like the Mackey-Glass and MSO8 series. Notably, the Edge of Chaos is introduced in a tunable way, and the internal dynamics of the system allow it to learn input sequences that feature multiple timescales, supported by favorable results on the MSO8 series.

Appendix

A. Activation function

When generating the nonlinear activation function as per Eq. (2.5), we briefly introduced the variables that make up this equation. The values used in this equation are crucial to tuning the behavior of the steady state conductance as a nonlinear activation function, and therefore it is necessary to denote the values used for these variables. These are displayed in Table 5 below.

Table 5: Values used in computing the steady state conductance as per Eq. (2.5)

Variable	Symbol	Value
Base radius	R_b	$100 * 10^{-6}$
Tip radius	R_t	$5 * 10^{-6}$
Channel length	L	$150 * 10^{-6}$
Permittivity	ϵ	$0.7104 * 10^{-9}$
Diffusion coefficient	D	10^{-9}
Surface potential	ψ_0	-0.039
Boltzman constant	k_b	$1.3806 * 10^{-23}$
Temperature	T	293.15
Viscocity	η	0.0010093
Electric charge	e	$1.602 * 10^{-19}$

The final activation function is produced by filling in these values into Eq. (2.5) and then numerically integrating the integral of that equation. To make simulation faster, the results are stored in a lookup table with a certain resolution. When computing this for a voltage V, we then take the nearest voltage for which we have stored a value of the activation function and return that stored value.

B. Model hyperparameters

For the hyperparameters, the following default values have been used for the different models:

Table 6: Default values for ESN

Hyperparameter	Symbol	Value
Leaking rate	α	0.95
Steady state conductance	g_0	10^{-12}
Timescale	τ	2.27
stepsize	Δt	1.0
Spectral radius	ρ	0.95
Sparsity	s	0.95
input scaling	ω	0.45
Reservoir size	N	400

Next, for the physical ES^{2}N, we ran some preliminary tests to find the correct hyperparameters for the Mackey-Glass series. The following hyperparameters are the results:

Table 7: Default values for pES^{2}N

Hyperparameter	Symbol	Value
proximity	β	0.068
coupling	b	8.8
Steady state conductance	g_0	10^{-12}
Timescale	τ	2.27
stepsize	Δt	0.154
Spectral radius	ρ	0.95
Sparsity	s	0.95
input scaling	ω	2.5
Reservoir size	N	400

For the tES^{2}N, the same values as the pES^{2}N are used where applicable. The values for b, β and ω are the ones optimized specifically for this model. The value of Δt is chosen to produce β from the given memristor timescale of the ESN.

Table 8: Default values for tES^{2}N

Hyperparameter	Symbol	Value
proximity	β	0.068
Timescale	τ	2.27
stepsize	Δt	0.154
Spectral radius	ρ	0.95
Sparsity	s	0.95
input scaling	ω	2.5
Reservoir size	N	400

