Reservoir Memory Networks: time-series classification with untrained RNNs

Claudio Gallicchio and Andrea Ceni

Department of Computer Science, University of Pisa
Largo B. Pontecorvo, 3 - 56127, Pisa, Italy
claudio.gallicchio@unipi.it, andrea.ceni@di.unipi.it

Abstract. We introduce Reservoir Memory Networks (RMNs), a novel class of Reservoir Computing (RC) models designed to enhance long-term information retention in Recurrent Neural Networks (RNNs) without training the dynamic components. By integrating a linear memory cell with a non-linear reservoir, RMNs efficiently manage long-term dependencies, a traditional challenge in standard RC approaches. Our empirical analysis on time-series classification and pixel-level 1-D classification tasks demonstrate that RMNs not only outperform conventional reservoir models but also exhibit competitive accuracy compared to fully trainable RNNs. These results highlight RMNs' potential as a computationally efficient alternative for handling complex sequential tasks.

Keywords: Reservoir Computing · Echo State Networks · Recurrent Neural Networks · Time-series classification · Long-range propagation

1 Introduction

Reservoir Computing (RC) [7] is a robust approach in the design of Recurrent Neural Networks (RNNs), known for its efficiency and minimal training demands. The core concept involves constructing an RNN architecture where the recurrent layer, the *reservoir*, is initialized with stability constraints and remains untrained, thereby shifting the training focus solely to the readout layer. This approach is particularly relevant in the realms of pervasive artificial intelligence (AI) and neuromorphic hardware implementations [14, 9], where RC facilitates low-power, high-speed processing, aligning with the goals of sustainable AI. Traditional RC architectures, however, often struggle with processing sequences in tasks that require long-range retention of information, a critical capability for many advanced AI applications.

In this paper, we introduce a novel class of RC systems, the *Reservoir Memory Networks* (RMNs), which combine a linear memory cell with a non-linear processing reservoir. This dual-reservoir approach aims to harness the strengths of both linear and non-linear dynamics to efficiently manage long-term dependencies, preserving the efficiency of the RC paradigm. The effectiveness of our proposed method is empirically tested on various time-series classification tasks and pixel-level 1-D classification, demonstrating significantly improved results

compared to state-of-the-art RC techniques and strong competitiveness with fully trainable RNN models.

2 Reservoir networks with linear memory cell

In developing RMNs, we build on the Echo State Network (ESN) formalism, known for its training efficiency in temporal data processing [5, 6]. ESNs, while effective, are limited by their inherent fading memory property, where information dissipates over time due to the asymptotically stable dynamics of the untrained recurrent layer [4]. RMNs overcome this limitation by combining a linear reservoir for sustained memory and a non-linear reservoir for complex processing tasks. This configuration allows RMNs to manage long-term dependencies more effectively than conventional ESNs, which often face a trade-off between memory capacity and non-linear processing [11].

As in standard RC, the architecture of our proposed RMN includes a fixed recurrent component, and a trainable feed-forward readout. The key distinction lies in RMN's dynamical component, which consists of a dual reservoir system: a linear reservoir *memory cell*, and a non-linear reservoir for non-linear processing over time. Conceptually, the reservoir memory cell should explicitly provide the system with a long-range temporal context on the external input driving signal, feeding the operation of the non-linear reservoir, which in turn should focus on the non-linear processing aspects required by the problem at hand. The state of the non-linear reservoir is then used as input to the readout component. The overall architecture of RMN is shown in Figure 1.

Note that our design allows handling the input memorization in isolation from the non-linear processing, thereby avoiding a classic weakness of conventional RC models where memory and non-linear processing are tightly intertwined. Moreover, it allows one to decouple the dimensionality of the non-linear reservoir from the size of the memory cell. We indicate the number of neurons in the linear reservoir by N_m, the number of input features by N_x, and the memory state at time-step t by $\mathbf{m}(t) \in \mathbb{R}^{N_m}$. The memory cell is updated based on the external driving input signal:

$$\mathbf{m}(t) = \mathbf{V_m}\mathbf{m}(t-1) + \mathbf{V_x}\mathbf{x}(t), \tag{1}$$

in which $\mathbf{V_m} \in \mathbb{R}^{N_m \times N_m}$ is a recurrent memory weight matrix, $\mathbf{V_x} \in \mathbb{R}^{N_m \times N_x}$ is an input memory weight matrix, and both of them are left untrained after initialization. The role of $\mathbf{V_m}$ is therefore crucial in determining the memorization abilities of the system. Here we leverage a design strategy based on an orthogonal $\mathbf{V_m}$ matrix, exploiting the optimal short-term memory properties of this type of dynamic neural systems [13, 10]. As a specific instance of an orthogonal weight matrix with fixed weights, we use a circular shift matrix, containing 1s

