Neuromorphic reservoir computing

Shirin Panahi,[1] Zheng-Meng Zhai,[1] Mulugeta Haile,[2] and Ying-Cheng Lai[1, 3, a)]
[1)] *School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85287, USA*
[2)] *Army Research Directorate, DEVCOM Army Research Laboratory, 6340 Rodman Road, Aberdeen Proving Ground, MD 21005-5069, USA*
[3)] *Department of Physics, Arizona State University, Tempe, Arizona 85287, USA*

(Dated: 29 May 2025)

Reservoir computing has emerged as a promising machine-learning approach to prediction and control of complex nonlinear dynamical systems, rendering important exploring schemes of physical realization. We articulate two frameworks of physical reservoir computing based on the electrophysiological mechanisms in mammalian neuronal networks. The first emulates sensory-motor coordination triggered by external stimuli, while the second mirrors modulatory inputs that regulate the neural state transitions. Both frameworks utilize a simplified yet dynamically rich, map-based behavioral neural model that preserves the essential neuronal functionalities. Computations conducted with sparse random interconnected networks and uncoupled topologies establish the workings of the proposed frameworks in terms of training, validation, and testing. These findings underline the potential of the proposed frameworks as foundational models for actual physical implementation of reservoir computing.

Recent years have witnessed a growing interest in reservoir computing, a machine-learning architecture that is particularly suited for nonlinear and complex dynamical systems. This is so because of (1) the basic property of any dynamical system: its state naturally evolves in time according to a set of rules that mathematically can be described by differential equations or discrete-time maps, and (2) a reservoir computer can be trained and designed to evolve itself in time as a so-called "closed-loop" system. Since its first articulation in 2001 and successful application in predicting spatiotemporal dynamical systems in 2019, reservoir computing has been demonstrated to have the potential to provide solutions for various challenging problems in nonlinear dynamics. For example, in nonlinear dynamical systems, various bifurcations leading to chaos and system collapse can take place and the bifurcation parameters can change with time. Reliably predicting the occurrence of future bifurcations, especially the critical ones leading to a catastrophic system collapse, based solely on historical data, has been deemed as a significant challenge since there are no data available from the future. Recent advances have indicated that reservoir computing may provide a practically viable solution. In view of the capabilities of reservoir computing, a question is whether it can actually be physically implemented, rendering physical reservoir computing as an active area of research with the goal to realize neural computing based on designing the reservoir network from physical or biological systems. This article introduces two frameworks of neuromorphic reservoir computing based on the electrophysiological mechanisms in mammalian neuronal networks: one based on sensory-motor coordination triggered by external stimuli and the other inspired by modulatory inputs that regulate the neural state transitions. Both frameworks exploit a dynamically rich, map-based behavioral neural model, in which the essential neuronal functionalities are preserved. Computations conducted using using random reservoir networks and uncoupled topology established the workings of the frameworks in terms of training, validation, and testing, suggesting their potential as the foundational models for actual implementation of biophysical reservoir computing.

I. INTRODUCTION

Reservoir computing is a machine-learning paradigm that leverages the natural dynamics of a complex system - typically a complex dynamical network in the reservoir or hidden layer. Specifically, an input vector, typically of low dimension, is projected into the high-dimensional vector in the phase space of the reservoir network, and a straightforward readout is then be trained to produce the desired output vector. The nodes in the reservoir network are neurons and are mutually coupled, generating memory and rendering the neural network recurrent. Unlike traditional recurrent neural networks, in reservoir computing only the output weights are trained, while the internal connections in the reservoir network remain fixed[1–3]. This drastically simplifies the training in terms of computational efficiency, while maintaining the power to accommodate nonlinear and complex dynamics. The original idea was articulated in the early 2000s with machine-learning models such as echo-state

[a)] Electronic mail: Ying-Cheng.Lai@asu.edu

network (ESN)[4,5] and liquid state machine (LSM)[6], and the term "reservoir computing" was later coined to unify these concepts[7]. More specifically, an ESN uses a recurrent analog network of artificial neurons (e.g., with sigmoid/tanh units) satisfying the so-called "echo state property," while an LSM employs a network of spiking neurons. In both cases, it has been demonstrated that an arbitrary, predefined recurrent neural network, such as a randomly connected network with fixed weights, can perform complex temporal computations if a linear readout is properly trained[8]. In recent years, there has been an explosive growth of research in reservoir computing[9–51], on a variety of topics ranging from chaotic time series prediction[13,19,22,31], classification of time-varying or sequential patterns[14,33], control systems and robotics[41,52] and signal processing and filtering[34] to system modeling and identification[35] as well as anticipating critical transitions[29] and tipping[45].

In view of the computational power and capability of reservoir computing, a natural question is whether it can actually be implemented by using some nonlinear physical systems. This leads to an area of research: physical reservoir computing, where the neural computing is realized by using physical or natural systems as the reservoir instead of a simulated network[20]. In principle, any physically implementable nonlinear dynamical system with memory can serve as a computational substrate, from optical lasers to sloshing water[53]. The appeal is twofold: (1) one can exploit the rich dynamics inherent in physical processes to potentially achieve computation with lower energy or hardware cost and (2) studying physical reservoir computing can lead to insights into how natural systems themselves process information.

A brief review of the history of physical reservoir computing is as follows. In a landmark early demonstration, a bucket of water was demonstrated to act as a computing reservoir[54]. By vibrating water with input signals and observing the surface waves with a camera, a "liquid brain" was built, which was exploited for solving pattern recognition tasks. In this framework, the bucket is the reservoir: it takes inputs (stones) and produces a complex pattern (ripples). There is no need to control the exact dynamics inside, but just observe the patterns and learn to interpret them for tasks such as predicting weather or recognizing speech[54]. Throughout the 2010s, physical reservoir computing diversified into many forms, such as optical devices, analog circuits, spintronic and quantum systems[55,56]. These studies demonstrated that a wide range of physical media can serve as reservoirs insofar as they have nonlinearity, high dimensionality, and a fading memory[4,57].

By the late 2010s and early 2020s, the convergence of unconventional computing and biology became more concrete. Notably, biological systems were recognized as natural candidates for reservoir computing, where two directions of research were opened. The first one used living organisms or natural phenomena for reservoir computing. For example, the gene regulatory network of E. coli was explored for a liquid state machine[58], and the recordings from a cat's primary visual cortex were analyzed with the finding that the recurrent neural microcircuit naturally had the properties of reservoir computing[59]. Later, an echo-state network was trained to mimic the monkey prefrontal cortex activity during a cognitive task and it was found that the network's mixed dynamics closely mirrored the biological neural dynamics[60]. Other studies reported that living neurons can be stimulated with inputs and their activity can be used as the reservoir state, although training such systems may be challenging due to their plasticity[61,62]. Another study found that biological neural reservoirs can act as filters, where linear readouts could classify input patterns from the collective spikes[63]. A recent study treated human soft tissue (the viscoelastic muscle and skin of an arm) as a reservoir that processes inputs in a prosthetic control system[64]. The concept of ecological reservoir computing was introduced and implemented[65], providing the first proof-of-concept that an ecological system can perform computation[65]. In 2022, the first experimental demonstration of physical reservoir computing using a living plant - strawberry (Fragaria × ananassa), was reported[66].

The second direction in biologically inspired reservoir computing expands upon the concept in biophysical systems engineered or designed to mimic natural processes, such as how neurons fire in the brain or how muscles contract in animals. For example, a soft robotic arm inspired by an octopus used its flexible movements, similar to how the octopus navigates, to process information[67]. Bio-inspired reservoir computing incorporating neuronal intrinsic plasticity and multi-clustered network structures was studied[68]. Reservoir computing has also been implemented within multicellular populations by leveraging diffusion-based cell-to-cell signaling, a common biological communication method[69]. A soft bio-inspired propulsor serving as a physical reservoir to perform state estimation tasks in autonomous underwater vehicles was developed[70]. These diverse studies collectively aimed to exploit the inherent efficiency and adaptability biological systems for applications in robotics, biomedical engineering, and other related fields.

In this paper, we articulate two frameworks of biophysical reservoir computing based on the electrophysiological mechanisms in mammalian neuronal networks. The first framework is based on sensory-motor coordination triggered by external stimuli, while the second is inspired by modulatory inputs that regulate the neural state transitions. Both frameworks utilize a simplified yet dynamically rich map-based behavioral neural model, in which the essential neuronal functionalities are preserved. We conduct computations using random reservoir networks and uncoupled topology, and establish the workings of the proposed frameworks in terms of training, validation, and testing. The results suggest the potential of the proposed frameworks as possible foundational models for actual implementation of biophysical reservoir computing.

In Sec. II, we provide the background on reservoir com-

puting and an overview of neural dynamics in the human brain. In Sec. III, we introduce the two biophysical reservoir-computing frameworks inspired by neural dynamics and study the computational capability of two distinct reservoir network topologies: one with random nodal connections and the other uncoupled. Numerical results are presented in Sec. IV. Section V summarizes the key findings and offers a comprehensive discussion. A detailed and comprehensive account of the workings of reservoir computing in terms of data preparation, training, validation, hyperparameter optimization and testing can be found in Appendix A.