References

1. International Energy Agency. *Energy Demand from AI* tech. rep. Licence: CC BY 4.0 (International Energy Agency (IEA), Paris, 2025). https://www.iea.org/reports/energy-and-ai/energy-demand-from-ai.
2. Balasubramanian, V. Brain power. *Proceedings of the National Academy of Sciences of the United States of America* **118.** PMCID: PMC8364152, e2107022118 (Aug. 2021).
3. Yu, L. *et al.* Bioinspired nanofluidic iontronics for brain-like computing. *Nano Research* **17,** 503–514. https://www.sciopen.com/article/10.1007/s12274-023-5900-y (2024).
4. Khan, M. U. *et al.* Advancement in Soft Iontronic Resistive Memory Devices and Their Application for Neuromorphic Computing. *Advanced Intelligent Systems* **5,** 2200281. eprint: https://advanced.onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.202200281. https://advanced.onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202200281 (2023).
5. Hou, Y. *et al.* Learning from the Brain: Bioinspired Nanofluidics. *Journal of Physical Chemistry Letters* **14,** 2891–2900 (Mar. 2023).
6. Xie, B. *et al.* Perspective on Nanofluidic Memristors: From Mechanism to Application. *Chemistry – An Asian Journal* **17,** e202200682 (Nov. 2022).
7. Kim, D. & Lee, J.-S. Liquid-based memory devics for next-generation computing. *ACS Applied Electronic Materials* **5,** 664–673 (Feb. 2023).
8. Xu, G. *et al.* Ångström-Scale-Channel Iontronic Memristors for Neuromorphic Computing. *ACS Applied Materials & Interfaces* (May 2025).
9. Chua, L. O. Memristor—The Missing Circuit Element. *IEEE Transactions on Circuit Theory* **18,** 507–519 (Sept. 1971).
10. Strukov, D. B., Snider, G. S., Stewart, D. R. & Williams, R. S. The missing memristor found. *Nature* **453,** 80–83 (2008).
11. Chen, X. & Victora, R. H. Spin-torque memristor based on magnetic tunnel junctions. *IEEE Transactions on Magnetics* **47,** 3640–3643 (2011).
12. Choi, S., Bezugam, S. S., Bhattacharya, T., *et al.* Wafer-scale fabrication of memristive passive crossbar circuits for brain-scale neuromorphic computing. *Nature Communications* **16,** 8757 (2025).
13. Chen, T. *et al. Magnetic-field controlled organic spintronic memristor for neural network computation* 2025. arXiv: 2510.23542 [cond-mat.mes-hall].
14. Driscoll, T., Kim, H.-T., Chae, B.-G., Di Ventra, M. & Basov, D. N. Phase-transition driven memristive system. *Applied Physics Letters* **95,** 043503 (2009).
15. Lin, Y.-P. *et al.* Physical Realization of a Supervised Learning System Built with Organic Memristive Synapses. *Scientific Reports* **6,** 31932 (2016).
16. Takeuchi, S. *et al.* Demonstrative operation of four-terminal memristive devices fabricated on reduced TiO_2 single crystals. *Scientific Reports* **9,** 2601 (2019).
17. Abraham, I. The case for rejecting the memristor as a fundamental circuit element. *Scientific Reports* **8** (July 2018).
18. Kamsma, T., van Roij, R. & Spitoni, C. A simple mathematical theory for Simple Volatile Memristors and their spiking circuits. *Chaos, Solitons & Fractals* **186,** 115320. ISSN: 0960-0779. https://www.sciencedirect.com/science/article/pii/S0960077924008725 (2024).
19. Kamsma, T. M., Teijema, J. J., van Roij, R. & Spitoni, C. Echo state and band-pass networks with aqueous memristors: Leaky reservoir computing with a leaky substrate. *Chaos: An Interdisciplinary Journal of Nonlinear Science* **35.** ISSN: 1089-7682. http://dx.doi.org/10.1063/5.0273574 (2025).
20. Sun, L. *et al.* Unsupervised EEG feature extraction based on echo state network. *Information Sciences* **475,** 1–17. ISSN: 0020-0255. https://www.sciencedirect.com/science/article/pii/S0020025518307692 (2019).
21. Wen, G., Li, H. & Li, D. *An ensemble convolutional echo state networks for facial expression recognition* in (Sept. 2015), 873–878.
22. Mici, L., Hinaut, X. & Wermter, S. *Activity recognition with echo state networks using 3D body joints and objects category* in (May 2016).
23. Dettori, S., Matino, I., Colla, V. & Speets, R. *Deep Echo State Networks in Industrial Applications* in *Artificial Intelligence Applications and Innovations* **584.** 16th IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI 2020) (Springer, Neos Marmaras, Greece, June 2020), 53–63.
24. Colla, V., Matino, I., Dettori, S., Cateni, S. & Matino, R. in, 66–79 (May 2019). ISBN: 978-3-642-54671-6.
25. Chang, M., Terzis, A. & Bonnet, P. *Mote-Based Online Anomaly Detection Using Echo State Networks* in. **5516** (June 2009), 72–86. ISBN: 978-3-642-02084-1.
26. Obst, O., Wang, bibinitperiod & Prokopenko, M. *Using echo state networks for anomaly detection in underground coal mines* English. in *Proceedings - 2008 International Conference on Information Processing in Sensor Networks, IPSN 2008* 2008 International Conference on Information Processing in Sensor Networks, IPSN 2008 ; Conference date: 22-04-2008 Through 24-04-2008 (Institute of Electrical and Electronics Engineers (IEEE), United States, 2008), 219–229. ISBN: 9780769531571.
27. Ayyagari, S. S., Jones, R. D. & Weddell, S. J. *EEG-based event detection using optimized echo state networks with leaky integrator neurons* in *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (IEEE, 2014), 5856–5859.