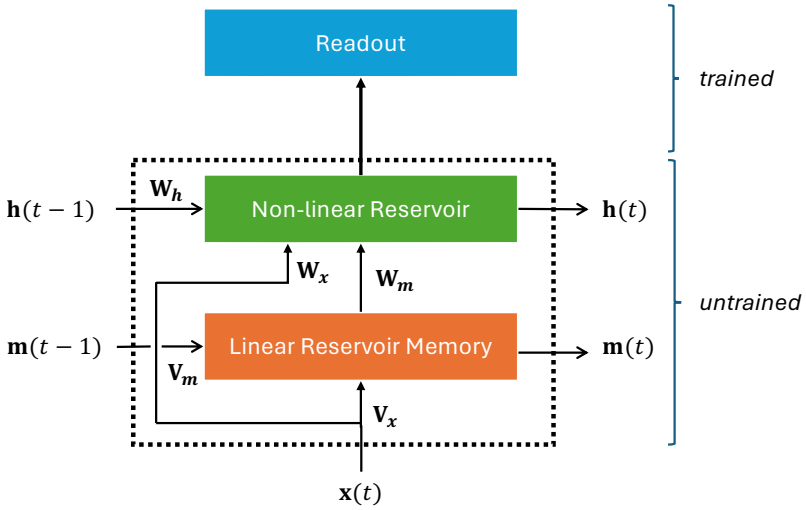

Fig. 1. Time unrolled architecture of a Reservoir Memory Network (RMN). The dynamical component includes two systems: (i) a memory cell implementing a linear reservoir driven by the external input, and (ii) a non-linear reservoir system that is fed by both the external input and the output of the memory cell. The output of the non-linear reservoir component is fed to the readout component, which is the only trained part in the architecture.

on the sub-diagonal and on the top-right element, and 0s elsewhere, i.e.,

$$\mathbf{V_m} = \begin{bmatrix} 0 & 0 & \dots & 1 \\ 1 & 0 & \dots & 0 \\ \vdots & \ddots & \dots & \vdots \\ 0 & \dots & 1 & 0 \end{bmatrix}. \tag{2}$$

Note that this construction leads to an eigenvalue spread around the unitary circle, as shown in Figure 2.

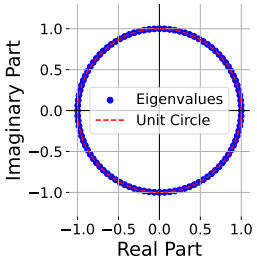

Fig. 2. Eigenvalues of the recurrent memory weight matrix $\mathbf{V_m}$ with circular shift structure (shown for $N_m = 100$).

In analogy to conventional RC, the weights in $\mathbf{V_x}$ are chosen from a uniform distribution modulated by an input memory scaling hyper-parameter ω_{x_m}.

The non-linear reservoir is implemented in a similar fashion to a leaky integrator ESN [6]. We indicate the number of non-linear reservoir units by N_h, and the state by $\mathbf{h}(t) \in \mathbb{R}^{N_h}$. This evolves based on both the external input and the memory cell state, as follows:

$$\mathbf{h}(t) = (1-\alpha)\mathbf{h}(t-1) + \alpha \tanh\left(\mathbf{W}_h\mathbf{h}(t-1) + \mathbf{W_m}\mathbf{m}(t) + \mathbf{W}_x\mathbf{x}(t) + \mathbf{b}_h\right), \quad (3)$$

where $\mathbf{W}_h \in \mathbb{R}^{N_h \times N_h}$ is the recurrent weight matrix, $\mathbf{W}_m \in \mathbb{R}^{(N_h \times N_h}$ is the memory weight matrix, $\mathbf{W_x} \in \mathbb{R}^{N_h \times N_x}$ is the input weight matrix, $\mathbf{b}_h \in \mathbb{R}^{N_h}$ is the bias vector, and $\alpha \in (0, 1]$ indicates the leaking rate hyper-parameter. The reservoir weight matrix is initialized in line with the echo state property [15], and then left untrained. Accordingly, its weights are initialized randomly and then re-scaled to control the resulting spectral radius of $\mathbf{W_h}$, denoted by ρ, which is treated as a hyper-parameter. The weight values in $\mathbf{W_m}$, $\mathbf{W_x}$, and $\mathbf{b}_h$ are initialized randomly from uniform distributions whose extremes are determined, respectively, by the hyper-parameters of memory scaling ω_m, input scaling ω_x and bias scaling ω_b.

