DOI: 10.15514/ISPRAS-2026-38(3)-5

Сравнительный анализ методов обучения и архитектур Echo State Network

И.А. Андросов, ORCID: 0000-0002-4090-6582 <i.androsov@kryptonite.ru>

АО «НПК «Криптонит»,
Россия, 115114, г. Москва, наб. Шлюзовая, д. 4.

Аннотация. В работе рассматриваются сети эхо-состояний (Echo State Network, ESN), которые являются одними из самых распространенных способов реализации резервуарных вычислений. Они состоят из рекуррентной нейронной сети, веса которой выбираются один раз и не обучаются, и выходного, обычно линейного, обучаемого слоя. Такой подход позволяет создавать энергоэффективные и быстрые нейронные сети, способные обучаться в режиме реального времени. Но так как веса ESN не обучаются, их выбор становится отдельной задачей, требующей анализа. Работа представляет собой сравнительных анализ и обзор различных архитектур ESN и методов обучения выходного слоя. Анализ построен на практическом опыте использования и теоретических основах, которые включают в себя исследование зависимости динамики резервуара от топологии графа связей и спектра матрицы связей. Для анализа структуры резервуара применяются такие инструменты, как конденсация графа связей, линеаризация динамики и вводится новое понятие графовой памяти. Кроме известных архитектур ESN, в обзор входят менее популярные или ранее не используемые в контексте резервуарных вычислений модели, такие как система реакции-диффузии, один нейрон с запаздыванием с FORCE обучением и нейронные поля. Эффективность подходов проверяется в ходе всесторонних экспериментов по прогнозированию хаотического временного ряда Макки-Гласса. Работа не только служит практическим руководством по выбору архитектуры ESN и метода обучения выходного слоя, но и обосновывает перспективные направления для дальнейших исследований.

Ключевые слова: резервуарные вычисления; рекуррентные нейронные сети; сети эхо-состояний; дифференциальные уравнения; нейронные поля; системы реакции-диффузии; графовая память.

Для цитирования: Андросов И.А. Сравнительный анализ методов обучения и архитектур Echo State Network. Труды ИСП РАН, том 38, вып. 3, часть 1, 2026 г., стр. 87–114. DOI: 10.15514/ISPRAS–2028–38(3)–5.

Благодарности: Автор выражает благодарность своим коллегам Никите Габдуллину, PhD, к.х.н. Антону Расковалову и к.ф.-м.н. Игорю Нетаю за плодотворные дискуссии, а также Василию Долматову за обсуждения и руководство проектом.

A Comparative Study of Training Methods and Architectures of Echo State Networks

I.A. Androsov, ORCID: 0000-0002-4090-6582 <i.androsov@kryptonite.ru>

JSC "Research and production company 'Kryptonite'",
4, Shlyuzovaya Naberezhnaya, Moscow, 115114, Russia.

Abstract. This paper examines echo state networks (ESNs), one of the most prevalent approaches to implementing reservoir computing. An ESN consists of a recurrent neural network with fixed (untrained) weights and a readout layer that is typically linear and trainable. This approach enables the creation of energy-efficient and computationally efficient neural networks capable of real-time learning. However, since ESN weights are not trained, their selection constitutes a separate challenge that requires careful analysis. The present paper provides a comparative analysis and review of various ESN architectures and readout layer training methods. This analysis is based on practical experience with implementations and theoretical foundations, including studies of how reservoir dynamics depend on the topology of the connectivity graph and the spectrum of the connectivity matrix. To examine the reservoir structure, tools such as connectivity graph condensation and linearization of dynamics are utilized, along with the introduction of a novel concept termed graph memory. In addition to well-established ESN architectures, the review includes less common or previously unapplied models in the context of reservoir computing, such as reaction-diffusion systems, a single neuron with delay, FORCE learning, and neural fields. Experimental evaluation is conducted through comprehensive experiments on the chaotic Mackey-Glass time series prediction task. This paper not only serves as a practical guide for selecting ESN architectures and readout layer training methods but also identifies promising directions for future research.

Keywords: reservoir computing; recurrent neural networks; echo state networks; differential equations; neural fields; reaction-diffusion systems; graph memory.

For citation: Androsov I.A. A Comparative Study of Training Methods and Architectures of Echo State Networks. Trudy ISP RAN/Proc. ISP RAS, vol. 38, issue 3, part 1, 2026, pp. 87-114 (in Russian). DOI: 10.15514/ISPRAS-2026-38(3)-5.

Acknowledgements. The author would like to thank his colleagues Dr Nikita Gabdullin, Dr Anton Raskovalov and Dr Igor Netay for fruitful discussions, and Vasily Dolmatov for discussions and project supervision.

1. Introduction

1.1 Reservoir Computing

Reservoir Computing (RC) is a mature yet continuously evolving computational framework that emerged from two independent works by Jaeger [1] and Maass et al. [2], in 2001 and 2002, respectively. A similar concept, the Context Reverberation Network, was proposed earlier by Kirby [3] in 1991. RC performs computations using a ‘reservoir’ and a trainable readout layer, which is typically linear. Abstractly, a reservoir is an open dynamical system – specifically, a system that evolves over time (continuous or discrete) and is driven by an external input signal.

Not all reservoirs are equally effective. Generally, effective reservoirs are high-dimensional, nonlinear, and possess the Echo State Property (ESP). The ESP is a key characteristic ensuring that the reservoir’s internal states are uniquely determined by the history of its input, asymptotically forgetting past inputs beyond a certain time horizon.

Notable RC models include Liquid State Machines (LSMs) [2] and Echo State Networks (ESNs) [1]. Since the reservoir is typically fixed (i.e., not trained), RC is amenable to physical implementations [4], with examples ranging from optical systems to (literally) a bucket of water [5].

1.2 Echo State Network (ESN)

One of the earliest and most well-known RC models is the Echo State Network (ESN) (see fig. 1), described by the following discrete time state update equation:

$$x_{n+1} = (1-\alpha)x_n + \alpha\tanh(Wx_n + s_n + b), s_n = W_{in}p_n, \alpha \in (0,1], \quad (1)$$

where x_n is the vector of reservoir states at time step n, W is the internal connectivity matrix, W_{in} is the input weight matrix, p_n is the external input signal, b is a bias vector, and α is the leaking rate.

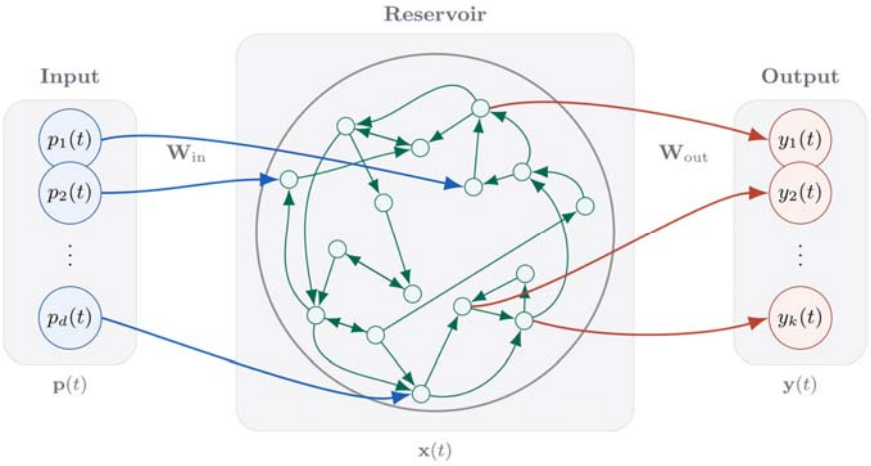


Fig. 1. Schematic diagram of an ESN, illustrating the input layer with signals $p(t)$, the reservoir with states $x(t)$, and the readout layer with outputs $y(t)$, connected via input weights W_{in} and readout weights W_{out}.

This work focuses on ESNs and their variants, though many principles discussed can be applied broadly to other types of reservoirs.

1.3 Differential Equation Formulation of ESN

The ESN update equation (1) can be considered as a discretization of the following continuous time differential equation:

$$\dot{x} = -x + \tanh(Wx + s(t) + b). \quad (2)$$

Applying the forward Euler method to solve this equation with a time step Δt yields:

$$x(t+\Delta t) = x(t) + \Delta t\big(-x(t) + \tanh(Wx(t) + s(t) + b)\big)$$
$$x(t+\Delta t) = (1-\Delta t)x(t) + \Delta t\tanh(Wx(t) + s(t) + b)$$

By setting the leaking rate $\alpha = \Delta t$, we recover the discrete time ESN update rule (1).

Therefore, the standard ESN update equation (1) can be seen as an Euler discretization of the continuous-time differential equation (2) with a proper Δt.

1.4 Related Papers

Several surveys on ESNs exist, each focusing on different aspects. For instance, [6] provides a comprehensive starting point. [7] focuses on multi-reservoir networks, while [8] examines various reservoir architectures (more general than ESN modifications), and [4] reviews physical implementations of reservoir computing.

This work takes a different approach. We focus exclusively on ESN-like models, a much narrower class than general reservoir computing, allowing for a more in-depth analysis. At the same time, we consider architectures that have not been previously utilized in this context, such as neural field models and an ESN reaction-diffusion model. Furthermore, we provide a comparison based on a classical chaotic system prediction task.

1.5 Paper Structure

The remainder of this paper is organized as follows. Section 2 compares several training approaches for the readout layer of ESN. Section 3 examines how the reservoir's graph topology and matrix spectrum influence its computational capabilities. Section 4 discusses various ESN architectures, including those incorporating time-delays and spatially continuous dynamics. Finally, Section 5 presents a comparative evaluation, providing both a theoretical analysis and experimental validation on a chaotic system prediction task using the Mackey-Glass time series.