II. BACKGROUND

A. Reservoir computing

A reservoir computer consists of an input, hidden, and an output layers, as shown in Fig. 1(a). The input layer feeds the time-varying input $\mathbf{u}(t)$ (typically a low-dimensional vector) into the hidden layer. The hidden layer, or the reservoir, hosts a dynamical neural network, typically of a large number of nodes, and all the nodal dynamical variables constitute a state vector, denoted as $\mathbf{r}(t)$, in the corresponding high-dimensional phase space. Mathematically, the mapping from the low-dimensional vector $\mathbf{u}(t)$ to the high-dimensional state vector $\mathbf{r}(t)$ is governed by the input matrix $\mathcal{W}_{\rm in}$:

$$\mathbf{r}(t) = \mathcal{W}_{\rm in} \cdot \mathbf{u}(t),$$

where the elements of $\mathcal{W}_{\rm in}$ are randomly chosen and then fixed throughout the training process. Because of its high dimensionality, the state vector $\mathbf{r}(t)$ implicitly contains the memory of the input vector from the past. The nodes of the reservoir network are nonlinear activation units defined, e.g., by the hyperbolic tangent function. The structure or topology of the reservoir network is described by the matrix $\mathcal{A}$ that is typically asymmetric, weighted and fixed through the training process. The nodal interactions described by $\mathcal{A}$ are thus mutual: there is information flow between any pair of nodes, rendering recurrent the dynamical network. Explicitly, in discrete time, the dynamical evolution of the state vector is governed by

$$\mathbf{r}(t+1) = (1-\alpha)\mathbf{r}(t) + \alpha \mathbf{f}\left[\mathcal{A} \cdot \mathbf{r}(t) + \mathcal{W}_{\rm in} \cdot \mathbf{u}(t)\right], \quad (1)$$

where α is the leakage parameter (one of the hyperparameters) and $\mathbf{f}$ denotes the vector field of all the nodal activation functions. After an update of the state vector from $\mathbf{r}(t)$ to $\mathbf{r}(t+1)$, the readout or output layer computes the vector $\mathbf{y}(t)$ from $\mathbf{r}(t+1)$ (by a simple weighted sum) through the output matrix $\mathcal{W}_{\rm out}$:

$$\mathbf{y}(t) = \mathcal{W}_{\rm out} \cdot \mathbf{r}(t), \quad (2)$$

where the elements of $\mathcal{W}_{\rm out}$ are determined through the training process by linear regression to fit the target output.

Let d be the dimension of the input vector $\mathbf{u}(t)$: $\mathbf{u} \in \mathbb{R}^d$, and let the number of nodes in the reservoir network be N. The inequality $d \ll N$ typically holds. In applications of reservoir computing in nonlinear dynamical systems prediction, it is often the case that the output vector $\mathbf{y}(t)$ has the same dimension as that of the input vector: $\mathbf{y} \in \mathbb{R}^d$. The dimensions of the input matrix $\mathcal{W}_{\rm in}$, the reservoir network matrix $\mathcal{A}$, and the output matrix $\mathcal{W}_{\rm out}$ are thus $N \times d$, $N \times N$, and $d \times N$, respectively. Since the randomly matrices $\mathcal{W}_{\rm in}$ and $\mathcal{A}$ are predefined and fixed, and only the output matrix $\mathcal{W}_{\rm out}$ needs to be determined through training, the number of trainable parameters is dN, which can be determined through standard linear regression. In particular, given the input vector $\mathbf{u}(t)$ at time step t, the dynamical evolution of the reservoir state gives the output vector $\mathbf{y}(t+1)$ at the next time step. The error between $\mathbf{u}(t+1)$ and $\mathbf{y}(t+1)$, accumulated over a large number of time steps, provides the base for determining the elements of the output matrix through linear regression. This makes learning fast and computationally efficient, avoiding the instability issues of training deep recurrent neural networks that rely on backpropagation for training[71]. This principle is analogous to the "random projection" idea in extreme learning machines[72] with the coupling matrix $\mathcal{A} = 0$ (not even self-coupling).

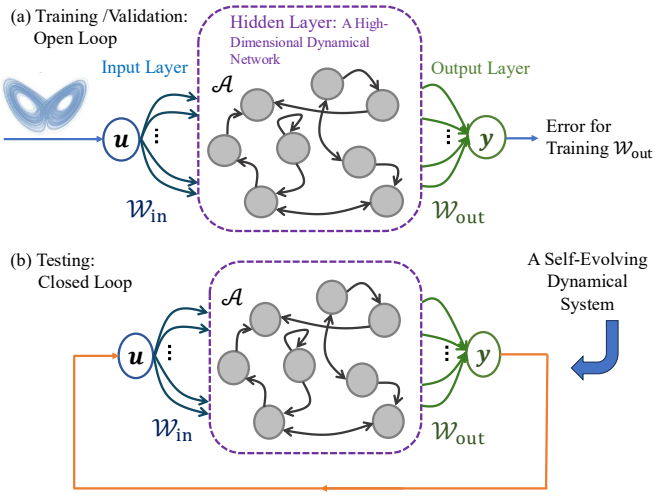

FIG. 1. Basic structure and function of reservoir computing. (a) Open-loop operation for training and validation. During training, the input vector $\mathbf{u}(t)$ is the time series data from the target system, and the reservoir computer generates the output vector $\mathbf{y}(t+1)$. The accumulative error between $\mathbf{y}(t+1)$ and $\mathbf{u}(t+1)$ is used to calculate the elements of the output matrix through linear regression. In the validation phase, the output matrix is fixed and the same error can be used for fine-tuning the hyperparameter values. (b) Closed-loop operation for testing. The output vector connects with the input vector, forming a self-evolved dynamical system.

Following training, a validation dataset (distinct from the training dataset) is used to evaluate the generalization performance of the reservoir computer and to tune the hyperparameter values. Within the open-loop frame-

work as shown in Fig. 1(a), similar to the training phase, the reservoir network is externally driven by a validation input sequence that is not from the training data and the corresponding target outputs are used to evaluate the predictive performance. In particular, given the input vector $\mathbf{u}(t)$ at time t, the reservoir computer generates the output vector $\mathbf{y}(t+1)$ at time $t+1$, and the difference between $\mathbf{y}(t+1)$ and $\mathbf{u}(t+1)$ is used to assess if the training is satisfactory in the sense that the accumulative error, e.g., the root mean square error (RMSE), must be below a predefined small threshold value. If not, more training steps will be needed to update the elements of the output matrix. This process should be repeated until the training is deemed satisfactory. Unlike the training phase, during the validation phase, the output matrix is fixed and not updated.

In the testing phase, the trained reservoir computer is evaluated in a closed-loop (autonomous) configuration, as shown in Fig. 1(b), where the system generates outputs without access to any ground-truth input. Instead, the predicted output is recurrently fed back into the reservoir as the next input, allowing the reservoir computer to evolve autonomously based on its internal dynamics and previous output. This setup is essential for applications involving multi-step-ahead forecasting, digital twin, or autonomous system modeling. The closed-loop structure tests the reservoir computer's ability to sustain accurate prediction over an extended time horizon, maintain internal stability, and reproduce complex dynamics without external corrections. In this phase, the reservoir state vector is typically initialized with a short sequence of real inputs to ensure that the dynamical state of the reservoir network does not depend on the initial condition. This is known as the "listening" stage. From this point forward, the reservoir-computing system evolves in time entirely on its own generated output. The closed-loop performance, or the reservoir-computer's long-term performance can be evaluated using statistical measures such as the Lyapunov exponents[73], deviation value (DV)[74], and Kullback–Leibler (KL) divergence[75] that quantify how well the reservoir computer can reconstruct or emulate the underlying dynamics without external guidance.

B. Electrophysiological mechanisms in neuronal networks

To motivate our articulation of the two biophysically inspired reservoir-computing frameworks, we briefly describe the basics of electrophysiological mechanisms in neuronal networks

Neuronal activities have stimulated research in different fields, specially in computer science where artificial neural networks originated. From the point of view of electrophysiology, neuronal activities are complex but they can be categorized into two types: a fast response mode and a slow modulatory mode[76–79]. In the first mode, rapid electrical signals in the neurons are triggered by an external sensory stimulus (e.g., a touch or a flash of light), which directly drives a chain of excitation and inhibition, leading to an immediate motor action. The second slow mode is more about adjusting and regulating the normal, spontaneous, ongoing patterns of neurons firing with neuromodulatory inputs that are chemicals such as dopamine or serotonin arriving from specialized centers. This mode adjusts the excitability of the neurons without instantly causing new spikes. The two types of neuron activities, one of sensory-motor integration and one of state modulation, illustrate how the nervous system uses different bioelectrophysiological mechanisms to achieve coordinated behavior and adaptive brain states.

1. External stimuli and sensory-motor coordination

Sensory stimulus (pain/heat) caused by an action (touching a hot spot or being exposed to a sudden light) starts a rapid chain of events that shapes preventing reactions. In such a situation, specific neural populations (e.g., touch receptors or retinal cells) are directly activated with external stimuli, and the activity propagates to other neurons in connected pathways through excitatory synapses (often using glutamate)[78]. However, to prevent over-excitability that spreads too broadly or intensely, inhibitory feedback is activated where interneurons using inhibitory transmitters (e.g., GABA) quickly fire in response to the excitation and suppress excessive activity in other neurons. These ionotropic actions create neural circuit dynamics with feedforward excitation and feedback inhibition that result in a balanced and controlled coherent output[80]. For instance, experiments show that sensory inputs often evoke a "center excitation, surround inhibition" pattern in motor areas, which yields smooth and targeted motor coordination instead of chaotic contractions.

A hypothesis to model such complex neural behavior is considering them as threshold-based mechanism, such that each neuron is in a resting state and, after reaching a level of depolarization (an activation threshold), it fires an action potential[78]. The external stimuli needs to be powerful enough to reach this threshold and make the neuron spike, and further increases in input produce higher firing rates until approaching a saturation point. Otherwise, the neuron stays silent and acts as minor noise filtering.

2. Modulatory inputs for state transitions

A neural circuit has other stimuli than immediate sensory inputs (the ionotropic mechanism), which contains firing rhythmically during normal activity on its own, e.g., the slow oscillations of deep sleep or the complex firing patterns of quiet wakefulness)[78]. These normal activities also have external stimuli that influence the overall behavior later in the form of neuromodulators, which are chemical signals (e.g., dopamine, serotonin,

norepinephrine, acetylcholine, etc.)[81]. These inputs come after the neurons are already active, and they do not initiate the activity from scratch. Instead, they alter the dynamics of the activities, which is different from the direct, fast action of excitatory/inhibitory transmitters. By this mechanism, after a neuromodulator (input) reaches a neuron, it typically binds to metabotropic receptors (G-protein-coupled receptors) on the neuron's membrane and sets off a cascade of intracellular signals, leading to chemical modifications inside the cell[82]. Biophysically, neuromodulators act as a regulator that alter the activity of the neuron by tweaking its ion channels. For instance, closing a type of potassium channel makes the cell less leaky (so it holds a depolarization longer), or opening a K^+ channel makes it more hyperpolarized (requiring more excitation to fire). While this process does not have a direct effect at the moment, it tunes the neuron's excitability or responsiveness.