As an alternative, we implement the non-linear reservoir as in the Euler State Network (EuSN) [4], a recent RC approach specifically designed to improve information retention by discretizing an ODE under non-dissipative constraints using anti-symmetric recurrent weight matrix. In this case, the reservoir state $\mathbf{h}(t)$ is updated as follows:

$$\mathbf{h}(t) = \mathbf{h}(t-1) + \varepsilon \tanh\left((\mathbf{W}_h - \mathbf{W}_h^T - \gamma\mathbf{I})\mathbf{h}(t-1) + \mathbf{W_m}\mathbf{m}(t) + \mathbf{W}_x\mathbf{x}(t) + \mathbf{b}_h\right), \quad (4)$$

where ε and γ are small positive hyper-parameters that respectively indicate the time-step of integration and the diffusion coefficient, and values in $\mathbf{W_h}$ are initialized from a uniform distribution modulated by recurrent scaling hyper-parameter ω_r. We dub this variant as RMN_{Eu}.

All weight values in eq. 1, 3 and 4 are left fixed after initialization. Training is restricted to a recurrent-free readout, implemented as a simple linear dense layer trained by ridge regression.

3 Experiments

We have experimentally validated the RMN approach on time-series classification tasks and on pixel-level 1-D classification. Details on datasets and experimental settings are given in Appendix A.

3.1 Results on time-series classification.

Test set accuracy at dataset level is given in Table 1, while an aggregated comparison is provided by Figure 3. Results show a substantial advantage of the proposed RMN models over conventional RC networks, both under the same number of overall parameters (subscript p) and the same number of trainable parameters (subscript tp).

Dataset	$\mathbf{ESN}_{\mathrm{p}}$	$\mathbf{ESN}_{\mathrm{tp}}$	$\mathbf{EuSN}_{\mathrm{p}}$	$\mathbf{EuSN}_{\mathrm{tp}}$	**RMN**	$\mathbf{RMN}_{\mathrm{Eu}}$
Beef	0.55 $\pm$0.02	0.53 $\pm$0.00	0.55 $\pm$0.06	0.42 $\pm$0.05	**0.82** $\pm$0.05	**0.82** $\pm$0.05
Car	0.70 $\pm$0.00	0.71 $\pm$0.01	0.81 $\pm$0.01	0.80 $\pm$0.01	**0.86** $\pm$0.02	0.85 $\pm$0.04
Coffee	0.98 $\pm$0.02	0.93 $\pm$0.02	0.91 $\pm$0.04	0.93 $\pm$0.00	0.99 $\pm$0.02	**1.00** $\pm$0.00
DDG	0.44 $\pm$0.06	0.49 $\pm$0.05	0.48 $\pm$0.04	0.48 $\pm$0.03	0.50 $\pm$0.05	**0.56** $\pm$0.05
FordA	0.71 $\pm$0.01	0.70 $\pm$0.01	0.72 $\pm$0.01	0.69 $\pm$0.01	**0.88** $\pm$0.02	0.87 $\pm$0.01
FordB	0.63 $\pm$0.01	0.61 $\pm$0.01	0.63 $\pm$0.01	0.64 $\pm$0.01	**0.74** $\pm$0.01	0.69 $\pm$0.02
OSULeaf	0.60 $\pm$0.01	0.60 $\pm$0.01	0.61 $\pm$0.02	0.63 $\pm$0.01	0.65 $\pm$0.02	**0.67** $\pm$0.02
Meat	0.85 $\pm$0.00	0.85 $\pm$0.01	0.89 $\pm$0.01	0.92 $\pm$0.02	0.93 $\pm$0.01	**0.97** $\pm$0.01
Symbols	0.67 $\pm$0.01	0.80 $\pm$0.02	0.81 $\pm$0.02	0.89 $\pm$0.00	**0.90** $\pm$0.02	0.89 $\pm$0.01
ShapeletSim	0.49 $\pm$0.01	0.52 $\pm$0.01	0.48 $\pm$0.01	0.51 $\pm$0.04	**0.59** $\pm$0.06	0.50 $\pm$0.03
ShapesAll	0.76 $\pm$0.00	0.75 $\pm$0.00	0.80 $\pm$0.00	**0.82** $\pm$0.01	0.81 $\pm$0.01	0.80 $\pm$0.01
Wine	0.46 $\pm$0.00	0.43 $\pm$0.02	0.61 $\pm$0.07	0.71 $\pm$0.03	**0.80** $\pm$0.03	0.64 $\pm$0.09