28. Li, N., Tuo, J., Wang, Y. & Wang, M. Prediction of blood glucose concentration for type 1 diabetes based on echo state networks embedded with incremental learning. *Neurocomputing* **378,** 248–259. ISSN: 0925-2312. https://www.sciencedirect.com/science/article/pii/S092523121931375X (2020).

29. Lin, X., Yang, Z. & Song, Y. Short-term stock price prediction based on echo state networks. *Expert Systems with Applications* **36,** 7313–7317. ISSN: 0957-4174. https://www.sciencedirect.com/science/article/pii/S0957417408006519 (2009).

30. Tong, M. H., Bickett, A. D., Christiansen, E. M. & Cottrell, G. W. Learning grammatical structure with Echo State Networks. *Neural Networks* **20,** 424–432 (2007).

31. Song, Q. & Feng, Z. *Stable trajectory generator - Echo state network trained by particle swarm optimization* in (Dec. 2009), 21–26.

32. Shrivastava, H., Garg, A., Cao, Y., Zhang, Y. & Sainath, T. *Echo State Speech Recognition* 2021. arXiv: 2102.09114 [cs.CL]. https://arxiv.org/abs/2102.09114.

33. Bertschinger, N. & Natschläger, T. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation* **16,** 1413–1436 (July 2004).

34. Boedecker, J., Obst, O., Lizier, J. T., Mayer, N. M. & Asada, M. Information processing in echo state networks at the edge of chaos. *Theory in Biosciences* **131,** 205–213 (Sept. 2012).

35. Verzelli, P., Livi, L. & Alippi, C. A characterization of the Edge of Criticality in Binary Echo State Networks. *CoRR* **abs/1810.01742.** arXiv: 1810.01742. http://arxiv.org/abs/1810.01742 (2018).

36. Zhang, L., Feng, L., Chen, K. & Lai, C. H. Edge of chaos as a guiding principle for modern neural network training. *CoRR* **abs/2107.09437.** arXiv: 2107.09437. https://arxiv.org/abs/2107.09437 (2021).

37. Kamsma, T. M. *et al.* Energy-efficient time series processing in real-time with fluidic iontronic memristor circuits. *Faraday Discussions*. ISSN: 1364-5498. http://dx.doi.org/10.1039/D5FD00168D (2026).

38. Kamsma, T. M., Boon, W. Q., ter Rele, T., Spitoni, C. & van Roij, R. Iontronic Neuromorphic Signaling with Conical Microfluidic Memristors. *Phys. Rev. Lett.* **130,** 268401. https://link.aps.org/doi/10.1103/PhysRevLett.130.268401 (26 June 2023).

39. SCHMUCK, M. ANALYSIS OF THE NAVIER–STOKES–NERNST–PLANCK–POISSON SYSTEM. *Mathematical Models and Methods in Applied Sciences* **19,** 993–1014 (2009).

40. :contentReference[oaicite:0]index=0 & :contentReference[oaicite:1]index=1. Long Short-Term Memory. *Neural Computation* **9,** 1735–1780 (Nov. 1997).

41. Kamsma, T. M. *et al.* Brain-inspired computing with fluidic iontronic nanochannels. *Proceedings of the National Academy of Sciences* **121,** e2320242121. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.2320242121. https://www.pnas.org/doi/abs/10.1073/pnas.2320242121 (2024).

42. Schmidt, R. M. Recurrent Neural Networks (RNNs): A gentle Introduction and Overview. *CoRR* **abs/1912.05911.** arXiv: 1912.05911. http://arxiv.org/abs/1912.05911 (2019).

43. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* http://www.deeplearningbook.org (MIT Press, 2016).