III. BIO-INSPIRED RESERVOIR COMPUTING

Inspired by the two basic electrophysiological mechanisms observed in neuronal networks as described in Sec. II B, we propose two distinct reservoir-computing architectures with the aim to enhance both the interpretability and implementability of physical reservoir computing. For simplicity and basic physical understanding, we utilize a discrete-time, map-based behavioral neural model, neglecting non-essential biophysical details while preserving the core functional dynamics of the neuronal activities. Such behavioral models have in fact attracted considerable attention over the past decade, recognized as effective and efficient alternatives to overly detailed biological neuron models[83]. This attention is underpinned by the fundamental principle that increased complexity and biophysical detail in neuronal modeling do not necessarily correlate with improved predictive performance.

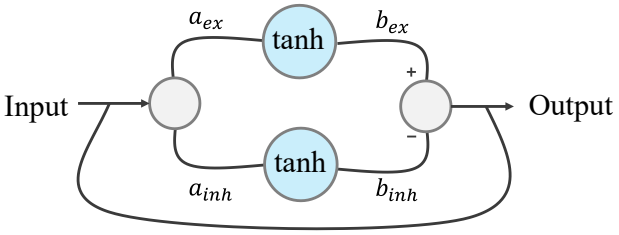

FIG. 2. Behavioral neural network model inspired by brain dynamics, incorporating both excitatory and inhibitory interactions. The network features a feedforward path with a hyperbolic tangent (tanh) activation function and a feedback loop to balance the neural activities. The parameters $a_{\rm ex}$, $a_{\rm inh}$, $b_{\rm ex}$, and $b_{\rm inh}$ are set to mimic the interplay between inhibitory and excitatory interactions in neuronal circuits.

We exploit the behavioral neural model originally introduced for encapsulating the essential neuronal dynamics through the interplay of feedforward and feedback pathways, mirroring key aspects of neuronal circuits[84]. As illustrated in Fig. 2, input signals propagate along a feedforward pathway consisting of both excitatory and inhibitory interactions, each mathematically modeled in terms of hyperbolic tangent activation functions. The balance between excitation and inhibition, critical for maintaining balance and stability of neuronal dynamics, is modeled via a recurrent feedback mechanism embedded within the neural-network architecture. The evolution of the neural state can be formulated as

$$\mathbf{x}(t+1) = b_{\rm ex}\tanh\left[a_{\rm ex}\mathbf{x}(t)\right] - b_{\rm inh}\tanh\left[a_{\rm inh}\mathbf{x}(t)\right], \quad (3)$$

where $a_{\rm ex}$, $a_{\rm inh}$, $b_{\rm ex}$, and $b_{\rm inh}$ are the inhibitory and excitatory factors associated with the brain synapse weights regulated by the release of different neurotransmitters.

A. Reservoir computing inspired by external stimuli and sensory-motor coordination

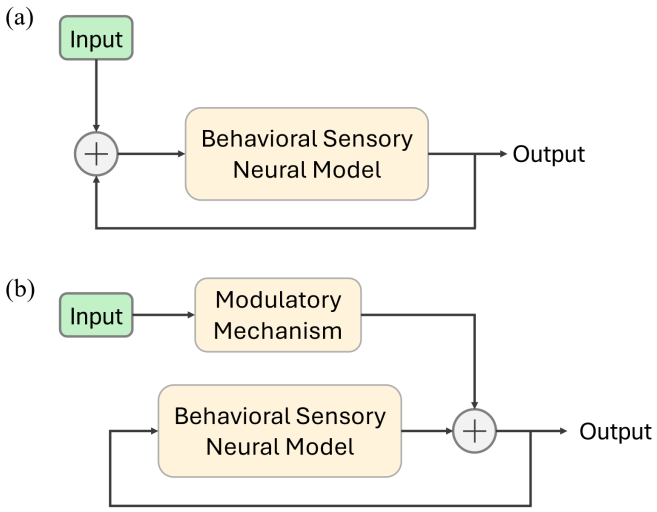

FIG. 3. Two proposed bio-inspired reservoir-computing architectures. (a) Reservoir computing inspired by sensory-motor coordination in mammals, where external stimuli directly drive the reservoir's internal states prior to the nonlinear activation, mimicking fast sensory processing. (b) Reservoir computing inspired by neuromodulatory mechanisms in the brain, where external inputs act after the activation function to modulate ongoing reservoir-network dynamics, reflecting slow, regulatory influence typical of neuromodulators such as dopamine or serotonin.

Here we propose a reservoir-computing framework inspired by the dynamics of cortical neural assemblies responsible for sensory-motor coordination in mammals, as schematically illustrated in Fig. 3(a). This reservoir-computing scheme consists of interconnected behavioral sensory neural models (3) as shown in Fig. 2. Each behavioral model captures aggregate neuronal activities, simplifying the detailed physiological dynamics while

preserving the essential behavior of neuronal populations. External sensory stimuli ($\mathbf{u}(t) \in \mathbb{R}^d$) representing environmental inputs such as tactile or visual signals directly modulate the reservoir network's internal state. The dynamical evolution of the reservoir state is governed by the following discrete-time equation:

$$\begin{aligned}\mathbf{r}(t+1) =& b_{\text{ex}} \tanh\left[a_{\text{ex}}\mathcal{A}\cdot\mathbf{r}(t) + \mathcal{W}_{\text{in}}\cdot\mathbf{u}(t)\right] \\ &- b_{\text{inh}} \tanh\left[a_{\text{inh}}\mathcal{A}\cdot\mathbf{x}(t) + \mathcal{W}_{\text{in}}\cdot\mathbf{u}(t)\right],\end{aligned} \tag{4}$$

where $\mathbf{r} \in \mathbb{R}^N$ is the high-dimensional reservoir state vector, $\mathcal{A} \in \mathbb{R}^{N\times N}$ is the internal recurrent coupling matrix that captures the connectivity within the reservoir network, and $\mathcal{W}_{\text{in}} \in \mathbb{R}^{N\times d}$ is the input weight matrix that projects external sensory inputs into the reservoir's state space. Biologically, this framework mimics the cortical processing pathways wherein sensory inputs propagate through excitatory connections, activating targeted neuronal sub-populations. Concurrently, inhibitory pathways regulate the neuronal excitability, thus preventing hyperactivity and ensuring balanced dynamics. The nonlinear hyperbolic tangent activation functions ensure the neural activity remains within biological constraints, saturating at high stimulus intensities. Consequently, the balanced excitatory-inhibitory recurrent network architecture generates coherent neural activity patterns that underpin coordinated motor responses.

B. Reservoir computing inspired by modulatory inputs and behavioral state transitions

In a complementary scenario illustrated in Fig. 3(c), we propose a bio-inspired reservoir-computing structure composed of interconnected behavioral neural models described by Eq. (3), where the external inputs ($\mathbf{u}(t) \in \mathbb{R}^d$) influence the neural network after the nonlinear activation stage. In contrast to the reservoir-computing scheme described in Sec. III A, here the input signal acts primarily as a neuromodulatory mechanism rather than a direct sensory driver, akin to the effects of biological neuromodulators such as dopamine or serotonin. These neuromodulators typically alter neural excitability and responsiveness, rather than directly triggering action potentials. The discrete-time dynamic evolution of the reservoir states mimicking this scenario is governed by

$$\begin{aligned}\mathbf{r}(t+1) = & b_{\text{ex}} \tanh\left[a_{\text{ex}}\mathcal{A}\cdot\mathbf{r}(t)\right] - b_{\text{inh}} \tanh\left[a_{\text{inh}}\mathcal{A}\cdot\mathbf{x}(t)\right] \\ &+ \mathbf{g}\left[\mathcal{W}_{\text{in}}\cdot\mathbf{u}(t)\right],\end{aligned} \tag{5}$$

where $\mathbf{r} \in \mathbb{R}^N$, $\mathcal{A} \in \mathbb{R}^{N\times N}$, and $\mathcal{W}_{\text{in}} \in \mathbb{R}^{N\times d}$ describe the same characteristics of reservoir computing as described in Sec. III A. In Eq. (III B), the nonlinear function $\mathbf{g}(\cdot)$ captures the intrinsic nonlinear characteristics of the neuromodulatory signaling pathways, modeling effects such as saturation, optimal dosing, and threshold-dependent modulation of neuronal activities. Under this arrangement, the neural ensemble inherently generates spontaneous and recurrent activity patterns that reflect baseline cognitive or behavioral states. The modulatory input adjusts the amplitude or threshold of these spontaneously emerging patterns, facilitating transitions between distinct neural states or behavioral regimes. For example, an appropriate modulatory signal can shift the network from a resting state into an alert or attentive state, thereby increasing sensitivities to subtle environmental cues or enhancing cognitive functions.

IV. RESULTS

To evaluate the performance of the proposed bio-inspired RC frameworks, we employ chaotic time series data generated by the Lorenz system - a benchmark for evaluating the predictive capabilities of reservoir computing:

$$\begin{aligned}\dot{x}_1 &= \sigma(x_2 - x_1), \\ \dot{x}_2 &= x_1(\rho - x_3) - x_2, \\ \dot{x}_3 &= x_1 x_2 - \beta x_3,\end{aligned} \tag{6}$$

where the parameters are set as $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. Numerical integration is performed using the fourth-order Runge-Kutta (RK4) method with the integration step size of $h = 10^{-3}$. The three state variables (x_1, x_2, x_3) are sampled at the discrete time interval of $\Delta t = 0.1$ and normalized to fit within the range $[0, 1]$.

Our experimental evaluation is structured into three distinct phases: training, validation, and testing. The training and validation phases constitute an open-loop configuration, during which the reservoir computer is provided with external input data to optimize and assess its predictive accuracy, as shown in Fig. 1(a). The testing phase adopts a closed-loop configuration, where the reservoir computer operates autonomously by feeding its output back as input, enabling the generation of self-sustained prediction, as shown in Fig. 1(b). A detailed description of the evaluation method is provided in Appendix.