2. Training Methods

This section compares different training approaches for the readout layer of ESN. We focus exclusively on linear readouts, as this approach is both versatile and widely used. Let x_k be the reservoir state vector at time step k, $X = [x_1, \dots, x_n]$ be the matrix of collected reservoir states, W_{out} be the trainable readout weight matrix, and $\hat{Y} = [\hat{y}_1, \dots, \hat{y}_n]$ be the matrix of desired target outputs. Training methods can be categorized into two main groups: offline and online methods.

2.1 Offline Methods

We begin by discussing offline methods. These methods are based on the least squares principle, aiming to find the weights W_{out} that solve the following optimization problem:

$$\underset{W_{out}}{\operatorname{argmin}} \|XW_{out} - \hat{Y}\|_2^2,$$

which is equivalent to:

$$X^T X W_{out} = X^T \hat{Y}.$$

2.1.1 Ridge Regression

The standard least squares solution [6] is found via the pseudoinverse:

$$W_{out} = (X^T X)^{-1} X^T \hat{Y}.$$

By introducing a Tikhonov regularization parameter $\lambda \geq 0$, we arrive at the *Ridge Regression* solution:

$$W_{out} = (X^T X + \lambda I)^{-1} X^T \hat{Y}. \quad (3)$$

While conceptually simple, this method can suffer from numerical instability due to the explicit matrix inversion operation, especially if $X^T X$ is ill-conditioned.

2.1.2 QR Decomposition

QR decomposition offers a more stable alternative [9]. It decomposes the state matrix X into an orthogonal matrix Q and an upper triangular matrix R ($X = QR$). The least squares problem becomes

$$RW_{out} = Q^T \hat{Y}, \quad (4)$$

which can be efficiently solved via back-substitution. This method is generally more robust than direct inversion, but computationally slightly more expensive.

Note that this method also incorporates a regularization parameter λ. This can be implemented by augmenting the system with $[X, \sqrt{\lambda}I]^T$ and $[\hat{Y}, 0]^T$.

2.1.3 Singular Value Decomposition (SVD)

SVD [9] provides the most numerically stable approach by decomposing X into $U\Sigma V^T$. Here, U and V are orthogonal matrices, and $\Sigma = \mathrm{Diag}\,(\sigma_1, \ldots, \sigma_n, 0, \ldots, 0)$ is a diagonal matrix of the singular values. The solution is then computed via the Moore-Penrose pseudoinverse, X^+, given by:

$$X^+ = V\Sigma^+ U^T,$$

where $\Sigma^+ = \mathrm{Diag}\,(\sigma_1^{-1}, \ldots, \sigma_n^{-1}, 0, \ldots, 0)$. This results in the final solution for the weights:

$$W_{out} = X^+\hat{Y}. \tag{5}$$

It typically has the highest computational cost among offline methods but offers the best stability.

Similarly, to the QR approach, this method also includes a parameter λ for regularization.

2.1.4 Truncated SVD (TSVD)

Truncated SVD (TSVD) [10] is a modification of SVD that offers an additional level of regularization. Small singular values (smaller than some parameter ε) in the matrix Σ, which are often associated with noise, can be truncated by setting them to zero. This approach is closely related to Principal Component Analysis (PCA) [11], as truncating singular values effectively reduces the dimensionality of the solution space.

Consequently, this method employs two types of regularization.

2.2 Online Methods

Online methods circumvent the need to store and process the entire data matrix X at once, making them suitable for streaming data and more robust against the ill-conditioning of the data matrix.

These methods are particularly relevant for time series prediction. When predicting a time series, two primary strategies are employed. First, the readout can be trained to predict k steps ahead. Alternatively, it can be trained to predict only the next step, with subsequent predictions made by feeding the output back into the reservoir. The first approach increases the number of output parameters by a factor of k, and the interpretation of the output weights becomes less straightforward. While the update rule for each strategy is the same in both the LMS (2.2.1) and RLS (2.2.2) algorithms, the FORCE (2.2.3) algorithm is capable of implementing only the second strategy.

2.2.1 Least Mean Squares (LMS) Filter

For a general nonlinear readout, training typically involves gradient descent. For a linear readout and a Mean Squared Error (MSE) loss function, gradient descent takes the form of the *Least Mean Squares (LMS) filter* [12]:

$$W_{n+1} = W_n - \eta x_n(\hat{y}_n - W_n x_n)^T, \tag{6}$$

where W is the readout weight matrix and η is the learning rate. This update rule is a LMS filter.

2.2.2 Recursive Least Squares (RLS) Filter

The Recursive Least Squares (RLS) [12] filter is another online method. It employs a similar update rule to LMS, but the constant learning rate η is replaced by a time-varying inverse covariance matrix P:

$$P_n = \frac{1}{\lambda}\left(P_{n-1} - \frac{P_{n-1}x_n x_n^T P_{n-1}}{\lambda + x_n^T P_{n-1} x_n}\right), P_0 = \frac{I}{\delta},$$
$$W_{n+1} = W_n + P_n x_n(\hat{y}_n - W_n x_n)^T, \tag{7}$$

where $\lambda \in (0,1]$ is a forgetting factor (typically close to 1, e.g., 0.999).

This algorithm has a higher computational complexity, typically $O(N^2)$ (where N is the reservoir size) compared to $O(N)$ for LMS, but it converges significantly faster to the optimal weights.

2.2.3 FORCE Learning

FORCE (**F**irst-**O**rder, **R**educed and **C**ontrolled **E**rror) [13] is an algorithm specifically designed for online training, particularly effective for generating complex temporal patterns. A key distinction of FORCE from standard RLS/LMS is the use of a feedback loop from the readout layer back to the reservoir *during* the training phase (see fig. 2). This mechanism allows the reservoir states to adapt in response to training, thereby reducing subsequent prediction errors. Consequently, two primary variations of FORCE learning exist: RLS-FORCE and LMS-FORCE, distinguished by the online algorithm used for the readout layer.

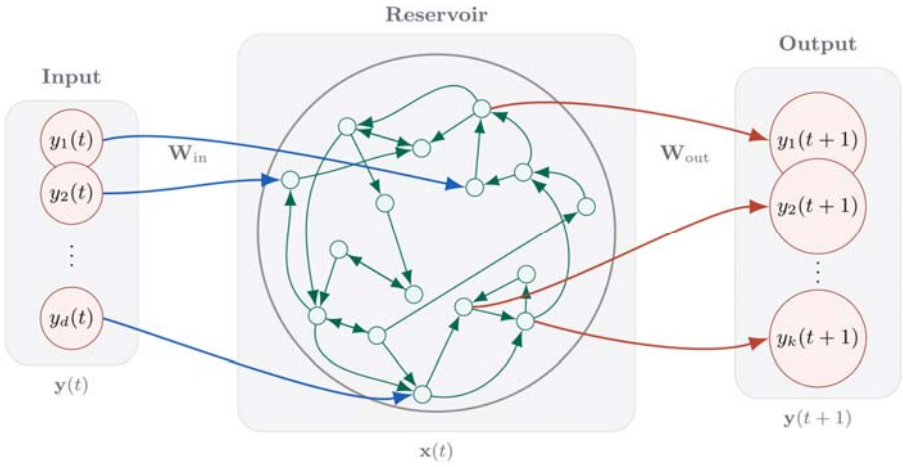


Fig. 2. Schematic diagram of the FORCE learning architecture in an ESN, showing the feedback loop from the output $\mathbf{y}(t+1)$ back to the reservoir during training.

2.4 Conclusion

For smaller datasets we can effectively use offline methods. The simplest of these is Ridge regression (2.1.1). For improved accuracy and complexity, you can move to QR decomposition (2.1.2), and for even higher precision, SVD (2.1.3). A modified version of SVD, called TSVD (2.1.4), is often used specifically as a regularization technique to prevent overfitting.

For more complex problems, it's often necessary to use online methods. A great starting point is the LMS (2.2.1) algorithm, which is known for its simplicity. For a more sophisticated and often more powerful approach, you can advance to RLS (2.2.2). Additionally, FORCE (2.2.3) algorithm is a specialized but highly effective option, particularly when your task requires the algorithm's input to be a function of its own output.

Other methods, such as Backpropagation De-correlation (BPDC) [14], also exist but are generally based on backpropagation, minor modifications of the aforementioned methods, and on Adaptive Filter Theory [12] and are beyond the scope of this paper. In practice, RLS-FORCE often

demonstrates superior effectiveness for time series generation tasks, while RLS is well-suited for a broader range of general problems.

3. Reservoir Structure

This section examines the impact of the reservoir's internal structure on its computational capabilities. We first examine the influence of the reservoir's connectivity graph and then explore methods for selecting the weight matrix spectrum.

Building upon these discussions, we then analyze various ESN architectures and their respective advantages and disadvantages.

3.1 Connectivity Graph

A connectivity graph is a directed graph where neurons are vertices, and a directed edge exists from vertex i to vertex j if the corresponding weight w_{ji} is non-zero (see fig. 3). Here, we focus on the existence of connections rather than their precise values.

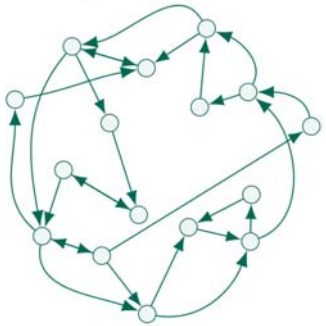


Fig. 3. Directed connectivity graph of an ESN,
where nodes represent neurons and directed edges indicate non-zero weights in the connectivity matrix.

3.1.1 Condensation Graph

The structure of the connectivity graph is particularly important when the graph is not strongly connected. Let us recall the definition of strong connectivity.

Definition 2. *A directed graph is called strongly connected if there is a path in each direction between each pair of vertices of the graph.*

If a graph is not strongly connected, signals from some neurons may never reach others. We can decompose the graph into its strongly connected components (SCCs).