Table 1. Test set accuracy on times-series classification tasks. Subscript *p*: comparison with the same total maximum number of parameters. Subscript *tp*: comparison with the same number of trainable parameters.

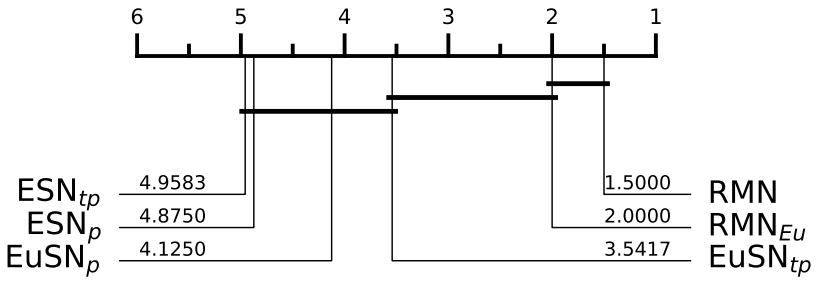

Fig. 3. Critical difference plot summarizing the aggregated scores across all time-series classification datasets.

3.2 Results on pixel-level 1-D classification.

We ran experiments on the permuted sequential MNIST (psMNIST) task at the increase of trainable parameters from $\approx 1k$ to $\approx 150k$. The results, presented in Figure 3.2, clearly demonstrate the efficacy of RMN. At configurations approximating 150k trainable parameters, RMN not only surpassed ESN's best accuracy by over 10% but also outperformed EuSN's peak performance by more than 5%. This advantage extends to smaller configurations with approximately 1k trainable parameters, where RMN improved over ESN by more than 24% and EuSN by over 29%. Furthermore, RMN demonstrated exceptional computational efficiency. For instance, it matched ESN's highest accuracy, achieved with approximately 150k trainable parameters, using merely about 5k, thereby reducing the required trainable parameters by 96.67%. Similarly, against EuSN's best performance at approximately 50k trainable parameters, RMN accomplished com-

parable results with a 90% reduction in parameters. As a side observation, incorporating non-dissipative dynamics within the nonlinear reservoirs of RMN_{Eu} does not yield performance enhancements over the standard RMN configuration. A broader comparison in the landscape of trainable RNNs is reported in Table 2. It is particularly noteworthy that our RMN and RMN_{Eu} models demonstrate competitive, and in some cases superior, performance compared to several fully trainable RNN architectures, despite the dynamical components of our networks remaining untrained. This underscores the effectiveness of our architectural choices and highlights the potential of our models to deliver high accuracy with non-trainable dynamics, challenging the conventional necessity for extensive parameter training in achieving state-of-the-art results.

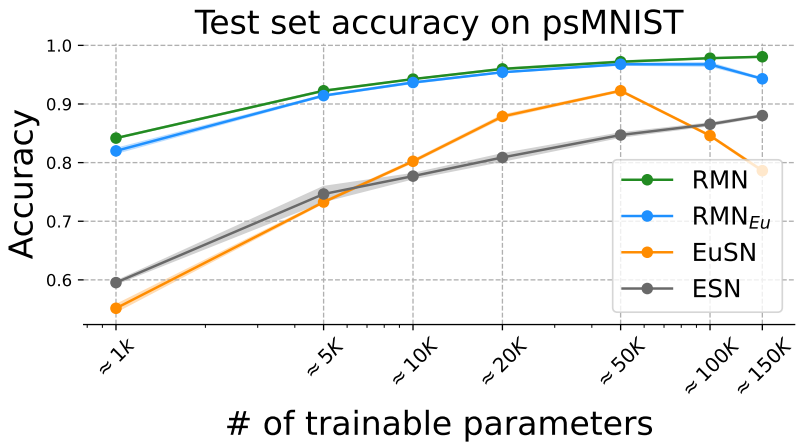

Fig. 4. psMNIST results increasing # of trainable parameters.