A. Interconnected bio-inspired reservoir computing

We present the results obtained from two bio-inspired reservoir-computing structures, each configured as a network of interconnected nodes. Each node in the reservoir network represents an aggregate neural unit, whose interactions are defined by the coupling matrix $\mathcal{A}$. The parameter values in Eq. (3) are set to $b_{\text{ex}} = 1.5$, $a_{\text{ex}} = 0.2$, $b_{\text{inh}} = 1$, and $a_{\text{inh}} = 0.1$.

To construct the reservoir network, we set the connectivity structure to be an undirected (symmetric) random network. The elements of the network coupling matrix with N number of interconnected nodes ($\mathcal{A} \in \mathbb{R}^{N\times N}$) are independently drawn from a normal distribution with zero mean. Additionally, we fix the elements of the input

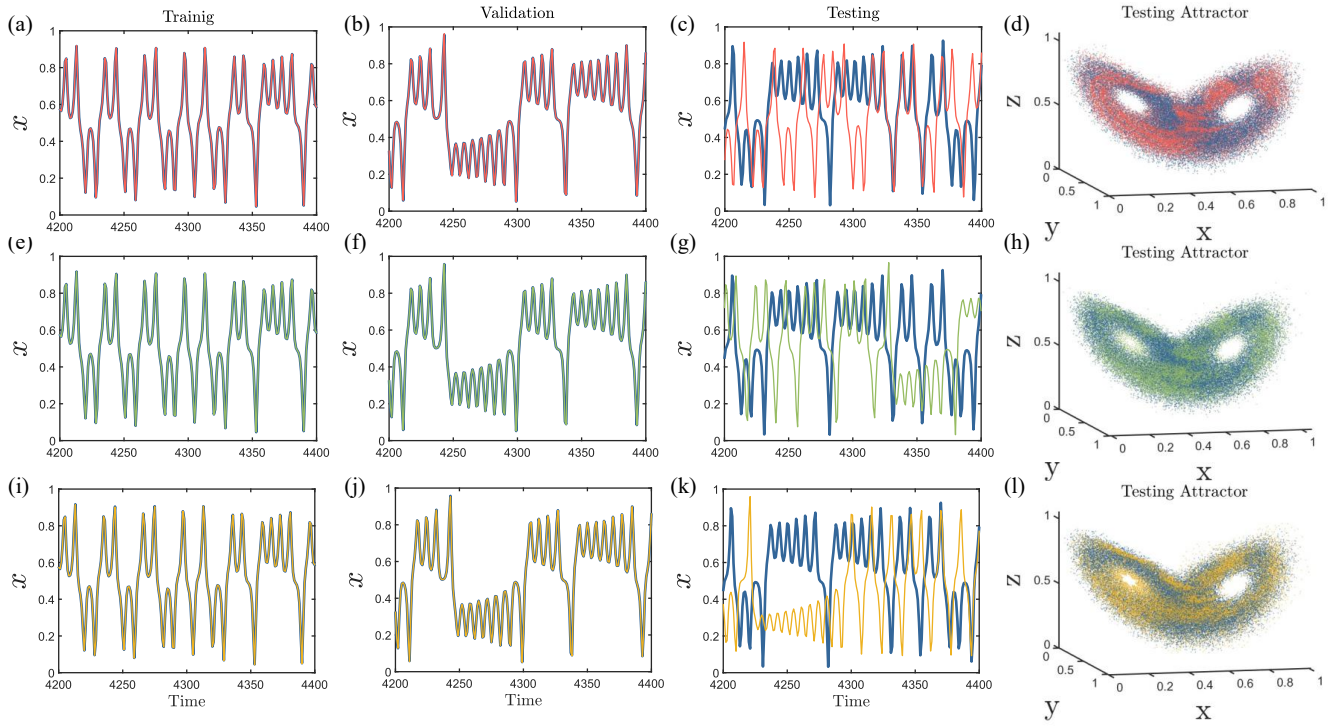

FIG. 4. Performance of the coupled bio-inspired reservoir computers with 30 coupled nodes. (a-d) Results from the first reservoir computing scheme driven by an external sensory input before activation (RC 1). Results from the second reservoir computing scheme are shown in (e-h) with $g(x) = \tanh(x)$ (RC 2) and in (i-l) with $g(x) = 0.5e^{-x^2}$ (RC 3), where modulatory input enters after the activation stage. The left two columns show the training and validation results (open-loop), while the right two columns illustrate autonomous testing results (closed-loop), including long-term trajectory reconstruction.

TABLE I. Statistical analysis comparison between the attractor generated by the bio-inspired coupled reservoir computers and the ground truth

System \ Index	Training RMSE	Validation RMSE	Testing DV	Testing LLE	Testing KL divergence
Bio-inspired RC 1	0.005	0.005	0.157	0.057	1.14×10^{-4}
Bio-inspired RC 2	0.003	0.003	0.165	0.056	7.67×10^{-4}
Bio-inspired RC 3	0.003	0.003	0.159	0.056	7.88×10^{-5}

weight matrix $\mathcal{W}_{\text{in}} \in \mathbb{R}^{N\times d}$ to be uniformly distributed within the range $[-1, 1]$. Once the coupled bio-inspired RC system is constructed, its performance can be evaluated. Figure 4 illustrates the predictive performance of all interconnected bio-inspired reservoir-computing configurations across training, validation, and autonomous testing phases.

Figures 4(a-d) show results from the first bio-inspired reservoir-computing scheme, where external sensory inputs directly drive the neural activation stages. Figures 4(e-h) present results form the second configuration with $g(x) = \tanh(x)$, where modulatory inputs affects the neural dynamics after activation. Results from the same configuration but with $g(x) = 0.5e^{-x^2}$ are shown in Figs. 4(i-l). In all cases, the hidden-layer reservoir network has 30 interconnected nodes. In comparison with conventional reservoir computing[9–20,22–35,37,40,41,45], this network size is small, but the results in Figs. 4(a-l) demonstrate that the scheme is capable of effectively capturing and reproducing the dynamics of the Lorenz chaotic system.

To quantitatively evaluate the performance of the bio-inspired reservoir computing, we employ the NRMSE during the training and validation phases. For the autonomous testing phase, we assess the long-term prediction fidelity using statistical measures including the largest Lyapunov exponent (LLE)[73], deviation value (DV)[74], and Kullback–Leibler (KL) divergence[75]. The results are summarized in Tab. I, demonstrating that the proposed bio-inspired reservoir-computing schemes are fully capable of long-term statistical prediction of chaotic systems by generating the correct chaotic Lorenz attractor. In particular, the results Tab. I indicate that both bio-inspired reservoir-computing architectures exhibit low training and validation errors, suggesting effective learning with little or no overfitting. Notably,

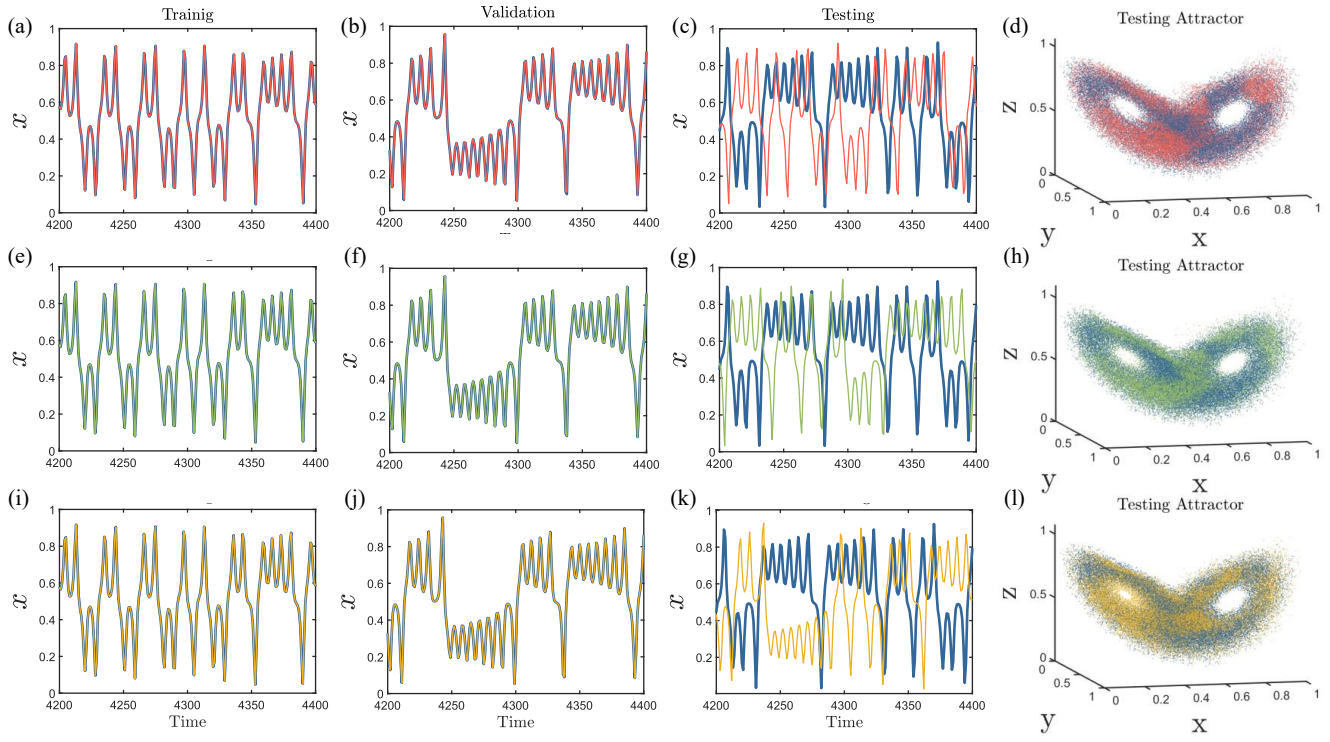

FIG. 5. Performance of the uncoupled bio-inspired reservoir computers with 30 independent nodes. Legends are the same as those in Fig. 4.