44. Pascanu, R., Mikolov, T. & Bengio, Y. *On the difficulty of training recurrent neural networks* in *Proceedings of the 30th International Conference on Machine Learning* (eds Dasgupta, S. & McAllester, D.) **28** (PMLR, Atlanta, Georgia, USA, 17–19 Jun 2013), 1310–1318. https://proceedings.mlr.press/v28/pascanu13.html.

45. Hochreiter, S. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **6,** 107–116 (Apr. 1998).

46. Bengio, Y., Simard, P. & Frasconi, P. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks* **5,** 157–166 (1994).

47. Jaeger, H., Lukoševičius, M., Popovici, D. & Siewert, U. Optimization and applications of echo state networks with leaky- integrator neurons. *Neural Networks* **20.** Echo State Networks and Liquid State Machines, 335–352. ISSN: 0893-6080. https://www.sciencedirect.com/science/article/pii/S089360800700041X (2007).

48. Mayer, N. & Browne, M. *Echo State Networks and Self-Prediction* in. **3141** (Jan. 2004), 40–48. ISBN: 978-3-540-23339-8.

49. Li, C. *et al.* Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature Communications* **9,** 2385 (June 2018).

50. Kamsma, T. M. & Teijema, J. J. *Code Repository for: Pyontronics* Mar. 31, 2025. https://github.com/TMKamsma/Pyontronics.

51. Mackey, M. C. & Glass, L. Oscillation and Chaos in Physiological Control Systems. *Science* **197.** Bibcode: 1977Sci...197..287M, 287–289 (July 1977).

52. Viehweg, J., Walther, D. & Mäder, P. Temporal convolution derived multi-layered reservoir computing. *Neurocomputing* **617,** 128938. ISSN: 0925-2312. http://dx.doi.org/10.1016/j.neucom.2024.128938 (Feb. 2025).

53. Hassaan, Z., Yacoub, M. & Said, L. FPGA-Accelerated ESN with Chaos Training for Financial Time Series Prediction. *Machine Learning and Knowledge Extraction* **7,** 160 (Dec. 2025).

54. Hurley, L. A. & Shaheen, S. E. *Reservoir computing with large valid prediction time for the Lorenz system* 2025. arXiv: 2508.06730 [cs.NE]. https://arxiv.org/abs/2508.06730.

55. Jaurigue, L. Chaotic attractor reconstruction using small reservoirs—the influence of topology. *Machine Learning: Science and Technology* **5,** 035058. https://iopscience.iop.org/article/10.1088/2632-2153/ad6ee8 (Aug. 2024).

56. De Oliveira, P. M. C. *Why do Evolutionary Systems Stick to the Edge of Chaos* 2001. arXiv: cond-mat/0101170 [cond-mat]. https://arxiv.org/abs/cond-mat/0101170.

57. Feng, L., Zhang, L. & Lai, C. H. *Optimal Machine Intelligence at the Edge of Chaos* 2020. arXiv: 1909.05176 [cs.LG]. https://arxiv.org/abs/1909.05176.

58. Tao, T. *An introduction to measure theory / Terence Tao*. eng. ISBN: 9780821869192 (American Mathematical Society, Providence, R.I, 2011).

59. Oseledets, V. I. A Multiplicative Ergodic Theorem: Lyapunov Characteristic Numbers for Dynamical Systems. *Transactions of the Moscow Mathematical Society* **19,** 197–231 (1968).

60. Manjunath, G. & Jaeger, H. Echo State Property Linked to an Input: Exploring a Fundamental Characteristic of Recurrent Neural Networks. *Neural Computation* **25.** Epub 2012 Dec 28, 671–696 (Mar. 2013).

61. Van den Bosch, M., van Gaans, O. & Lunel, S. V. *Stochastic Mackey-Glass Equations and Other Negative Feedback Systems: Existence of Invariant Measures* 2026. arXiv: 2605.14134 [math.DS]. https://arxiv.org/abs/2605.14134.