4 Conclusions

We have introduced Reservoir Memory Networks (RMNs), a novel integration of a linear memory cell with a nonlinear reservoir, designed to enhance the handling of long-term dependencies in sequence processing tasks. This architecture effectively mitigates the inherent limitations of traditional Reservoir Computing, demonstrating substantial performance enhancements in tasks requiring robust memory management. RMNs also present a viable, efficient alternative to fully trainable RNNs, delivering competitive performance while significantly reducing the training complexity, ideal for computationally constrained environments.

Model	# par.	Acc.
coRNN [8]	$\approx 134k$	97.3%
GRU [12]	$\approx 165k$	92.4%
NRU [1]	$\approx 165k$	95.4%
Lip. RNN [3]	$\approx 34k$	96.3%
LMU [12]	$\approx 102k$	97.2%
LMU-par [2]	$\approx 165k$	98.5%
RNN-orth [12]	$\approx 165k$	89.3%
ESN	$\approx 150k$	88.0%
EuSN	$\approx 50k$	92.3%
RMN_{Eu} *(ours)*	$\approx 50k$	96.8%
RMN *(ours)*	$\approx 150k$	98.1%

Table 2. psMNIST results.

A Details on experiments

Here we report additional information on the datasets and experimental settings used for our experiments described in Section 3.

A.1 Time-series classification

Datasets. We have considered 12 datasets of diverse nature from the UCR archive (from http://timeseriesclassification.com/), namely: Beef, Car, Coffee, DuckDuckGeese (abbreviated DDG), FordA, FordB, OSULeaf, Meat, Symbols, ShapeletSim, ShapesAll, and Wine. For each dataset, we used the original partition into training and test sets.

Experimental settings. We ran experiments with RMN and RMN_{Eu}, using $N_h = 500$ units in the non-linear reservoir component, and exploring configurations with a number of units in the reservoir memory cell N_m equal to $1/3\ T$, $1/2\ T$, T, and $2\ T$, where T indicates the maximum length of a time-series in the training set. The other hyper-parameters were explored in the following ranges: ω_x, ω_{x_m}, ω_m, ω_b, and ω_r in $\{0.01, 0.1, 2, 1, 5\}$, ρ in $\{0.8, 0.9, 1.0\}$, α, ϵ, γ in $\{0.01, 0.1, 1.0\}$. For comparison, we ran the same experiments with ESNs and EuSNs, exploring the corresponding hyper-parameters[1] within the same ranges indicated above for the RMN variants. As regards the non-linear reservoir dimensionality in these latter experiments, we considered two different setups. The first with $N_h = 500$, leading to a comparison under equal conditions of trainable parameters with the RMN variants. This setting is indicated with a subscript tp. A second experimental setup for ESNs and EuSNs involved exploring a number of reservoir units varying within a range such that the total number of internal connections is in the same range as explored for the case of experiments with RMNs. This second setting led to a comparison under equal conditions of a total

[1] For ESN: ω_x, ω_b, ρ, α. For EuSN: ω_x, ω_b, ω_r, ϵ, γ.

number of reservoir parameters with the RMN variants, and it is indicated with a subscript p. In all cases, the readout was applied to the reservoir state computed at the last time-step of each sequence, and was trained by ridge regression with Tikhonov regularization hyper-parameter λ, exploring values in $\{1.0, 0.1, 0.01\}$.

For each model, hyper-parameters were optimized through model selection on a validation set, derived by a further stratified splitting 33% / 67% of the training data, utilizing a random search across 200 trials (or until reaching a maximum of 10 hours of computation for model selection) and averaging results over 3 random guesses. Following model selection, the selected network configuration was trained on the entire training set and then assessed on the test set, with averages and standard deviations calculated from 10 random guesses.