TABLE II. Statistical analysis comparison between the attractor generated by the bio-inspired uncoupled reservoir computers and the ground truth

Index / System	Training RMSE	Validation RMSE	Testing DV	Testing LLE	Testing KL divergence
Bio-inspired RC 1	0.005	0.005	0.144	0.056	8.4×10^{-5}
Bio-inspired RC 2	0.004	0.004	0.132	0.057	1.21×10^{-5}
Bio-inspired RC 3	0.003	0.003	0.153	0.056	3.75×10^{-4}

the LLEs estimated from the autonomous testing phase closely match the ground truth value for the Lorenz attractor (about 0.058). This alignment demonstrates that both architectures successfully reproduce the characteristic chaotic behavior of the original system in the closed-loop operation, generating trajectories that remain on the same attractor. These findings are further supported by the small values observed for the DV and the KL divergence, reinforcing the reservoir computer's ability to maintain the statistical and dynamical structure of the Lorenz system over long-term autonomous prediction.

B. Uncoupled bio-inspired reservoir-computing architecture

While the coupled bio-inspired reservoir-computing architectures discussed in Sec. IV A exhibit predictive capabilities, hardware realization of the complex coupling structure could be a challenge. More specifically, the architectures require a precisely controlled network of interconnected nodes, which poses considerable difficulties in terms of scalability, hardware complexity, and long-term stability. To address these limitations, we explore an alternative that is potentially more hardware-implementable: uncoupled reservoir computing[85]. In this framework, each node in the reservoir network operates independently of the others: there are no recurrent connections among the nodes. Mathematically, this entails setting the coupling matrix to the identity matrix: $\mathcal{A} = \mathcal{I}_N$ so that each node is only influenced by its own past state. Each node acts as a stand-alone nonlinear dynamical unit driven by the same input stream.

To enable a fair comparison with the coupled architectures, we use the same network size $N = 30$. Both bio-inspired models are implemented on this uncoupled configuration and simulations for training, validation, and testing phases are carried out. Figures 5(a-d) show the results from the first configuration. Figures 5(e-h) and 5(i-l) present results from the second configuration with

$g(x) = \tanh(x)$ and $g(x) = 0.5e^{-x^2}$, respectively. In all cases, the uncoupled reservoir computer is capable of accurately learning and reproducing the dynamics of the Lorenz chaotic system. Results of quantitative evaluation, including the NRMSE for the training and validation phases and DV, LLE, and the KL divergence for the testing phase are summarized in Tab. II. It can be seen that all the configurations exhibit low NRMSE values (below 0.005), indicating effective short-term learning without overfitting. The reconstructed trajectories remain on the Lorenz attractor for an extended time period. The estimated values of the LLE are remarkably close to the ground-truth value, confirming faithful reproduction of the chaotic dynamics. These findings suggest that the uncoupled bio-inspired reservoir-computing framework is capable of not only matching the predictive performance of the coupled counterpart but also offering substantial practical advantages with respect to physical implementation. By eliminating the need for node-to-node coupling, the uncoupled reservoir-computing scheme permits more efficient, scalable, and physically realizable implementation for neuromorphic and analog computing platforms.

V. DISCUSSION

In recent years, biological systems have emerged as powerful sources of inspiration for unconventional computing. While prior efforts exploited living organisms directly as computational substrates, our approach belongs to designing engineered physical systems that imitate fundamental biological principles. In this paper, we distill essential electrophysiological motifs from neuronal behavior and translate them into discrete-time reservoir-computing dynamics capable of efficient and robust prediction tasks. In particular, we draw from two distinct modes of neuronal function: (1) fast sensory-driven activity, in which external stimuli directly trigger cascades of excitation and inhibition and (2) slow neuromodulatory regulation, where ongoing spontaneous dynamics are adjusted in response to modulatory inputs such as dopamine or serotonin. These models provide insights into biological information processing between direct reactive behavior and context-sensitive modulation and serve as the foundation for our two complementary designs of bio-inspired reservoir computing.

Our first reservoir-computing structure is inspired by sensory-motor integration, where external input is applied prior to the activation function, which models scenarios where sensory inputs immediately trigger neuronal firing and the following responses. Our second reservoir-computing scheme captures the modulatory paradigm by introducing the external input after the activation stage, thereby altering the ongoing reservoir network dynamics rather than initiating them. The role of neuromodulators is further emphasized by incorporating nonlinear transformations of the modulatory input, using either a saturating hyperbolic tangent or a Gaussian-like function to model the biologically plausible dose-response characteristics of such signals. Our experimental evaluations conducted using time-series data from the benchmark chaotic Lorenz system tested both reservoir-computing architectures across training, validation, and autonomous testing phases. The results consistently demonstrate that the proposed schemes are capable of learning and reproducing the complex dynamics of the paradigmatic chaotic system. In particular, during training and validation, low NRMSE values indicate strong short-term predictive accuracy and generalizability in an open-loop configuration. In the closed-loop testing phase where the reservoir computer operates without external input, the autonomous trajectories generated remain bounded within the ground-truth Lorenz attractor for an extended time span. This is confirmed through multiple statistical indices such as LLE, DV and KL divergence. For example, the agreement between the predicted and ground truth LLE values is strong indication that the dynamical system structure has been faithfully captured by the bio-inspired reservoir computers, not merely approximated. These results suggest that all the proposed bio-inspired reservoir-computing schemes possess the core properties required of physical reservoir computing: nonlinearity, fading memory, and insensitivity to initial conditions, alongside the ability to support autonomous, self-sustained output generation - a critical requirement for applications such as tipping point prediction, robotics, and real-time feedback control.

A possible extension of our work involves the comparison between the coupled and uncoupled architectures. While the coupled reservoir-computing systems demonstrate strong predictive performance, their hardware implementation is challenging due to the requirement of maintaining precisely synchronized and densely interconnected nodes. To overcome the limitations, we tested uncoupled reservoir computing - a simplified architecture in which each reservoir node operates independently. By setting the internal coupling matrix to the identity, we effectively eliminated inter-node communication, simplifying both design and physical realization. Remarkably, our results show that uncoupled reservoir computing can match the performance of their coupled counterparts in every metric in that the uncoupled scheme succeeds in learning the Lorenz dynamics and maintaining autonomous trajectories over a long duration, while requiring significantly less structural complexity. This key result reinforces the idea that the essential computational capability of reservoir computing needs not rely on complex connectivity. Instead, it can be achieved through a collection of nonlinear responses in a parallel structure with different inputs. The implications for hardware design encouraging: uncoupled reservoir computing may pave the way for more scalable, energy-efficient, and fault-tolerant physical implementations, particularly in neuromorphic and analog computing platforms.

Taken together, our proposed bio-inspired reservoir-

computing schemes represent a conceptual and practical stepping stone toward physical implementation. Bio-inspired reservoir computing is more than just effective predictors of chaotic dynamics - they are prototypes of physical computation devices that combine the adaptability of neuronal systems with the efficiency and reproducibility of engineered hardware. The proposed bio-inspired reservoir frameworks grounded in fast and slow neural signaling dynamics have the potential of serving not only as computational tools but also as conceptual foundations for the future of physical reservoir computing.

ACKNOWLEDGMENTS

This work was supported by the US Army Research Office under Grant No. W911NF-24-2-0228 and by the Air Force Office of Scientific Research under Grant No. FA9550-21-1-0438.

Appendix A: Reservoir computing: preparation, training, validation, hyperparameter optimization, and testing

1. Input data preprocessing

Any modeling task begins with preparing the input data. Time series data often require preprocessing to ensure the reservoir computer can effectively learn the underlying dynamics of the target system. A commonly used method is normalization, by which the time series data are scaled or normalized so that the input and output lie in a convenient range, e.g., $[0, 1]$. For example, one might apply min-max or z-score normalization. In particular, given a scalar time series $X(t)$ with minimum $X_{\min}$ and maximum $X_{\max}$, min-max normalization lead to the normalized time series $\bar{X}(t)$ given by

$$\bar{X}(t) \equiv \frac{X(t) - X_{\min}}{X_{\max} - X_{\min}}.$$

For z-score normalization, the given time series $X(t)$ is modified to

$$Z(t) \equiv \frac{X(t) - \mu}{\sigma},$$

where μ and σ are the mean and standard deviation of $X(t)$, respectively. The normalization prevents possible outliers in the input from driving the dynamical state of the reservoir network into saturation of the nodal activation function in the reservoir network (e.g., ± 1 output if the activation function is tanh). The saturation region should be avoided because it will diminish memory and lead to unpredictable behavior. Keeping the data bounded and approximately zero-centered helps the reservoir computer operate in its active regime.

For training and validation (essentially one-step-ahead prediction tasks), the reservoir network's input at discrete time t will be the time series value at time t (without any delays) and the target output is the value at time $t + 1$. In the simplest case, the reservoir computer will be trained as an autoregressive model that consumes the previous output to predict the next one. In this cases, the data can be formatted as sequences of input-output pairs $[\mathbf{u}(t), \mathbf{y}_{\text{target}}(t)]$ at consecutive time steps.

2. Initialization of reservoir network

To prepare a reservoir network, the values of some structural parameters need to be initialized, such as the reservoir-network size N (the number of neurons in the network). Intuitively, a larger reservoir network can capture more complex dynamics but the computational cost will be high. For d-dimensional input, the weights or elements of the input matrix $\mathcal{W}_{in} \in \mathbb{R}^{N\times d}$ are randomly initialized with entries drawn from a uniform or the normal distribution. The range of the input weights is often controlled by a small input scaling factor β_{in}. For instance, in case of tanh as an activation function, a smaller input scale keeps the nodes in the reservoir network in the linear region centered at zero, while a larger scale can push them into saturation (± 1) more frequently, enhancing nonlinearity. The reservoir network coupling matrix $\mathcal{A} \in \mathbb{R}^{N\times N}$ is often chosen as a sparse random network. After $\mathcal{A}$ is initialized, the desired spectral radius $\rho(\mathcal{A})$, the largest absolute eigenvalue of $\mathcal{A}$, is adjusted to ensures the reservoir computer's behavior is stable and dependent on input history rather than on initial conditions. The so-called leaking-rate parameter α ($0 \le \alpha \le 1$) controls the speed at which the reservoir network updates its state. In particular, for $\alpha = 1$, the state of the reservoir network is updated fully with the new input influence at each step. This corresponds to the case of "no leak." A small value of α effectively "slows down" the dynamics of the reservoir network in the sense of partial state update, meaning that the reservoir network changes more slowly and retains some of its previous state. The initial condition of the state vector of the reservoir network, $\mathbf{r}(0)$, is usually set to zero or near-zero random values. To remove any effect of $\mathbf{r}(0)$ on the performance, a listening/washout period can be used to remove any transient dynamical evolution from the initial condition.