Definition 3. *A strongly connected component (SCC) of a directed graph is a maximal subgraph in which every pair of vertices is mutually reachable (i.e., the subgraph is strongly connected).*

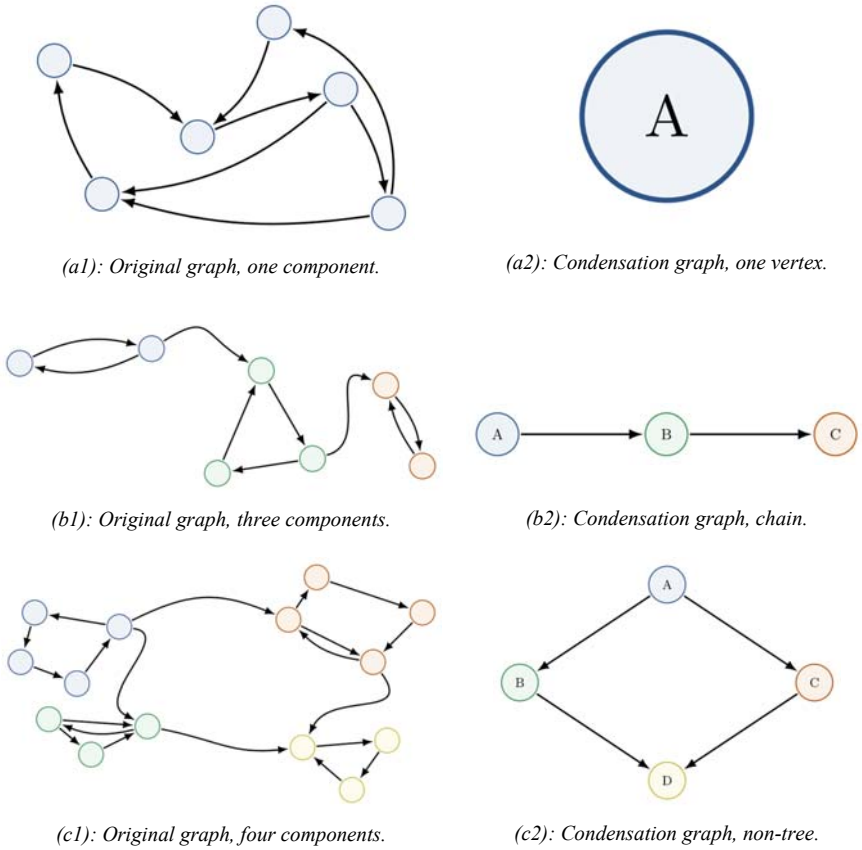
A new directed graph, the *condensation graph*, can then be constructed where vertices represent the SCCs (see fig. 4).

Definition 4. *The condensation graph of a directed graph G is a new directed graph formed by contracting every strongly connected component of G to a single vertex. An edge connects one component-vertex to another if there is an edge in G from any vertex in the first component to any vertex in the second.*

Let us recall the definition of a directed acyclic graph.

Definition 5. *A directed acyclic graph (DAG) is a directed graph with no directed cycles.*

Obviously, the condensation graph is always a DAG.



(a1): Original graph, one component. *(a2): Condensation graph, one vertex.*

(b1): Original graph, three components. *(b2): Condensation graph, chain.*

(c1): Original graph, four components. *(c2): Condensation graph, non-tree.*

Fig. 4. Examples of original graphs and their condensation graphs.

A non-strongly connected reservoir can thus be viewed as a deep or layered network (allowing connections through layers), where SCCs form the layers. This allows for the independent study of its components and can create functional hierarchies within the reservoir. Further, we assume strong connectivity.

3.1.2 Graph Memory (GM)

In certain scenarios, the path length between neurons becomes significant. In the continuous time ESN model, connection strengths are regulated by weight values, and path length is not a primary factor. However, a large discretization step Δt (meaning fewer steps over the same total time) in the discrete time model introduces additional dynamic effects. For instance, if the shortest path between two neurons is through five edges, signal from the first neuron will reach the second only after five-time steps. This discrete propagation creates a form of memory embedded in the graph topology itself, which we will call *graph memory*[1] (see fig. 5). This property can be beneficial, as it introduces additional non-locality and can potentially mitigate neuron synchronization.

[1] In [3], this was called 'topographic context'.

However, relying on a large time step to achieve graph memory is suboptimal, as it often appears more as a discretization artifact than an intentional design choice. Alternative methods for inducing graph memory are discussed in Section 4sec:ReservoirArchitectures.

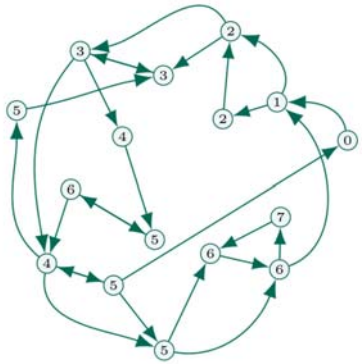


Fig. 5. Directed graph demonstrating the concept of GM, with vertices labeled by the shortest path distances from a chosen starting node (labeled 0), highlighting how signal propagation delays depend on graph topology.

3.1.3 Sparse Connectivity Matrices

In Jaeger's work [15], it is claimed that low-density (sparse) connectivity enhances computational performance. Sparsity can contribute to graph memory, especially with a large time step, or lead to a non-strongly connected graph with a layered structure. Another practical advantage is that sparse connectivity matrices enable the use of specialized multiplication algorithms, which can be significantly faster and more memory efficient.

3.2 Spectral Properties

Now, let us consider the properties of the weights themselves. Since the connectivity matrix W is typically chosen randomly and remains fixed, its spectral properties are crucial.

3.2.1 Ring Spectrum

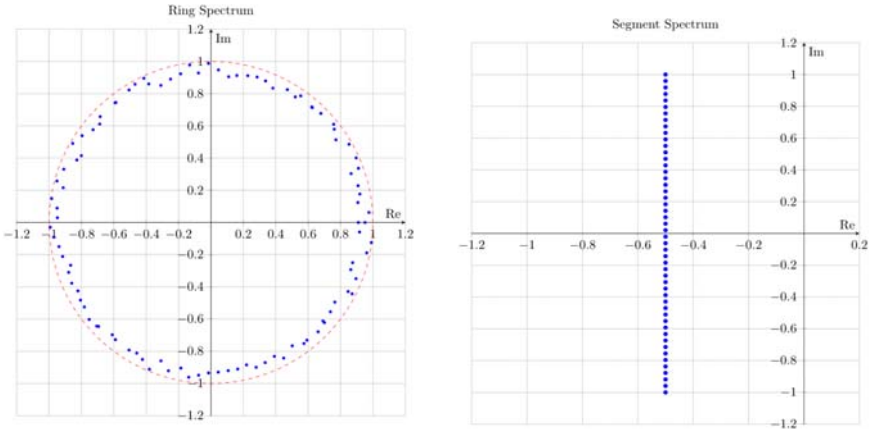
In the discrete time model (1), the dynamics near the origin ($x = 0, s = 0$) are approximated by the linear map $x_{n+1} = ((1-\alpha)I + \alpha W)x_n$. Let $M = (1-\alpha)I + \alpha W$. The Echo State Property is typically satisfied if the spectral radius $\rho(M)$ is less than one [13]. It is possible to have ESP with a spectral radius greater than 1, but this often leads to chaotic behavior of the system [6].

Richer dynamics are achieved when the eigenvalues of M have similar absolute values. Achieving exact equality is avoided because generating a numerically stable, sparse orthogonal matrix with a specific sparsity pattern is a non-trivial task. This leads to a spectrum concentrated on a 'ring' in the complex plane (see fig. 6a). The distribution of oscillation frequencies (the arguments of the eigenvalues) contributes to neuron desynchronization, which in turn enhances the reservoir's computational capabilities.

3.2.2 Segment Spectrum

Now, consider the continuous time model (2). The linearization near the origin is $\dot{x} = (W - I)x$. The Jacobian of the system is $M = W - I$. For the system to be stable, the eigenvalues of M must have negative real parts. This condition is analogous to $\rho(M) < 1$ in the discrete case. The imaginary parts of the eigenvalues determine the oscillation frequencies.

In the continuous time case, it is possible to decouple the decay rates (real parts) from the oscillation frequencies (imaginary parts). To achieve rich dynamics, it is desirable to have a wide range of oscillation frequencies but a common decay rate. This can be achieved by constructing W such that its eigenvalues lie on a vertical line in the complex plane (a 'segment' spectrum, see fig. 6(b)). A simple way to do this is to set $W = S - \gamma I$, where S is a skew-symmetric matrix ($S^T = -S$) and $\gamma > 0$. The eigenvalues of S are purely imaginary, so the eigenvalues of W are $i\omega_k - \gamma$, and the eigenvalues of the Jacobian $M = W - I$ are $i\omega_k - \gamma + 1$.



(a): Discrete time ESN. *(b): Continuous time ESN.*

Fig. 6. Eigenvalue distributions for reservoir matrices. (a) A 'ring' spectrum for the matrix M in a discrete time ESN, close to the unit circle (dashed line). (b) A 'segment' spectrum for the matrix M in a continuous time ESN, ensuring a common decay rate with diverse oscillation frequencies.

The skew-symmetric architecture is used in [15], but for the differential equation:

$$\dot{x} = \tanh\big(Wx + s(t)\big). \tag{8}$$

Architecture (9) is called *Euler State Network* [17].

3.3 Input Matrix

The input matrix W_{in} (used to compute $s_n = W_{in}p_n$) determines how the external signal influences on the reservoir state. In a randomly connected reservoir, which neurons receive the input may not be critical. However, in structured reservoirs, the choice of input neurons is important. In all cases, the scaling of the input signal significantly affects the dynamics. If the input is too weak, it may not sufficiently excite the reservoir's nonlinearities; if it is too strong, it may overwhelm the internal dynamics and erase the reservoir's memory. General guidelines for scaling are provided in [6].

We define two hyperparameters. First, p_{in} denotes the probability of a neuron receiving an input signal. Second, $normal_{in}$ represents the Euclidean norm of the weights for each input signal coordinate.

4. Reservoir Architectures

This section presents several ESN architectures and analyzes them using the concepts discussed previously. We will primarily use the continuous time formulation for clarity.