A.2 Pixel-level 1-D classification

Dataset. The permuted sequential MNIST (psMNIST) dataset modifies the traditional MNIST dataset to test sequence-processing architectures like recurrent neural networks (RNNs) on their ability to handle long-term dependencies. Each 28x28 pixel grayscale image from the standard MNIST dataset, representing handwritten digits from 0 to 9, is transformed into a 784-step sequence, with each step representing the normalized intensity of a pixel. A fixed permutation is applied to the pixel order in all images, adding complexity by dispersing critical information throughout the sequence. This setup challenges models to maintain accuracy while processing sequences with significant informational dispersion, mirroring challenges in practical applications.

Experimental settings. We conducted experiments using four models: RMN, RMNEu, ESN, and EuSN. Each model varied in the size of its non-linear reservoir N_h, tested at sizes 100, 500, 1000, 2000, 5000, 10000, 15000, corresponding to between 1k to 150k trainable parameters at the readout level. The linear memory cell in RMN and RMNEu was consistently set at $N_m = 784$ units.

Hyper-parameters, including ω_x, ω_{x_m}, ω_m, ω_b, and ω_r were tested at values $\{0.1, 0.5, 1.0\}$, ρ at $\{0.8, 0.9, 1.0\}$, and α, ϵ, γ at $\{0.01, 0.1, 1\}$. These were optimized using a validation set split 33%/67% from the training data through a random search over 100 trials or up to 10 hours. This model selection was conducted at the smallest non-linear reservoir setting ($N_h = 100$) to ensure robustness; for the baseline EuSN model, optimization occurred at $N_h = 1000$ to ensure stability across scale variations. After model selection, the best hyper-parameter configuration was applied to all considered N_h sizes. Each model was trained on the full training set and evaluated on the test set, with results averaged over 3 runs. The readout was applied to the non-linear reservoir state computed at the last time-step, and trained by ridge regression. The Tikhonov regularization parameter λ was fine-tuned using a nested leave-one-out strategy on the training data, exploring values in $\{10^{-5}, 10^{-4}, \dots 10^{4}\}$.

References

1. Chandar, S., Sankar, C., Vorontsov, E., Kahou, S.E., Bengio, Y.: Towards non-saturating recurrent units for modelling long-term dependencies. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 3280–3287 (2019)
2. Chilkuri, N.R., Eliasmith, C.: Parallelizing legendre memory unit training. In: International Conference on Machine Learning. pp. 1898–1907. PMLR (2021)
3. Erichson, N.B., Azencot, O., Queiruga, A., Hodgkinson, L., Mahoney, M.W.: Lipschitz recurrent neural networks. arXiv preprint arXiv:2006.12070 (2020)
4. Gallicchio, C.: Euler state networks: Non-dissipative reservoir computing. Neurocomputing p. 127411 (2024)
5. Jaeger, H., Haas, H.: Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. science (2004)
6. Jaeger, H., Lukoševičius, M., Popovici, D., Siewert, U.: Optimization and applications of echo state networks with leaky-integrator neurons. Neural networks **20**(3), 335–352 (2007)
7. Nakajima, K., Fischer, I.: Reservoir Computing. Springer (2021)
8. Rusch, T.K., Mishra, S.: Coupled oscillatory recurrent neural network (cornn): An accurate and (gradient) stable architecture for learning long time dependencies. In: International Conference on Learning Representations (ICLR 2021). OpenReview (2021)
9. Tanaka, G., et al.: Recent advances in physical reservoir computing: A review. Neural Networks **115**, 100–123 (2019)
10. Tino, P.: Dynamical systems as temporal feature spaces. Journal of Machine Learning Research **21**(44), 1–42 (2020)
11. Verstraeten, D., Dambre, J., Dutoit, X., Schrauwen, B.: Memory versus nonlinearity in reservoirs. In: The 2010 international joint conference on neural networks (IJCNN). pp. 1–8. IEEE (2010)
12. Voelker, A., Kajić, I., Eliasmith, C.: Legendre memory units: Continuous-time representation in recurrent neural networks. NeurIPS **32** (2019)
13. White, O.L., Lee, D.D., Sompolinsky, H.: Short-term memory in orthogonal neural networks. Physical review letters **92**(14), 148102 (2004)
14. Yan, M., Huang, C., Bienstman, P., Tino, P., Lin, W., Sun, J.: Emerging opportunities and challenges for the future of reservoir computing. Nature Communications **15**(1), 2056 (2024)
15. Yildiz, I.B., Jaeger, H., Kiebel, S.J.: Re-visiting the echo state property. Neural networks **35**, 1–9 (2012)