3. Training

The goal of training is to determine the reservoir output matrix $\mathcal{W}_{\text{out}}$ from the input data. Depending on the specific type of reservoir-network dynamics, for breaking the symmetries in the reservoir state and improving the predictive power, the reservoir state matrix should be slightly augmented or changed.

The input time series is fed into the reservoir network one step at a time. At each time step t, the reservoir state $\mathbf{r}(t)$ is updated according to Eq. (1). In a one-step-ahead prediction task, the input $\mathbf{u}(t)$ is the actual values of the time series at time t (the correct value "forced" by the "teacher") and the reservoir computer is expected to produce an output approximating the time series values at the next time step: $\mathbf{y}(t) = \mathbf{u}(t+1)$. As a result, the reservoir computer is driven by the ground-truth sequence and transforms this input history into a trajectory of the high-dimensional state vector in the reservoir network.

To remove the effect of random reservoir initialization, a sequence of length $T_{\text{transient}} + T_{\text{train}} + T_{\text{valid}}$ is fed into the reservoir network. The initial portion of the state vector

$$\mathbf{r}(1), \mathbf{r}(2), \ldots, \mathbf{r}(T_{\text{transient}})$$

during which the reservoir is being "warmed up", is discarded as a listening/washout period to ensure that the influence of the initial state will fade out.

For convenience, after the transient listening/washout period, the time is reset to zero. The remaining reservoir state vector and the corresponding vector of the target system are utilized for training and validation. Let the time lengths of the training and validation phases be T_{train} and T_{valid}, respectively. For the training and validation phases, the time index thus runs from one to $T_{\text{train}} + T_{\text{valid}}$:

$$t : \underbrace{1, 2, \cdots, T_{\text{train}}}_{\text{Training}}, \quad \underbrace{T_{\text{train}}+1, T_{\text{train}}+2, \cdots, T_{\text{train}}+T_{\text{valid}}}_{\text{Validation}}. \tag{A1}$$

To describe the procedure, it is useful to concatenate the reservoir state vector $\mathbf{r}(t)$ horizontally in terms of the time index to obtain an $N \times T_{\text{train}}$ dimensional training state matrix:

$$\mathcal{R}_{\text{train}} \equiv \big(\mathbf{r}(1), \mathbf{r}(2), \ldots, \mathbf{r}(T_{\text{train}})\big)_{N \times T_{\text{train}}} \tag{A2}$$

Since the N-dimensional state vector $\mathbf{r}(t)$ is generated by the hyperbolic tangent, odd activation function of the reservoir node via the iterative process governed by Eq. (1), the map $\mathbf{r}(t) \to -\mathbf{r}(t)$ leaves the reservoir equations invariant. For zero-mean inputs, the network can lock onto an inverted copy of the true attractor, leading to large prediction errors due to the "mirror-attractor." In addition, if the input amplitude is small and around origin, the reservoir network remains in the near-linear region of the hyperbolic tangent activation function, so the state matrix $\mathcal{R}_{\text{train}}$ can become rank-deficient, which could present difficulties for determining the output matrix $\mathcal{W}_{\text{out}}$ via Eq. (2) and making the least-squares solution for the read-out weights ill-conditioned. There are different ways to overcome these difficulties. One method[13] was to take the square of all even elements of the reservoir state vector $\mathbf{r}(t)$ for all the time indices specified in (A1). In this case, the dimension of the reservoir state matrix $\hat{\mathcal{R}}_{\text{train}}$ is unchanged. Another method is to augment the reservoir states with their quadratic (and higher) terms, e.g.,

$$\hat{\mathbf{r}}(t) = [\mathbf{r}(t), \mathbf{r}^2(t)]^t, \tag{A3}$$

where the dimension of the reservoir state matrix $\hat{\mathcal{R}}_{\text{train}}$ is changed to $\mathbb{R}^{2N \times T_{\text{train}}}$.

An alternative low-cost approach to addressing the "mirror-attractor" problem is designating an additional "bias" node in the reservoir network that injects a small, time-independent driving signal into the reservoir dynamics. The additional bias has no connection with any other node in the network, whose state variable is a constant b, e.g., $b = 1$. Specifically, by this approach, the dimension of the state vector becomes $(N+1)$ and the corresponding state matrix has the dimension $(N+1) \times T_{\text{train}}$. Let the state vector $\mathbf{r}(t)$ in terms of its component be written as

$$\mathbf{r}(t) = (r_1(t), r_2(t), \ldots, r_N(t))^T.$$

Explicitly, the augmented reservoir state matrix for training can be written as

$$\hat{\mathcal{R}}_{\text{train}} = \begin{pmatrix} r_1(1) & r_1(2) & \cdots & r_1(T_{\text{train}}) \\ r_2(1) & r_2(2) & \cdots & r_2(T_{\text{train}}) \\ \vdots & \vdots & \ldots & \vdots \\ r_N(1) & r_N(2) & \cdots & r_N(T_{\text{train}}) \\ 1 & 1 & \cdots & 1 \end{pmatrix} \tag{A4}$$

where the notation $\hat{\mathcal{R}}_{\text{train}}$ is used to distinguish it from the original matrix $\mathcal{R}_{\text{train}}$ and the dimension of $\hat{\mathcal{R}}_{\text{train}}$ is $(N+1) \times T_{\text{train}}$. As a result of this state matrix augmentation, the output matrix $\mathcal{W}_{\text{out}}$ needs to be augmented to the dimension $d \times (N+1)$ to keep the dimension of the output vector $\mathbf{y}(t)$ unchanged, which can be denoted as $\hat{\mathcal{W}}_{\text{out}}$, whose dimension is $d \times (N+1)$.

Let $\hat{\mathcal{Y}}_{\text{train}}$ be the matrix concatenating all the output vectors from the initial time to time T_{train}:

$$\hat{\mathcal{Y}}_{\text{train}} \equiv \big(\mathbf{y}(1), \mathbf{y}(2), \ldots, \mathbf{y}(T_{\text{train}})\big), \tag{A5}$$

whose dimension is $d \times T_{\text{train}}$. Based on Eq. (2), $\mathcal{Y}_{\text{train}}$ can be expressed in terms of the augmented reservoir state matrix $\hat{\mathcal{R}}_{\text{train}}$ and the augmented output matrix $\hat{\mathcal{W}}_{\text{out}}$ as

$$\hat{\mathcal{Y}}_{\text{train}} = \hat{\mathcal{W}}_{\text{out}} \cdot \hat{\mathcal{R}}_{\text{train}}. \tag{A6}$$

Let $\mathcal{Y}_{\rm train}$ be the matrix concatenating all the ground-truth output vectors that are essentially the corresponding input vectors for training. The augmented output matrix $\hat{\mathcal{W}}_{\rm out}$ can be calculated through linear regression of Eq. (A6) with the following loss function:

$$\mathcal{L} = \|\mathcal{Y}_{\rm train} - \hat{\mathcal{W}}_{\rm out} \cdot \hat{\mathcal{R}}_{\rm train}\|^2 + \beta \|\hat{\mathcal{W}}_{\rm out}\|^2, \qquad \text{(A7)}$$

where $\|\hat{\mathcal{W}}_{\rm out}\|^2$ is the sum of squared elements of $\hat{\mathcal{W}}_{\rm out}$ and $\lambda > 0$ is the Tikhonov regularization parameter introduced for preventing overfitting by imposing a penalty on large values of the fitting parameters. The standard linear regression leads to

$$\hat{\mathcal{W}}_{\rm out} = \mathcal{Y}_{\rm train} \cdot \hat{\mathcal{R}}^T_{\rm train} \cdot \left(\hat{\mathcal{R}}_{\rm train} \cdot \hat{\mathcal{R}}^T_{\rm train} + \lambda \mathcal{I}\right)^{-1}, \quad \text{(A8)}$$

where $\mathcal{I}$ is the $(N+1) \times (N+1)$ dimensional identity matrix.

4. Validation and hyperparameter optimization

A machine-learning architecture often has a small number of "global" or hyperparameters. In reservoir computing, the hyperparameters include the input scaling factor β_{in}, the size N and the spectral radius $\rho(A)$ of the reservoir network, the leaking rate α, and the regularization parameter λ. Unlike the parameters determined by training, e.g., all elements of the output matrix $\hat{\mathcal{W}}_{\rm out}$, the hyperparameters are those that define the overall machine-learning architecture and are initially set prior to the training process. The hyperparameter values can significantly impact the performance of the machine-learning model. To achieve the best possible testing performance and to minimize the probability of overfitting, after training is done, it is necessary to optimize the hyperparameters through fine tuning.

For reservoir computing, after training is done in the sense that the optimal output matrix $\hat{\mathcal{W}}_{\rm out}$ has been found, the process of optimizing the hyperparameter values can begin, which is crucial for complex tasks. For this purpose, a validation dataset can be used through evaluating the performance of the trained reservoir computer on unseen data - a kind of generalization test. This dataset thus should have no overlap with the training data, which can be conveniently chosen to be a specifically allocated portion of the available time series, often a data segment following the training dataset. Different methods can be used for fine-tuning the hyperparameter values, such as random search and Bayesian optimization. Specifically, random search is a method in which the hyperparameter values are selected randomly within the search domain. Suppose there are six hyperparameters that need to be tuned, each with many possible values. Testing every combination (known as grid search) would require high computation cost. Random search selects a combination of random values for each hyperparameter from its respective pool at each time step and test the reservoir-computing performance, which is more efficient than the brute-force type of grid search. Alternatively, the fine-tuning of the hyperparameters can be done through different optimization methods such as Bayesian[18], surrogate[86], or genetic algorithm[87]. We use the Bayesian method, which is general and efficient for optimizing complex functions. It uses a probabilistic model to guide the search for optimal hyperparameter values, balancing exploration and exploitation.