4.1 Standard ESN

This is the foundational architecture, given by equation (2). We denote the continuous time standard variant as ESN (DE). As discussed, for its discrete time counterpart, a 'ring' spectrum for the update matrix is often preferred, while for the continuous time form, a 'segment' spectrum for the internal weight matrix W is advantageous. The latter is referred to as ESN (skew). We use the term 'ESN' to denote the classical baseline with a random weight matrix without any additional structure.

To numerically integrate the differential equation in ESN (DE) and ESN (skew), the forward Euler method with step 0.1 is used.

ESN models utilize a connectivity graph, which can be constructed in various ways. We use a random graph where the probability of an edge existing is equal to the hyperparameter p_{rec}. For ESN we denote the spectral radius of weights matrix W as $sr = \rho(W)$. For ESN (DE) we denote $sr_{re} = \max \mathrm{Re}\ \lambda_i$ and $sr_{im} = \max \mathrm{Im}\ \lambda_i$, where λ_i are eigenvalues of W.

Let us consider alternative formulations.

4.1.1 W outside tanh

One variant places the weight matrix W outside the activation function:

$$\dot{x} = -x + W\tanh(x + s(t) + b) \tag{9}$$

The primary difference is that the neuron states $x(t)$ are no longer bounded by the range of tanh. This formulation is closer to the Amari model of neural fields, discussed later. However, consistency suggests placing both recurrent weights W and input $s(t)$ either inside or outside tanh, as both are summed neuronal inputs. For this reason, we will not use this model.

4.1.2 Unbound model

Another modification of previous equation (9) is

$$\dot{x} = -x + W\tanh(x) + s(t) + b. \tag{10}$$

With some condition on weights matrix W, it is a discretization of Amari's neural field model, which is discussed in Section 4.5.1. In this case, the influence of input signal $s(t)$ is not bounded. This is often undesirable for many tasks; consequently, this model is rarely employed in practice.

4.1.3 Non-dissipation

Another variant is obtained by removing the linear dissipation term $-x$:

$$\dot{x} = \tanh(Wx + s(t) + b). \tag{11}$$

This architecture is analyzed in [17] and called an *Euler State Network* (EuSN). While the explicit linear dissipation term is absent, the system can still be dissipative through its nonlinear dynamics, though the dissipation is no longer uniform across the state space. This can undermine the benefits of a simple segment spectrum.

An alternative form places W outside the tanh:

$$\dot{x} = W\tanh(x + s(t) + b). \tag{12}$$

If W is invertible, equations (11) and (12) are equivalent up to a linear change of coordinates $y = Wx$. Since the readout is also linear, these two systems have identical computational capabilities.

To numerically integrate this equation, the forward Euler method with a smaller step size 0.01 is used, as the equation exhibits fewer stable dynamics compared to dissipative variants. This system has the same hyperparameters, as a ESN (DE).

4.2 Nonlinear Vector AutoRegressive (NVAR) Models

While not strictly an ESN, the Nonlinear Vector Autoregressive (NVAR) model is a useful baseline. An NVAR model's state is composed of delayed input signals, and the readout maps polynomial function (typically degree 2) of these states to the output:

$$y(t) = P\big(s(t), s(t-1), \dots, s(t-M)\big). \tag{13}$$

Its simplicity helps to understand the core mechanisms of reservoir computing. Interestingly, this simple model is sufficient for certain tasks and serves as a valuable baseline, helping to isolate the contribution of memory from that of complex dynamics. The use of NVAR in the context of reservoir computing was notably highlighted in [19] paper. Another notable example is [17] paper, where NVAR is called a 'Next generation reservoir computing'. This example provides a useful comparison base for the following model.

4.3 Single Neuron with Delayed Feedback

This system, comprising a single neuron with delayed feedback, is conceptually similar to NVAR models and was explored in particular in [20] and [21]. The state is constructed from the history of a single nonlinear node. The governing equation is a delay differential equation:

$$\dot{x}(t) = -x(t) + \tanh\big((w * x)(t)\big), \tag{14}$$

where '$*$' denotes convolution over $[0, \infty)$ or $[0, T]$.

After time discretization, the convolution kernel w can be represented by a weight matrix connecting past states to the present, which can then be trained using the FORCE algorithm (see fig. 7). Since w is trainable, all weights are trainable, and $x(t-\Delta t), x(t-2\Delta t), \dots, x(t-T)$ is an approximately $\frac{T}{\Delta T}$ previous points in time series. Hence, the reservoir predicts the next value in time series based on $\frac{T}{\Delta T}$ previous points, which is similar to the NVAR model. But unlike the NVAR model, there is a DE and tanh-based nonlinearity instead of polynomial.

In general, this model can be better than NVAR because of more nonlinearity but suffers from the same problems. First, dynamics of virtual neurons (past neuron states) is a simple shift:

$$\big(x(t), x(t-\Delta t), \dots, x(t-T)\big) \mapsto \big(x(t+\Delta t), x(t), x(t-\Delta t), \dots, x(t-T+\Delta t)\big), \tag{15}$$

which is trivial. Second, reservoir cannot compress past but memorize exactly the previous values. In a strict sense, this model and NVAR might not be considered true reservoirs, as they lack the core component with fixed, non-trainable dynamics.

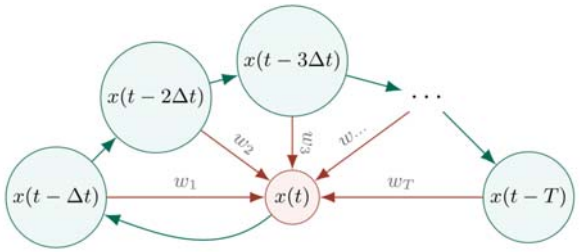


Fig. 7. Schematic diagram of a single neuron with delayed feedback, where the current state $x(t)$ is influenced by weighted past states $x(t-\Delta t), x(t-2\Delta t), \dots, x(t-T)$, approximating a delay differential equation.

To numerically integrate this equation, we use the Euler method with a step size of $1/k$, where $\Delta t = 1$ and k is a hyperparameter.

4.4 ESN with Delays

The next model is ESN with delays. There are several real neurons with some delay in connections between neurons (e.g., based on distance).

$$\dot{x}_i(t) = -x_i(t) + \tanh\left(\sum_j w_{ij}\, x_j\big(t - h_{ij}\big) + s_i(t) + b_i\right), \tag{16}$$

where h_{ij} is the delay from neuron j to neuron i. The goal is to explicitly create graph memory. The disadvantage is that the dynamics can be dominated by simple shifts, and the delays must be chosen carefully. However, even a system of two neurons with delays can exhibit very rich dynamics [22], but its analysis requires different tools than those presented here. Hence, this model remains purely theoretical.

4.5 Neural Field Models (NF)

Neural field models are used in computational and mathematical neuroscience [23]. These models are not used for reservoir computing (but the idea of using it was proposed in [3]). There are reasons for this. All these models are continuous not only in time, but also in space. But since any computer simulation requires discretization, these models can be viewed as a special case of ESN. The weight matrix W in corresponding ESN does not have good properties, discussed in Section 3, since it is often symmetric.

At the same time, these models have two nice properties. First, these networks are easy to scale, since it is just a discretization step similar to the delayed ESN in Section 4.4. Second, they naturally incorporate a form of graph memory (see fig. 8) with non-trivial dynamics (unlike the simple shifts in delay-based ESNs), which is more akin to the incidental graph memory discussed in Section 3.1.2. Moreover, neural field models are much easier to compute, as they use convolution instead of matrix-vector multiplication, like in (18).

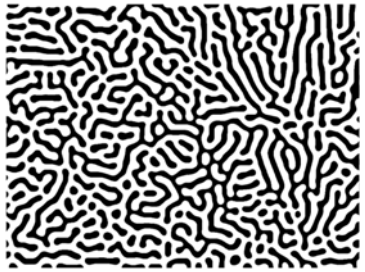


Fig. 8. Example of a spatial activation pattern generated by a neural field model, exhibiting Turing-like pattern [24].

4.5.1 Amari Model

The Amari neural field model ([25], [26]) is given by:

$$\dot{u}(x,t) = -u(x,t) + \int_B w\,(x,y)\tanh(u(y,t))dy + s(t) + b, x, y \in B \tag{17}$$

Often, the kernel is chosen to be space-invariant, $w(x,y) = w(|x-y|)$, making the resulting weight matrix symmetric after discretization. This precludes oscillations, which are generally beneficial for reservoir computing.

Alternatively, an ESN-like formulation can be used to ensure bounded states:

$$\dot{u}(x,t) = -u(x,t) + \tanh\left(\int_B w\,(x,y)u(y,t)dy + s(t) + b\right), x, y \in B, \tag{18}$$

where w is a 'Mexican hat' (see fig. 9):

$$w(x,y) = w(|x-y|) = \frac{a}{\pi\sigma^4}\left(1 - \left(\frac{x^2+y^2}{2\sigma^2}\right)\right)e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{19}$$

Here, a and σ are hyperparameters. To numerically integrate this equation, we use the forward Euler method with step 0.1.

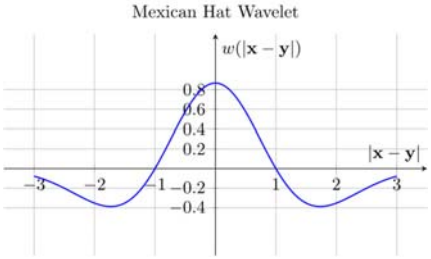


Fig. 9. The Mexican hat kernel function w, characterized by short-range excitation and longer-range inhibition.

4.5.2 Nunez Model

Another model extends the previous one by introducing delays [27]. Hence, this model has two types of graph memory.

$$\dot{u}(x,t) = -u(x,t) + \int_B w\,(x,y)\tanh(u(y,t-h(x,y)))dy + s(t) + b, x, y \in B \tag{20}$$

This model requires additional analysis to use appropriate delays for computation, and analysis of such a system is much harder. Moreover, we will see that it is better to use another neural field model where delays are added. Hence, this model remains purely theoretical.