62. Arnold, L. & Crauel, H. in, 1–22 (Jan. 1991). ISBN: 978-3-540-54662-7 (Print) 978-3-540-46431-0 (Online).

63. Ceni, A. & Gallicchio, C. *Edge of stability echo state networks* 2023. arXiv: 2308.02902 [cs.LG]. https://arxiv.org/abs/2308.02902.

64. Jaeger, H. The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* **148** (Jan. 2001).

65. Bauer, F. L. & Fike, C. T. Norms and exclusion theorems. *Numerische Mathematik* **2,** 137–141. https://api.semanticscholar.org/CorpusID:121278235 (1960).

66. Feuvrie, B., Diop, M. & Wang, Y. Efficient Baseband Digital Predistorter Using LUT for Power Amplifier (PA) with Memory Effect. *American Journal of Electrical and Electronic Engineering* **2,** 72–81 (2014).

67. Akiba, T., Sano, S., Yanase, T., Ohta, T. & Koyama, M. *Optuna: A Next-generation Hyperparameter Optimization Framework* in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, 2019), 2623–2631.

68. Matzner, F. & Mráz, F. *Locally Connected Echo State Networks for Time Series Forecasting* in *The Thirteenth International Conference on Learning Representations* (2025). https://openreview.net/forum?id=KeRwLLwZaw.

69. Jaeger, H. & Haas, H. Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. *Science (New York, N.Y.)* **304,** 78–80 (May 2004).

70. Nowak, A. I., Gniecki, Ł., Szatkowski, F. & Tabor, J. *Sparser, Better, Deeper, Stronger: Improving Sparse Training with Exact Orthogonal Initialization* 2024. arXiv: 2406.01755 [cs.LG]. https://arxiv.org/abs/2406.01755.

71. Esguerra, K., Nasir, M., Tang, T. B., Tumian, A. & Ho, E. T. W. Sparsity-Aware Orthogonal Initialization of Deep Neural Networks. *IEEE Access* **11,** 74165–74181 (2023).

72. Bindel, D., Demmel, J., Kahan, W. & Marques, O. On computing givens rotations reliably and efficiently. *ACM Trans. Math. Softw.* **28,** 206–238. ISSN: 0098-3500. https://doi.org/10.1145/567806.567809 (June 2002).

73. Mezzadri, F. *How to generate random matrices from the classical compact groups* 2007. arXiv: math-ph/0609050 [math-ph]. https://arxiv.org/abs/math-ph/0609050.

74. Wringe, C., Trefzer, M. & Stepney, S. Reservoir computing benchmarks: a tutorial review and critique. *International Journal of Parallel, Emergent and Distributed Systems* **40,** 313–351. ISSN: 1744-5779. http://dx.doi.org/10.1080/17445760.2025.2472211 (Mar. 2025).

75. Koryakin, D., Lohmann, J. & Butz, M. V. Balanced echo state networks. *Neural Networks* **36,** 35–45. ISSN: 0893-6080. https://www.sciencedirect.com/science/article/pii/S0893608012002213 (2012).

76. Goudarzi, A., Banda, P., Lakin, M. R., Teuscher, C. & Stefanovic, D. *A Comparative Study of Reservoir Computing for Temporal Signal Processing* 2014. arXiv: 1401.2224 [cs.NE]. https://arxiv.org/abs/1401.2224.

77. Jaeger, H. *Short Term Memory in Echo State Networks* Technical Report 152 (German National Research Center for Information Technology (GMD), 2002).

78. Van der Pol, B. LXXXV. On Oscillation Hysteresis in a Triode Generator with Two Degrees of Freedom. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **43,** 700–719 (1922).

79. Patra, S., Chen, D. & Geiger, R. *Reliability degradation with electrical, thermal and thermal gradient stress in interconnects* in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)* (2013), 1063–1066.

80. Woo, J., Kim, H., Kim, S. H., Han, K. & Hens, C. Echo State Property upon Noisy Driving Input. *Complexity* **2024,** 5593925. https://doi.org/10.1155/2024/5593925 (2024).

81. Antoniades, I. P. *et al.* ECG Heartbeat Classification Using Echo State Networks with Noisy Reservoirs and Variable Activation Function. *Computation* **14,** 49. https://www.mdpi.com/2079-3197/14/2/49 (2026).