Concretely, the state matrix of the reservoir network produced by the validation dataset, denoted as $\hat{\mathcal{R}}_{\rm valid}$, is given by

$$\hat{\mathcal{R}}_{\rm valid} = \begin{pmatrix} r_1(T_{\rm train}+1) & r_1(T_{\rm train}+2) & \cdots & r_1(T_{\rm train}+T_{\rm valid}) \\ r_2(T_{\rm train}+1) & r_2(T_{\rm train}+2) & \cdots & r_2(T_{\rm train}+T_{\rm valid}) \\ \vdots & \vdots & \ldots & \vdots \\ r_N(T_{\rm train}+1) & r_N(T_{\rm train}+2) & \cdots & r_N(T_{\rm train}+T_{\rm valid}) \\ 1 & 1 & \cdots & 1 \end{pmatrix} \qquad \text{(A9)}$$

The predicted validation output with the reservoir output matrix determined during the training phase can be written as

$$\hat{\mathcal{Y}}_{\rm valid} = \hat{\mathcal{W}}_{\rm out} \cdot \hat{\mathcal{R}}_{\rm valid}, \qquad \text{(A10)}$$

which can be compared with the corresponding ground-truth matrix $\mathcal{Y}_{\rm valid}$ be the matrix concatenating all the ground-truth output vectors that are the corresponding input vectors for validation. To measure the one-step-ahead prediction error for hyperparameter tuning, an error metric such as the normalized root mean square error (NRMSE) over the training and validation datasets can be used

$$\text{NRMSE} = \frac{\sqrt{\langle \|\hat{\mathcal{Y}}_{\rm valid} - \mathcal{Y}_{\rm valid}\|^2 \rangle}}{\sigma_y}, \qquad \text{(A11)}$$

where σ_y is the standard deviation of the ground-truth output vector $\mathbf{y}$ and $\langle \cdot \rangle$ denotes the average over a number of statistical realizations of validation. The training and validation processes are repeated until the optimal hyperparameters are identified, with the validation error falling below a predefined threshold.

5. Testing

After training and validation, the reservoir computer can be tested for multi-step prediction performance. During testing, the output of the reservoir computer is connected to its input: $\mathbf{u}(t) = \mathbf{y}(t)$, leading to a closed-loop operation and making the machine a self-evolving, autonomous dynamical system without any external input. The dynamical evolution of the state of the reservoir network is now governed by

$$\mathbf{r}(t+1) = (1-\alpha)\mathbf{r}(t) + \alpha\mathbf{f}\left[\mathcal{A}\cdot\mathbf{r}(t) + \mathcal{W}_{\text{out}}\cdot\mathbf{r}(t-1)\right]. \tag{A12}$$

This is effectively an echo-state network with the state at time $t+1$ depending on the states at time t and $t-1$. Let the asymptotic state of the target system on some kind of invariant set, e.g., a chaotic attractor, be $\mathbf{y}_{\text{target}}(t)$, which represents the ground truth. Setting the initial condition of the reservoir computer as $\mathbf{y}(0) = \mathbf{y}_{\text{target}}(0)$, one can compare the output vector $\mathbf{y}(t)$ of the reservoir computer with the ground truth $\mathbf{y}_{\text{target}}(t)$ for an arbitrarily long time interval. For short-term prediction, the root-mean square difference between the two vectors can be used to characterize the performance. If the target system is chaotic, the output vector $\mathbf{y}(t)$ will diverge exponentially from the ground-truth vector $\mathbf{y}_{\text{target}}(t)$ with the rate determined by the largest Lyapunov exponent of the target system. After a few Lyapunov times, the distance between $\mathbf{y}(t)$ and $\mathbf{y}_{\text{target}}(t)$ can be as large as the range of either vector. The long-term prediction performance can be evaluated by examining whether the reservoir-computer generated attractor is in the neighborhood of the ground-truth attractor. The statistical measures that can be used for this purpose include the Lyapunov exponents[73], deviation value (DV)[74], and Kullback–Leibler (KL) divergence[75].

[1] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, "An overview of reservoir computing: Theory, applications and implementations," in *Proceedings of the 15th European Symposium on Artificial Neural Networks. p. 471-482 2007* (2007) pp. 471–482.

[2] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," Comput. Sci. Rev. **3**, 127–149 (2009).

[3] K. Nakajima and I. Fischer, *Reservoir Computing*, edited by R. A. Meyers (Springer Singapore, Singapore, 2021).

[4] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," GMD **148**, 13 (2001).

[5] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," Science **304**, 78–80 (2004).

[6] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," Neural Comput. **14**, 2531–2560 (2002).

[7] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, "An experimental unification of reservoir computing methods," Neural Netw. **20**, 391–403 (2007).

[8] T. L. Carroll and L. M. Pecora, "Network structure effects in reservoir computers," Chaos **29**, 083130 (2019).

[9] N. D. Haynes, M. C. Soriano, D. P. Rosin, I. Fischer, and D. J. Gauthier, "Reservoir computing with a single time-delay autonomous Boolean node," Phys. Rev. E **91**, 020801 (2015).

[10] L. Larger, A. Baylón-Fuentes, R. Martinenghi, V. S. Udaltsov, Y. K. Chembo, and M. Jacquot, "High-speed photonic reservoir computing using a time-delay-based architecture: Million words per second classification," Phys. Rev. X **7**, 011015 (2017).

[11] J. Pathak, Z. Lu, B. Hunt, M. Girvan, and E. Ott, "Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data," Chaos **27**, 121102 (2017).

[12] Z. Lu, J. Pathak, B. Hunt, M. Girvan, R. Brockett, and E. Ott, "Reservoir observers: Model-free inference of unmeasured variables in chaotic systems," Chaos **27**, 041102 (2017).

[13] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, "Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach," Phys. Rev. Lett. **120**, 024102 (2018).

[14] A. Jalalvand, K. Demuynck, W. De Neve, and J.-P. Martens, "On the application of reservoir computing networks for noisy image recognition," Neurocomputing **277**, 237–248 (2018).

[15] T. L. Carroll, "Using reservoir computers to distinguish chaotic signals," Phys. Rev. E **98**, 052209 (2018).

[16] K. Nakai and Y. Saiki, "Machine-learning inference of fluid variables from data using reservoir computing," Phys. Rev. E **98**, 023111 (2018).

[17] R. S. Zimmermann and U. Parlitz, "Observing spatio-temporal dynamics of excitable media using reservoir computing," Chaos **28**, 043118 (2018).

[18] A. Griffith, A. Pomerance, and D. J. Gauthier, "Forecasting chaotic systems with very low connectivity reservoir computers," Chaos **29**, 123108 (2019).

[19] J. Jiang and Y.-C. Lai, "Model-free prediction of spatiotemporal dynamical systems with recurrent neural networks: Role of network spectral radius," Phys. Rev. Res. **1**, 033056 (2019).

[20] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," Neu. Net. **115**, 100–123 (2019).

[21] T. Weng, H. Yang, C. Gu, J. Zhang, and M. Small, "Synchronization of chaotic systems and their machine-learning models," Phys. Rev. E **99**, 042203 (2019).

[22] P. Chen, R. Liu, K. Aihara, and L. Chen, "Autoreservoir computing for multistep ahead prediction based on the spatiotemporal information transformation," Nat. Commun. **11**, 4568 (2020).

[23] H. Fan, J. Jiang, C. Zhang, X. Wang, and Y.-C. Lai, "Long-term prediction of chaotic systems with machine learning," Phys. Rev. Res. **2**, 012080 (2020).

[24] C. Zhang, J. Jiang, S.-X. Qu, and Y.-C. Lai, "Predicting phase and sensing phase coherence in chaotic systems with machine learning," Chaos **30**, 083114 (2020).

[25] C. Klos, Y. F. K. Kossio, S. Goedeke, A. Gilra, and R.-M. Memmesheimer, "Dynamical learning of dynamics," Phys. Rev. Lett. **125**, 088103 (2020).

[26] D. Patel, D. Canaday, M. Girvan, A. Pomerance, and E. Ott, "Using machine learning to predict statistical properties of non-stationary dynamical processes: System climate, regime transitions, and the effect of stochasticity," Chaos **31**, 033149 (2021).

[27] J. Z. Kim, Z. Lu, E. Nozari, G. J. Pappas, and D. S. Bassett, "Teaching recurrent neural networks to infer global temporal structure from local examples," Nat. Machine Intell. **3**, 316–323 (2021).

[28] H. Fan, L.-W. Kong, Y.-C. Lai, and X. Wang, "Anticipating synchronization with machine learning," Phys. Rev. Res. **3**, 023237 (2021).

[29] L.-W. Kong, H.-W. Fan, C. Grebogi, and Y.-C. Lai, "Machine learning prediction of critical transition and system collapse," Phys. Rev. Res. **3**, 013090 (2021).

[30] L.-W. Kong, H.-W. Fan, C. Grebogi, and Y.-C. Lai, "Emergence of transient chaos and intermittency in machine learning," J. Phys. Complex. **2**, 035014 (2021).