Other neural field models, that do not based on ESN can be found in [28].

4.5.3 Neural Field (skew)

This neural field model is inspired by the ESN (skew) architecture, but in neural field form. That is, we use $w(x,y) = -w(y,x)$ for $x \neq y$. We employ the ESN-like bounded equation:

$$\dot{u}(x,t) = -u(x,t) + \tanh\left(\int_B w\,(x,y)u(y,t)dy + s(t) + b\right), x, y \in B \tag{21}$$

As a concrete example, we employ convolution on a discrete lattice with the following kernel:

$$\begin{pmatrix} 0 & -a & 0 \\ -a & b & a \\ 0 & a & 0 \end{pmatrix},$$

where $b = sr_{re}$, but $|a| \neq sr_{im}$. Instead of using sr_{im}, we use the parameter a, since it has the same meaning.

To numerically integrate this equation, we use the forward Euler method with step 0.1.

4.6 Reaction–Diffusion System (R-D System)

A final class of models to consider are reaction-diffusion systems. The concept of employing such models was first introduced in [3]. The paper [29] illustrates how chemical computing can be realized in a reaction-diffusion system, specifically through the dynamics of the Belousov–Zhabotinsky reaction. Fig. 8 is also an example of a reaction-diffusion system model.

Instead of using a convolution kernel, this model is using a diffusion – Laplacian Δ_x. To add oscillation, it is needed to use two components – one excitatory (u) and one inhibitory (v):

$$\begin{cases} u_t = d_u \Delta_x u + \alpha\big(-u + \tanh\big(\gamma u - \omega v + s(t)\big)\big) \\ v_t = d_v \Delta_x v + \alpha(-v + \tanh(\gamma v + \omega u)), \end{cases} \quad (22)$$

where $d_u, d_v, \omega(x) > 0$.

While d_u, d_v, γ are constant hyperparameters, ω may vary from point to point. For this reason, we introduce two more hyperparameters ω_μ and ω_σ. Thus, $\omega \sim |\mathrm{Normal}(\mu = \omega_\mu, \sigma = \omega_\sigma)|$.

To numerically integrate this equation, we use the forward Euler method with step 0.1.

5. Comparative Study

This section presents a comparative experimental evaluation on chaotic system prediction (using the Mackey-Glass time series) along with a theoretical comparison of ESN architectures.

5.1 Theoretical Comparison

This section provides a theoretical comparison of the architectures. Their key characteristics are summarized in table 1.

Implementing ESNs in continuous time is a natural extension, as it allows for the handling of arbitrary discretization steps. A challenge arises with graph memory (GM): while GM can be incorporated into discrete time ESNs, it is not compatible with the classic continuous time architecture. Furthermore, configuring GM in discrete-time models is difficult and appears as a discretization artifact rather than a fundamental feature.

Table 1. Theoretical comparison of architectures. Non-generative tasks refer to problems like classification, as opposed to time-series generation.

#	Architecture	Non-generative Task	Useful Dynamics	GM	Scalability
1.2	ESN	+	+	+	-
1.3	ESN (DE)	+	+	+	-
4.3	Delay Neuron	-	+	+	+
4.4	Delay ESN	+	+	+	+
4.5.1	Neural Field	+	-	+	+
4.5.2	Delay Neural Field	+	+	+	+
4.5.3	Neural Field (skew)	+	+	+	+
4.6	R–D System	+	+	+	+

One method for incorporating GM into a differential ESN is by introducing delays. An ESN with delays exhibits rich dynamics and effectively adds virtual neurons. A key limitation, however, is that these virtual neurons have simplistic dynamics, merely performing a shift operation. This

suggests that more computationally efficient methods for adding GM may exist. A special case of a delay-based ESN is a single neuron with delayed feedback, which is similar to a NVAR model. This model is notable for having no non-trainable parameters but is primarily limited to generative tasks. A significant drawback of delay-based models is their scalability; while adding more virtual neurons is as simple as decreasing the discretization step, this can lead to computational inefficiencies.

Another approach to integrating GM is through neural field models. These models are spatially continuous systems that do not rely on virtual neurons and offer excellent scalability. Moreover, since their connection weights are based solely on distance, they are computationally inexpensive. The primary disadvantage of classical neural field models is their lack of sufficiently complex dynamics for many tasks. While this can be mitigated by adding delays to the neural field, which introduces two forms of GM, this reintroduces the issue of virtual neurons with simple shift dynamics. More promising alternatives are the R-D system and Neural Field (skew). These frameworks inherently provide GM and useful dynamics while remaining computationally efficient and scalable.

5.2 Mackey-Glass Equation

First, we evaluate the architectures using a classic reservoir computing benchmark: forecasting the Mackey-Glass time series [30]. The first application of ESNs to this task involved predicting the 84th point of the time series [31]. According to [32], this is an ideal benchmark within the class of known dynamics.

The Mackey-Glass equation is given by the following delay differential equation:

$$\dot{x} = \frac{\beta x(t-\tau)}{1 + \big(x(t-\tau)\big)^n} - \gamma x(t) \quad (23)$$

We use this equation with the following parameters: $n = 10, \beta = 0.2, \gamma = 0.1, \tau = 17$:

$$10\dot{x} = \frac{2x(t-17)}{1 + \big(x(t-17)\big)^{10}} - x(t) \quad (24)$$

The equation with these parameters exhibits chaotic behavior; thus, our problem is forecasting a chaotic system. The initial values for the interval $[x(t-\tau), x(0)]$ are drawn from a uniform distribution on $[1.1, 1.3]$. The initial value problem is solved using the Euler method with a step size of $\Delta t = 0.1$. The time series is then generated by sampling points at unit intervals (step = 1).

The first 1000 points serve as a ‘warmup’ (or washout) period (see fig. 10). These points are fed into the reservoir to eliminate initial transience, but the readout layer is not updated during this phase.

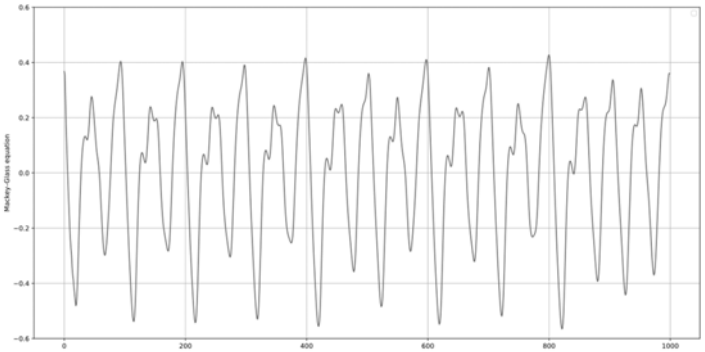


Fig. 10. Time series segment from the Mackey-Glass equation during the warmup period.

The subsequent 3000 points are used for readout training, followed by 1000 or 400 points used for testing. To measure the error during testing, the Normalized Root Mean Square Error (NRMSE) is used:

$$NRMSE = \frac{\sqrt{\text{MSE}}}{\sigma_{truth}}.$$

Note that the NRMSE is equal to 1 if the prediction is a constant at the mean. Consequently, an NRMSE greater than 1 indicates performance worse than a constant mean prediction and is considered a failure.

Before feeding the time series into the reservoir, the data is normalized to a mean of zero and a standard deviation of 0.25 using only the warmup and training data; the test data is then normalized using these same coefficients.

All architectures have hyperparameters that need to be optimized. To achieve this, we fix a seed for the initial values and use Optuna [33] with the default Tree-structured Parzen Estimator (TPE) sampler for 1000 epochs to optimize the hyperparameters. One hyperparameter (the number of nodes and weights in the readout layers) is fixed at 400 for all architectures. Hence, the neural dield model is 20×20, and the R-D system is a two-component 20×10 system.

Once the hyperparameters are chosen, each architecture is tested across 20 different seeds to produce 20 NRMSE forecasting error values based on different initial conditions.

5.3 Training Methods Comparison

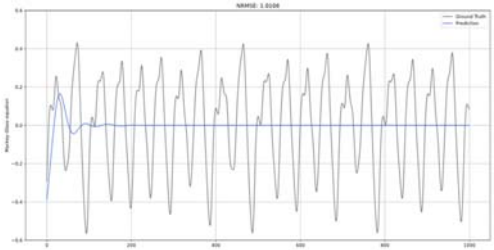
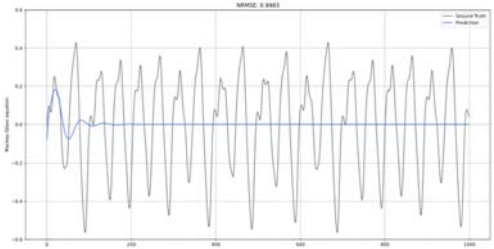
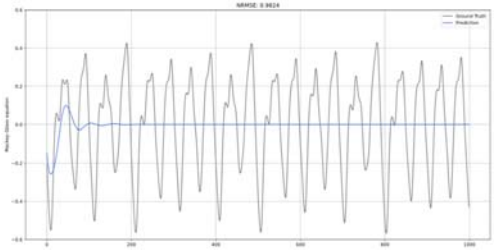
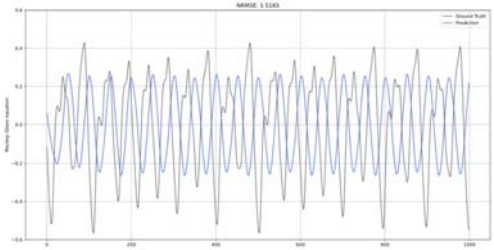
First, we compare training methods on two architectures: ESN (DE) with a random weight matrix and ESN (skew) with a skew-symmetric matrix. The results are presented in table 2 and table 3.

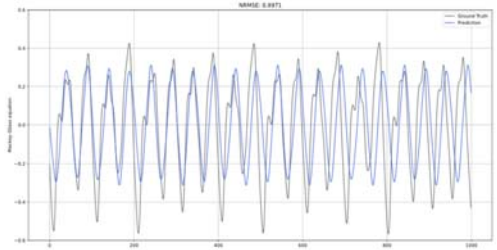
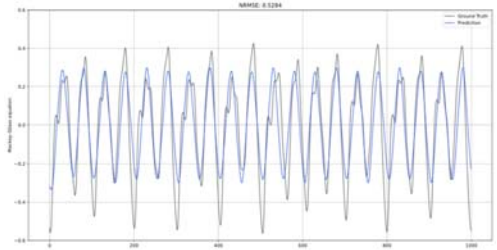
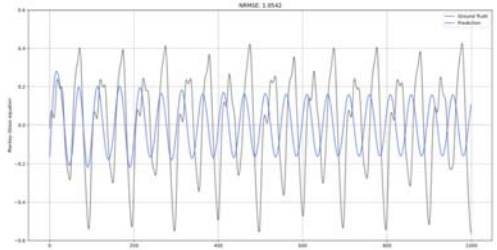
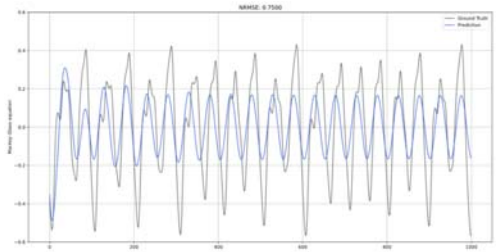
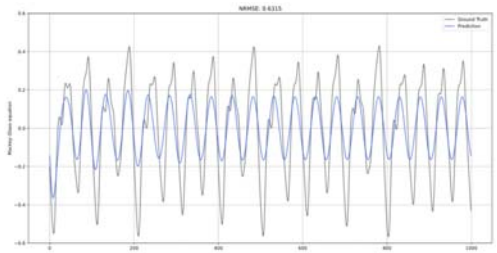
Table 2. Comparison of training methods (W: Worst, M: Mean, S: Std. Dev., B: Best; numbers indicate prediction length). DNC: Did not converge.

Arch	Method	W1000	M1000	S1000	B1000	W400	M400	S400	B400
skew	**RLS-FORCE**	**0.895**	**0.412**	0.237	**0.013**	**0.353**	**0.066**	0.074	**0.005**
skew	LMS-FORCE	1.759	1.236	0.272	0.730	1.721	1.083	0.343	0.581
skew	RLS	1.054	0.751	0.105	0.631	0.861	0.688	0.075	0.600
skew	LMS	6.035	3.423	1.701	1.055	5.715	3.163	1.683	0.908
skew	SVD/TSVD	DNC	DNC	DNC	DNC	DNC	DNC	DNC	DNC
skew	QR	DNC	DNC	DNC	DNC	DNC	DNC	DNC	DNC
skew	Ridge	5.289	2.087	1.000	0.999	2.723	1.217	0.476	0.615
DE	**RLS-FORCE**	**1.431**	**0.766**	0.285	**0.352**	**0.984**	**0.405**	0.249	**0.034**
DE	LMS-FORCE	5.270	5.369	0.046	5.452	5.233	5.375	0.069	5.490
DE	RLS	3.585	2.411	0.876	0.999	2.707	1.694	0.618	0.997
DE	LMS	5.455	1.401	0.955	0.979	3.163	1.231	0.504	0.618

Arch	Method	W1000	M1000	S1000	B1000	W400	M400	S400	B400
DE	SVD/TSVD	1.576	1.137	0.142	1.003	1.325	1.128	0.104	1.007
DE	QR	1.517	0.883	0.307	0.528	1.446	0.767	0.273	0.488
DE	Ridge	1.011	0.998	0.008	0.982	1.027	0.995	0.021	0.955

Table 3. Graphical comparison of converged training methods. The mean graph is chosen as the seed with the NRMSE nearest to the mean.

Arch	Type	Graph
DE + Ridge	Worst	
DE + Ridge	Mean	
DE + Ridge	Best	
DE + QR	Worst	

Arch	Type	Graph
DE + QR	Mean	
DE + QR	Best	
skew + RLS	Worst	
skew + RLS	Mean	
skew + RLS	Best	

As shown in table 2, the ESN (skew) architecture combined with RLS-FORCE outperforms all other configurations. At the same time, it is evident that only RLS-FORCE is capable of capturing the underlying chaotic dynamics. Other learning methods may achieve an error less than one, but their predictions are merely periodic rather than chaotic. Consequently, subsequent architectures will be compared only using the RLS-FORCE learning method.

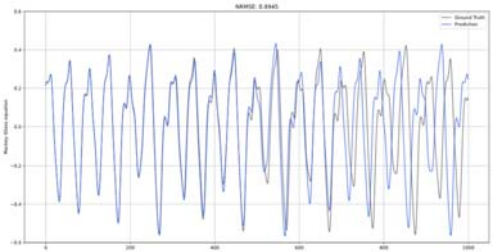
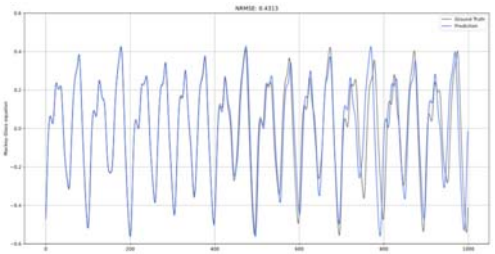
5.4 Architectures Comparison

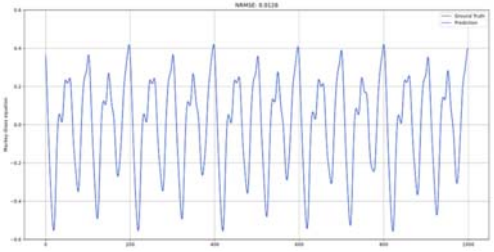
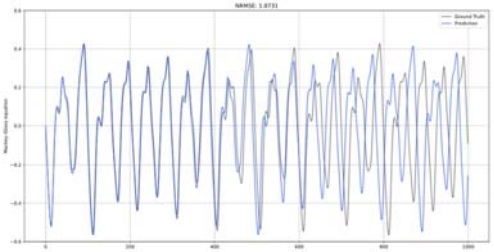
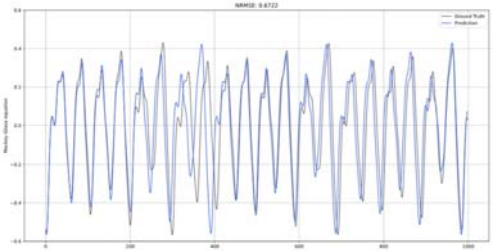
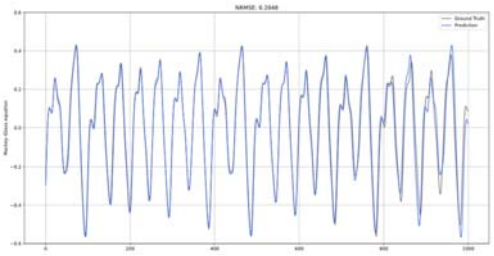
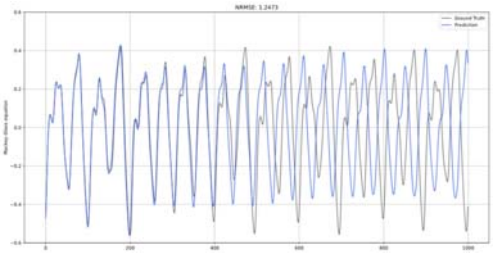
In this section ESN architectures are compared with the RLS-FORCE learning method. The results are presented in table 4 and table 5.

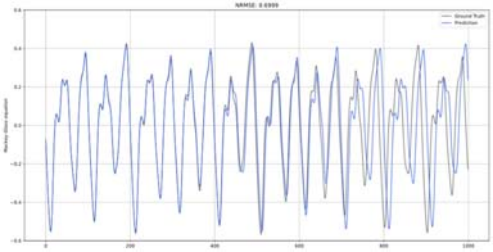
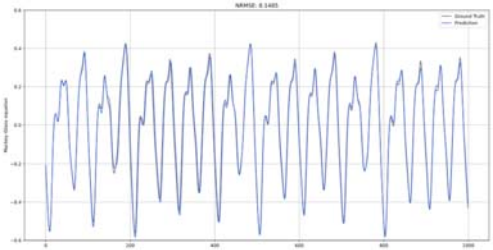
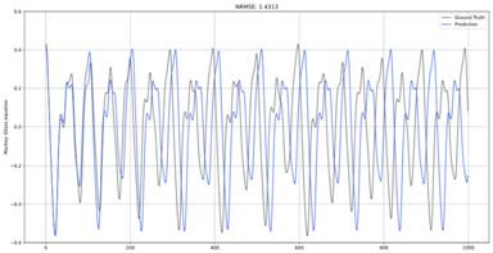
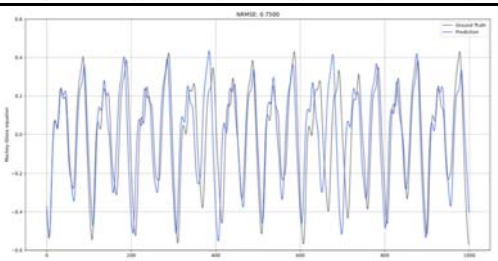
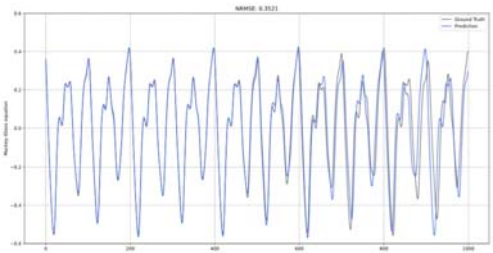
Table 4. Comparison of ESN architectures (W: Worst, M: Mean, S: Std. Dev., B: Best; numbers indicate prediction length). Architectures are sorted by M1000. DNC: Did not converge.