[31]E. Bollt, "On explaining the surprising success of reservoir computing forecaster of chaos? The universal machine learning dynamical system with contrast to VAR and DMD," Chaos **31**, 013108 (2021).
[32]D. J. Gauthier, E. Bollt, A. Griffith, and W. A. Barbosa, "Next generation reservoir computing," Nat. Commun. **12**, 1–8 (2021).
[33]F. M. Bianchi, S. Scardapane, S. Løkse, and R. Jenssen, "Reservoir computing approaches for representation and classification of multivariate time series," IEEE Trans. Neural Netw. Learn. Syst. **32**, 2169–2179 (2021).
[34]Y. Zhong, J. Tang, X. Li, B. Gao, H. Qian, and H. Wu, "Dynamic memristor-based reservoir computing for high-efficiency temporal signal processing," Nat. Commun. **12**, 408 (2021).
[35]D. Canaday, A. Pomerance, and D. J. Gauthier, "Model-free control of dynamical systems with deep reservoir computing," J. Phys. Complex. **2**, 035025 (2021).
[36]L. Jaurigue, E. Robertson, J. Wolters, and K. Lüdge, "Reservoir computing with delayed input for fast and easy optimisation," Entropy **23**, 1560 (2021).
[37]T. L. Carroll, "Optimizing memory in reservoir computers," Chaos **32**, 023123 (2022).
[38]L. Jaurigue and K. Lüdge, "Connecting reservoir computing with statistical forecasting and deep neural networks," Nat. Commun. **13**, 227 (2022).
[39]T. Jüngling, T. Lymburn, and M. Small, "Consistency hierarchy of reservoir computers," IEEE Trans. Neu. Net. Learn. Sys. **33**, 2586–2595 (2022).
[40]L.-W. Kong, Y. Weng, B. Glaz, M. Haile, and Y.-C. Lai, "Reservoir computing as digital twins for nonlinear dynamical systems," Chaos **33**, 033111 (2023).
[41]Z.-M. Zhai, M. Moradi, L.-W. Kong, B. Glaz, M. Haile, and Y.-C. Lai, "Model-free tracking control of complex dynamical trajectories with machine learning," Nat. Commun. **14**, 5698 (2023).
[42]A. Flynn, V. A. Tsachouridis, and A. Amann, "Seeing double with a multifunctional reservoir computer," Chaos **33**, 113115 (2023).
[43]F. Köster, D. Patel, A. Wikner, L. Jarigue, and K. Lüdge, "Data-informed reservoir computing for efficient time-series prediction," Chaos **33**, 073109 (2023).
[44]X. Li, M. Small, and Y. Lei, "Reservoir computing with higher-order interactive coupled pendulums," Phys. Rev. E **108**, 064304 (2023).
[45]S. Panahi, L.-W. Kong, M. Moradi, Z.-M. Zhai, B. Glaz, M. Haile, and Y.-C. Lai, "Machine learning prediction of tipping in complex dynamical systems," Phys. Rev. Res. **6**, 043194 (2024).
[46]A. Flynn and A. Amann, "Exploring the origins of switching dynamics in a multifunctional reservoir computer," Front. Netw. Physiol. **4**, 1451812 (2024).
[47]L. Jaurigue and K. Lüdge, "Reducing reservoir computer hyperparameter dependence by external timescale tailoring," Neuromorph. Comput. Eng. **4**, 014001 (2024).
[48]U. Parlitz, "Learning from the past: reservoir computing using delayed variables," Front. Appl. Math. Stat. **10**, 1221051 (2024).
[49]L. Fleddermann, S. Herzog, and U. Parlitz, "Enhancing reservoir predictions of chaotic time series by incorporating delayed values of input and reservoir variables," Chaos **35**, 053147 (2025).
[50]A. Mizzi, M. Small, and D. M. Walker, "Reservoir computing with the minimum description length principle," Chaos **35**, 043132 (2025).
[51]B. J. Thorne, D. C. Correa, A. Zaitouny, M. Small, and T. Jüngling, "Reservoir computing approaches to unsupervised concept drift detection in dynamical systems," Chaos **35**, 023136 (2025).
[52]P. Bhovad and S. Li, "Physical reservoir computing with origami and its application to robotic crawling," Sci. Rep. **11**, 13002 (2021).
[53]K. Nakajima, "Physical reservoir computing—an introductory perspective," Jpn. J. Appl. Phys. **59**, 060501 (2020).
[54]C. Fernando and S. Sojakka, "Pattern recognition in a bucket," in *Advances in Artificial Life* (Springer Berlin Heidelberg, 2003) p. 588–597.
[55]M. Cucchi, S. Abreu, G. Ciccone, D. Brunner, and H. Kleemann, "Hands-on reservoir computing: A tutorial for practical implementation," Neuromorph. Comput. Eng. **2**, 032002 (2022).
[56]S. Stepney, "Physical reservoir computing: A tutorial," Nat. Comput. **23**, 665–685 (2024).
[57]M. Lukoševičius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks of the Trade: Second Edition*, edited by G. Montavon, G. B. Orr, and K.-R. Müller (Springer Berlin Heidelberg, Berlin, Heidelberg, 2012) pp. 659–686.
[58]B. Jones, D. Stekel, J. Rowe, and C. Fernando, "Is there a liquid state machine in the bacterium escherichia coli?" in *2007 IEEE Symposium on Artificial Life* (2007) pp. 187–191.
[59]D. Nikolić, S. Häusler, W. Singer, and W. Maass, "Distributed fading memory for stimulus properties in the primary visual cortex," PLoS Biol. **7**, e1000260 (2009).
[60]P. Enel, E. Procyk, R. Quilodran, and P. F. Dominey, "Reservoir computing properties of neural dynamics in prefrontal cortex," PLoS Comput. Biol. **12**, e1004967 (2016).
[61]Y. Yada, S. Yasuda, and H. Takahashi, "Physical reservoir computing with FORCE learning in a living neuronal culture," Appl. Phys. Lett. **119**, 173701 (2021).
[62]I. Auslender, G. Letti, Y. Heydari, C. Zaccaria, and L. Pavesi, "Decoding neuronal networks: A reservoir computing approach for predicting connectivity and functionality," Neural Netw. **184**, 107058 (2025).
[63]T. Sumi, H. Yamamoto, Y. Katori, K. Ito, S. Moriya, T. Konno, S. Sato, and A. Hirano-Iwata, "Biological neurons act as generalization filters in reservoir computing," Proc. Natl. Acad. Sci. (USA) **120**, e2217008120 (2023).
[64]Y. Kobayashi, "Information processing via human soft tissue: Soft tissue reservoir computing," IEEE Access **13**, 53706–53716 (2025).
[65]M. Ushio, K. Watanabe, Y. Fukuda, Y. Tokudome, and K. Nakajima, "Computational capability of ecological dynamics," R. Soc. Open Sci. **10**, 221614 (2023).
[66]O. Pieters, T. De Swaef, M. Stock, and F. wyffels, "Leveraging plant physiological dynamics using physical reservoir computing," Sci. Rep. **12**, 12594 (2022).
[67]K. Nakajima, H. Hauser, T. Li, and R. Pfeifer, "Information processing via physical soft body," Sci. Rep. **5**, 10487 (2015).
[68]F. Xue, Q. Li, H. Zhou, and X. Li, "Reservoir computing with both neuronal intrinsic plasticity and multi-clustered structure," Cognit. Comput. **9**, 400–410 (2017).
[69]V. Nikolić, M. Echlin, B. Aguilar, and I. Shmulevich, "Computational capabilities of a multicellular reservoir computing system," PLOS ONE **18**, e0282122 (2023).
[70]S. He and P. Musgrave, "Physical reservoir computing on a soft bio-inspired swimmer," Neural Netw. **181**, 106766 (2025).
[71]P.-R. Vlachas, J. Pathak, B. R. Hunt, T. P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos, "Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics," Neural Netw. **126**, 191–217 (2020).
[72]G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, Vol. 2 (2004) pp. 985–990 vol.2.
[73]M. T. Rosenstein, J. J. Collins, and C. J. De Luca, "A practical method for calculating largest lyapunov exponents from small data sets," Physica D: Nonlinear Phenomena **65**, 117–134 (1993).
[74]Z.-M. Zhai, L.-W. Kong, and Y.-C. Lai, "Emergence of a resonance in machine learning," Phys. Rev. Res. **5**, 033127 (2023).
[75]L. Pardo, *Statistical Inference Based on Divergence Measures*, Statistics: A Series of Textbooks and Monographs (CRC Press, 2018).

[76]J. J. B. Jack, D. Noble, and R. W. Tsien, *Electric Current Flow in Excitable Cells*, 1st ed. (Oxford University Press, 1975).
[77]B. Hille, *Ionic Channels of Excitable Membranes*, 3rd ed. (Sinauer Associates, 1984).
[78]E. R. Kandel, J. H. Schwartz, and T. M. Jessell, *Principles of Neural Science*, 3rd ed. (Elsevier, 1991).
[79]D. J. Aidley, *The Physiology of Excitable Cells*, 4th ed. (Cambridge University Press, 1998).
[80]H.-Y. He and H. T. Cline, "What is excitation/inhibition and how is it regulated? a case of the elephant and the wisemen," J. Exp. Neurosci. **13**, 1179069519859371 (2019).
[81]C. Heckman, C. Mottram, K. Quinlan, R. Theiss, and J. Schuster, "Motoneuron excitability: The importance of neuromodulatory inputs," Clin. Neurophysiol. **120**, 2040–2054 (2009).
[82]D. Tsuboi, T. Nagai, J. Yoshimoto, and K. Kaibuchi, "Neuromodulator regulation and emotions: insights from the crosstalk of cell signaling," Front. Mol. Neurosci. **17**, 1376762 (2024).
[83]B. Ibarz, J. Casado, and M. Sanjuán, "Map-based models in neuronal dynamics," Phys. Rep. **501**, 1–74 (2011).
[84]G. Baghdadi, S. Jafari, J. Sprott, F. Towhidkhah, and M. Hashemi Golpayegani, "A chaotic model of sustaining attention problem in attention deficit disorder," Commun. Nonlinear Sci. Numer. **20**, 174–185 (2015).
[85]L. Jaurigue, "Chaotic attractor reconstruction using small reservoirs—the influence of topology," Mach. Learn.: Sci. Technol. **5**, 035058 (2024).
[86]Y. Wang and C. A. Shoemaker, "A general stochastic algorithmic framework for minimizing expensive black box objective functions based on surrogate models and sensitivity analysis," arXiv preprint arXiv:1410.6271 (2014).
[87]D. E. Goldberg, *Genetic Algorithms* (Pearson Education India, 2006).