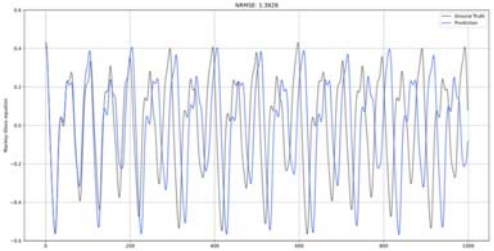
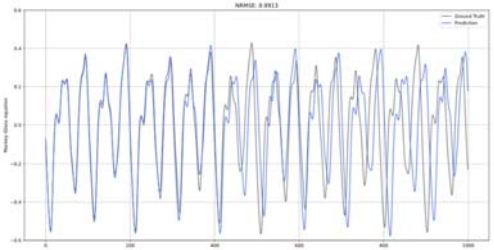
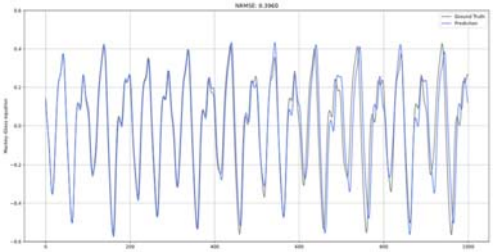
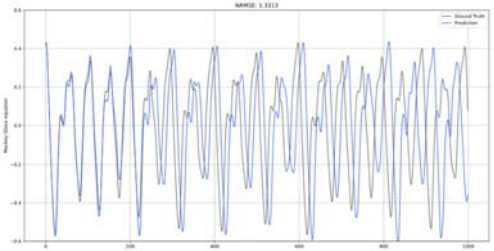
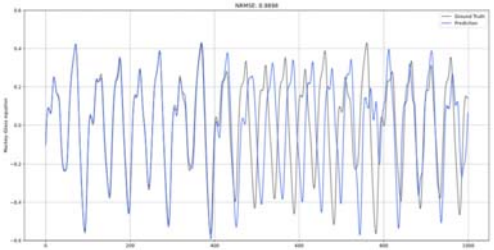
Arch	W1000	M1000	S1000	B1000	W400	M400	S400	B400
ESN (skew)	**0.895**	**0.412**	0.237	**0.013**	**0.353**	**0.066**	0.074	**0.005**
ESN	1.073	0.663	0.197	0.205	0.808	0.273	0.196	0.043
NF (skew)	1.247	0.694	0.266	0.141	0.503	0.192	0.131	0.036
ESN (DE)	1.431	0.766	0.285	0.352	0.984	0.405	0.249	0.034
EuSN	1.383	0.867	0.264	0.396	0.961	0.361	0.229	0.087
Delay Neuron	1.331	0.990	0.273	0.426	0.823	0.361	0.262	0.024
NF (mexhat)	1.748	1.181	0.267	0.561	1.657	1.016	0.301	0.578
R-D System	9.282	1.694	1.807	0.542	7.662	1.345	1.496	0.498
NVAR	DNC	DNC	DNC	DNC	DNC	DNC	DNC	DNC

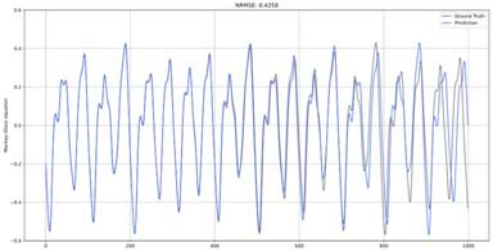
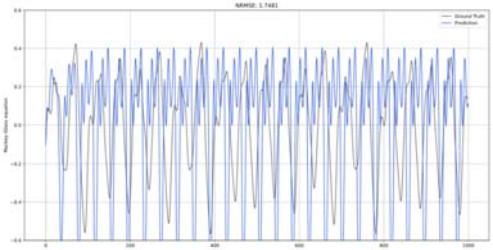
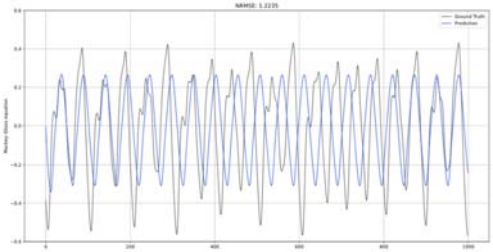
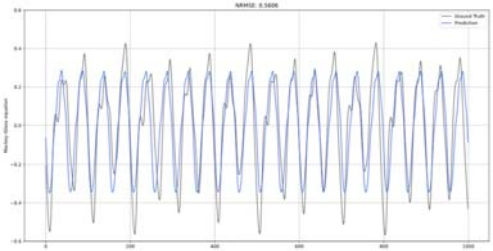
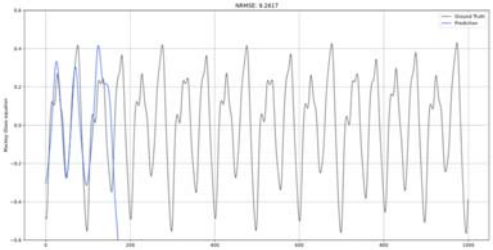
Table 5. Graphical comparison of architectures. The mean graph is chosen as the seed with the NRMSE nearest to the mean.

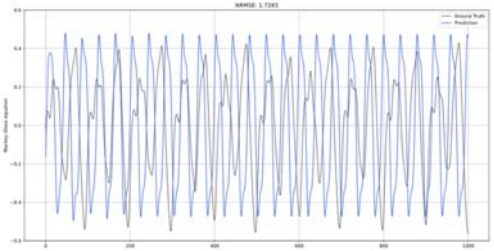
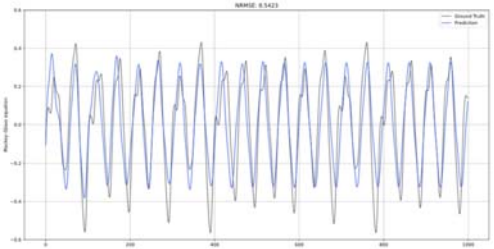
Arch	Type	Graph
ESN (skew)	Worst	
ESN (skew)	Mean	

Arch	Type	Graph
ESN (skew)	Best	
ESN	Worst	
ESN	Mean	
ESN	Best	
NF (skew)	Worst	

Arch	Type	Graph
NF (skew)	Mean	
NF (skew)	Best	
ESN (DE)	Worst	
ESN (DE)	Mean	
ESN (DE)	Best	

Arch	Type	Graph
EuSN	Worst	
EuSN	Mean	
EuSN	Best	
Delay Neuron	Worst	
Delay Neuron	Mean	

Arch	Type	Graph
Delay Neuron	Best	
NF (mexhat)	Worst	
NF (mexhat)	Mean	
NF (mexhat)	Best	
R-D System	Worst	

Arch	Type	Graph
R-D System	Mean	
R-D System	Best	

As shown in table 4, ESN (skew) outperforms all other architectures across all categories. However, NF (skew) is among the top performers and possesses several valuable properties, such as reduced randomness, computational efficiency, easy visualization, scalability, and graph memory.

Simultaneously, we observe that the classical ESN can outperform ESN (DE). We believe that graph memory is the property that helps the classical ESN outperform ESN (DE).

As previously mentioned, NF (mexhat) do not exhibit useful dynamics, which explains these results. The RD model could utilize three components instead of two, which might introduce chaos to the system, which is necessary for non-periodic prediction; however, this would require three times more neurons than the more powerful NF (skew) model. It is possible that these two models could perform well in different tasks, such as classification.

EuSN performs well but is not among the best. Nevertheless, it is interesting that a non-dissipative reservoir can function.

NVAR cannot handle this task, demonstrating its lack of universality; it is more useful as a mathematical model than a practical tool for complex tasks.

The ‘one neuron’ model, however, does work, though it performs well only over shorter time periods. We attribute this to the trivial dynamics of virtual neurons, as noted earlier. Nonetheless, the idea of using delayed feedback may be useful, particularly in scenarios where delay can be introduced naturally via a physical reservoir. Otherwise, delayed feedback is computationally expensive but may assist in tasks requiring a more powerful reservoir than a single neuron.

5.5 Hyperparameter Optimization

In this section, the hyperparameter optimization rule is presented. Moreover, for models that work, actual hyperparameters are presented in table 6. But beware, some values may not work with different reservoir seeds (in our case, all reservoir seeds are the same).

The one neuron model also has the hyperparameter $0 < k \leq 20$, which is an integer. There are two more models that do not work: NF (mexhat), which has hyperparameters $0 < \sigma \leq 10$ and $-3 \leq a \leq 3$; and the R-D model, which has $0 < \omega_\mu \leq 10$, $0 < \omega_\sigma \leq 3$, $-1 \leq \gamma \leq 3$, $0 < d_u \leq 1$, and $0 < d_v \leq 1$.

Also, other training methods have one or both hyperparameters: λ for regularization, ε in TSVD, and η in LMS. λ is $10^0, \dots, 10^9$ or not used, ε is $10^0, \dots 10^{-9}$ or not used, and η is $10^{-4}, \dots 10^0$.

Some of these values can be used as a starting point in different tasks, such as $\lambda \neq 1$, $\delta = 10^4$, $sr_{re} = 1$ in ESN (skew). Others are more or less dependent on the seed and task. Another way to optimize hyperparameters may involve using not one trajectory, but several. However, this requires more epochs and is much more computationally intensive.

Table 6. Values of optimized hyperparameters and ranges of their optimization. Where a non-default range is used, the specific segment is presented.

Arch	p_{in}	p_{rec}	$normal_{in}$	sr	sr_{im}	sr_{re}	α	δ	λ
Range	$[0, 1]$	$[0, 1]$	$[0, 10]$	$[0, 3]$	$[0, 10]$	$[-1, 3]$	$[0, 1]$	$10^{\{0,\dots,4\}}$	$\{1, 0.999\}$
ESN (skew)	0.593	0.850	8.429	-	0.936	0.998	0.527	10^4	0.999
ESN	0.880	0.447	5.507	1.219	-	-	0.673	10^3	0.999
NF (skew)	0.913	-	8.371	-	[0, 20] 2.908	0.321	0.028	10^4	0.999
ESN (DE)	0.312	0.513	6.300	-	2.084	1.640	0.486	10^3	0.999
EuSN	0.274	0.744	8.095	-	[0, 20] 14.494	[-1, 1] -0.736	0.043	10^4	0.999
Delay Neuron	-	-	7.961	-	-	-	0.848	10^4	0.999

6. Conclusion

This paper presents a comparative theoretical analysis of training methods and architectures for Echo State Networks, integrating theoretical examination with experimental evaluation. Our analysis demonstrates that the computational power of a reservoir is deeply tied to its structural properties, such as graph topology, spectral characteristics, and the novel concept of graph memory (GM) introduced herein. For training, online methods like RLS offer a robust and adaptive alternative to offline regression, with RLS-FORCE being particularly effective for generative tasks, as confirmed by its superior convergence and ability to capture chaotic dynamics in our experiments.

Regarding architecture, we argue that the most promising designs are those that generate desynchronized oscillations and incorporate GM. Standard ESNs achieve the former through carefully chosen spectral properties, such as a ‘ring’ spectrum for discrete-time models or a ‘segment’ spectrum for continuous-time models, where the latter is closer to what is desired. Thus, we recommend using the ESN (skew) model. More advanced architectures, like the Neural Field (skew), offer a principled way to achieve scalability and incorporate a natural form of GM through spatial coupling

These findings are confirmed through comprehensive experiments on the Mackey-Glass chaotic time series prediction task, where the ESN (skew) architecture combined with RLS-FORCE achieved the lowest mean NRMSE (0.412 over 1000 steps and 0.066 over 400 steps across 20 seeds), outperforming other architectures and underscoring the practical advantages of skew-symmetric designs and online training methods. At the same time, the Neural Field (skew) architecture demonstrated competitive performance (mean NRMSE of 0.694 over 1000 steps and 0.192 over 400

steps), while offering several additional benefits: inherent GM through spatial coupling, computational efficiency via convolution operations, reduced randomness in reservoir construction, and improved interpretability and visualization capabilities.

The results of this analysis provide a robust framework for developing ESN models. We intend to use this framework to guide the design and selection of ESN networks for practical applications.

Список литературы / References

[1]. Jaeger H. The echo state approach to analysing and training recurrent neural networks. German National Research Institute for Computer Science. GMD-Report 148, 2001. Available at: https://www.researchgate.net/publication/215385037_The_echo_state_approach_to_analysing_and_training_recurrent_neural_networks-with_an_erratum_note', accessed 13.03.2026.

[2]. Maass W., Natschlaeger T., Markram H. Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation, vol. 14, no. 11, pp. 2531-2560, 2002. DOI: 10.1162/089976602760407955.

[3]. Kirby K., Context dynamics in neural sequential learning. In Proceedings of the Florida artificial intelligence research symposium FLAIRS, 1991, pp. 66-70.

[4]. Tanaka G. et al., Recent advances in physical reservoir computing: A review. Neural Networks, vol. 115, Mar. 2019, DOI: 10.1016/j.neunet.2019.03.005.

[5]. Fernando C., Sojakka S. Pattern recognition in a bucket. In Proc. of ECAL, Sep. 2003, pp. 588-597. DOI: 10.1007/978-3-540-39432-7_63.

[6]. Lukoševičius M. A Practical Guide to Applying Echo State Networks, in Neural Networks: Tricks of the Trade. Reloaded, vol. 7700, Montavon G., Orr G. B., Müller K.-R., Eds., in Lecture notes in computer science, vol. 7700, Springer, 2012, pp. 659-686.

[7]. Gallicchio C., Micheli A. Deep echo state network (DeepESN): A brief survey. CoRR, vol. abs/1712.04323, 2017, Available at: http://arxiv.org/abs/1712.04323

[8]. Zhang H., Vargas D. V. A survey on reservoir computing and its interdisciplinary applications beyond traditional machine learning. IEEE Access, vol. 11, pp. 81033-81070, 2023, DOI: 10.1109/access.2023.3299296.

[9]. Hastie T., Tibshirani R., Friedman J. H. The elements of statistical learning: Data mining, inference, and prediction. In Springer series in statistics. Springer, 2001. Available at: https://books.google.ru/books?id=VRzITwgNV2UC, accessed 13.03.2026.

[10]. Engl H. W., Hanke M., Neubauer A. Regularization of inverse problems. Kluwer, 1996.

[11]. Pearson K., On lines and planes of closest fit to systems of points in space. Philosophical Magazine, vol. 2, no. 11, pp. 559-572, 1901, DOI: 10.1080/14786440109462720.

[12]. Haykin S. S. Adaptive filter theory. Pearson, 2014. Available at: https://books.google.co.za/books?id=J4GRKQEACAAJ, accessed 13.03.2026.

[13]. Sussillo D., Abbott L. F. Generating coherent patterns of activity from chaotic neural networks. Neuron, vol. 63, no. 4, pp. 544-557, Aug. 2009, DOI: 10.1016/j.neuron.2009.07.018.

[14]. Steil J. Backpropagation-decorrelation: Online recurrent learning with o(n) complexity. In IEEE International Conference on Neural Networks – Conference Proceedings, Aug. 2004, vol. 2, pp. 843-848. DOI: 10.1109/IJCNN.2004.1380039.

[15]. Jaeger H. Echo state network. Scholarpedia, 2007, vol. 2, no. 9, p. 2330, DOI: 10.4249/scholarpedia.2330.

[16]. Manjunath G., Jaeger H. Echo state property linked to an input: Exploring a fundamental characteristic of recurrent neural networks. Neural Computation, vol. 25, no. 3, pp. 671-696, Mar. 2013, DOI: 10.1162/NECO_a_00411.

[17]. Gallicchio C. Euler State Networks: Non-dissipative Reservoir Computing. arXiv preprint arXiv:2203.09382, 2023, DOI: 10.48550/arXiv.2203.09382.

[18]. Bollt E. On explaining the surprising success of reservoir computing forecaster of chaos? The universal machine learning dynamical system with contrast to VAR and DMD. Chaos, vol. 31, no. 13108, 2021.

[19]. Gauthier D. J., Bollt E., Griffith A. Barbosa W. A. S. Next generation reservoir computing. Nature Communications, vol. 12, p. 5564, 2021, DOI: 10.1038/s41467-021-25801-2.

[20]. Parlitz U. Learning from the past: reservoir computing using delayed variables. Frontiers in Applied Mathematics and Statistics, vol. 10, p. 1221051, Mar. 2024, DOI: 10.3389/fams.2024.1221051.

[21]. Hart J. D., Larger L., Murphy T. E., Roy R. Delayed dynamical systems: networks, chimeras and reservoir computing. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 377, no. 2153, 2019, DOI: 10.1098/rsta.2018.0123.

[22]. Li M. Y., Wei J. Global hopf bifurcation analysis of a neuron network model with time delays. In Infinite dimensional dynamical systems, J. Mallet-Paret, Wu J., Yi Y., Zhu H. (eds.), New York, NY: Springer New York, 2013, pp. 141-168. DOI: 10.1007/978-1-4614-4523-4_5.

[23]. Coombes S., beim Graben P., Potthast R., Wright J. (eds.), Neural fields: Theory and applications, 1st ed. Springer Berlin, Heidelberg, 2014, pp. X, 487. DOI: 10.1007/978-3-642-54593-1.

[24]. Turing A. M. The chemical basis of morphogenesis. Philosophical Transactions of the Royal Society of London. B, Biological Sciences, vol. 237, no. 641, pp. 37-72, Aug. 1952, DOI: 10.1098/rstb.1952.0012.

[25]. Amari S. Homogeneous nets of neuron-like elements. Biological Cybernetics, vol. 17, pp. 211-220, 1975.

[26]. Amari S. Dynamics of pattern formation in lateral-inhibition type neural fields. Biological Cybernetics, vol. 27, pp. 77-87, 1977.

[27]. Nunez P. L. The brain wave equation: A model for the EEG. Mathematical Biosciences, vol. 21, no. 3, pp. 279–297, 1974.

[28]. Cook B. J., Peterson A. D. H., Woldman W., Terry J. R. Neural field models: A mathematical overview and unifying framework. Mathematical Neuroscience and Applications, vol. 2, Mar. 2022, DOI: 10.46298/mna.7284.

[29]. Gorecki J. et al. Chemical computing with reaction–diffusion processes. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 373, no. 20140219, 2015, DOI: 10.1098/rsta.2014.0219.

[30]. Mackey M. C., Glass L. Oscillation and chaos in physiological control systems. Science, vol. 197, no. 4300, pp. 287-289, 1977, DOI: 10.1126/science.267326.

[31]. 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, vol. 148, Jan. 2001.

[32]. Wringe C., Trefzer M., Stepney S. Reservoir computing benchmarks: A tutorial review and critique. International Journal of Parallel, Emergent and Distributed Systems, vol. 40, no. 4, pp. 313-351, Mar. 2025, DOI: 10.1080/17445760.2025.2472211.

[33]. Ozaki Y., Watanabe S., Yanase T. OptunaHub: A platform for black-box optimization. arXiv preprint arXiv:2510.02798, 2025. Available at: https://arxiv.org/pdf/2510.02798, accessed 12.03.2026.

Информация об авторах / Information about authors

Илья Александрович АНДРОСОВ – младший специалист-исследователь отдела перспективных исследований АО «НПК «Криптонит». Сфера научных интересов: резервуарные вычисления, динамические системы и рекуррентные нейронные сети.

Ilya Aleksandrovich ANDROSOV – Junior research specialist in the Division of perspective investigation at JSC "Research and production company 'Kryptonite'". Research interests: reservoir computing, dynamic systems, and recurrent neural networks.