nature communications

Article

https://doi.org/10.1038/s41467-024-49190-4

Reservoir-computing based associative memory and itinerancy for complex dynamical attractors

Received: 16 October 2023

Accepted: 24 May 2024

Published online: 06 June 2024

Check for updates

Ling-Wei Kong[1,2], Gene A. Brewer[3] & Ying-Cheng Lai[2,4]

Traditional neural network models of associative memories were used to store and retrieve static patterns. We develop reservoir-computing based memories for complex dynamical attractors, under two common recalling scenarios in neuropsychology: location-addressable with an index channel and content-addressable without such a channel. We demonstrate that, for location-addressable retrieval, a single reservoir computing machine can memorize a large number of periodic and chaotic attractors, each retrievable with a specific index value. We articulate control strategies to achieve successful switching among the attractors, unveil the mechanism behind failed switching, and uncover various scaling behaviors between the number of stored attractors and the reservoir network size. For content-addressable retrieval, we exploit multistability with cue signals, where the stored attractors coexist in the high-dimensional phase space of the reservoir network. As the length of the cue signal increases through a critical value, a high success rate can be achieved. The work provides foundational insights into developing long-term memories and itinerancy for complex dynamical patterns.

While the development of artificial neural memories had been earlier, the celebrated Hopfield model[1,2] was a significant milestone in the modeling of associative memory in neural networks. In this model, the neurons are characterized by discrete dynamical states (e.g., +1 or −1). With training to adjust the connection weights among the neurons, the network can store a number of patterns represented by a grid of such discrete values. When a new pattern is presented to the network, it responds by setting the neural states in such a way that the resulting pattern most closely resembles the new one - the process of memory retrieval. The neural network thus constitutes effectively an associative memory. The dynamics of the neurons in the original Hopfield model are simply binary, but more sophisticated, oscillatory neural dynamical models can be used for associative memory[3–8]. Recently, deep neural networks have been used to implement associative memory[9]. It is noteworthy that, in most previous works on artificial neural networks as associative memory, the patterns to be "memorized" were static.

Can artificial neural networks be exploited as memory device for complex dynamical patterns, such as chaotic attractors? Such an object is the result of the time evolution of a nonlinear dynamical system, which manifests itself as a dynamical trajectory in the phase space. One might be tempted to represent a chaotic attractor using a grid of cells as a static pattern. There are two fundamental difficulties rendering this approach inappropriate. First, a chaotic attractor typically possesses a fractal structure in phase space[10], so representing it using a static grid may require a prohibitively high resolution. Second, the probabilities for the trajectory to land in different cells can be singular, defying using a finite set of discrete values to represent the "frequencies of visit" to different cells. Here, for realizing associative memory for complex dynamical attractors, we propose a drastically different approach than the Hopfield type of neural networks. The idea is to train a neural machine as a dynamical system with the capability to generate a variety of distinct dynamical trajectories, each

[1]Department of Computational Biology, Cornell University, Ithaca, New York, USA. [2]School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, Arizona, USA. [3]Department of Psychology, Arizona State University, Tempe, Arizona, USA. [4]Department of Physics, Arizona State University, Tempe, Arizona, USA. e-mail: Ying-Cheng.Lai@asu.edu

corresponding to a complex attractor to be memorized. The purpose of this paper is to demonstrate that this is indeed feasible and, moreover, to study what phenomena would emerge in such multifunctional dynamical systems.

For the associative memory to produce the correct dynamical trajectory corresponding to a desired attractor, an essential requirement for successful retrieval of the attractor is that the memory itself be a closed-loop dynamical system capable of generating continuous time evolution of the relevant state variables. Reservoir computing[11–13], a class of recurrent neural networks (RNNs) that has been extensively investigated for model-free prediction of chaotic systems in recent years[14–38], stands out as a suitable choice. Quite recently, an implementation of neural computation through reservoir computing was demonstrated[39]. Reservoir computing is a particularly suitable framework for our purposes. The training process is highly efficient, requiring only a linear regression to achieve satisfactory results. This efficiency proves invaluable as the number of target states increases. More importantly, this straightforward training method helps to avoid several significant issues. Notably, it circumvents the problem of catastrophic forgetting, which is particularly challenging when neural networks need to memorize a large number of distinct states[40]. Additionally, it addresses the vanishing/exploding gradient problem that can hinder the learning process in RNNs. Here, we shall demonstrate how reservoir computing can be exploited to store and retrieve complex dynamical attractors. Different attractors are trained to coexist within a single reservoir neural network and, when needed, can be recalled later by suitable cues. For simplicity, a reservoir neural network is sometimes also called a reservoir computer (RC).

In the science of memory, there are two types of memories: short-term and long-term[41–45]. Reservoir computing, because of its recurrent structure, has naturally incorporated memories in its dynamical evolution in that past inner states and inputs can affect the future state and outputs. The temporary information of previous inputs stored within the dynamical state evolution of the hidden layer represents a kind of short-term memory of the RNN. The focus of our study is on long-term memories that are encoded within the weights and connections in the neural network architecture, manifested as the stabilized dynamical trajectories that can be maintained in the dynamical network. In neuropsychology, two distinct types of models of long-term memory are often used: "location-addressable" and "content-addressable" memories[46], where the cue used to address the target memory for the former corresponds to a specific index for each memorized pattern and, for the latter, the cue can be made of a short time series correlated with the dynamical trajectory from the memorized attractor. Our machine-learning memory for complex dynamical attractors incorporates both types of models, named index-based and index-free memory, respectively.

A neural network capable of memorizing and retrieving multiple memory states is also a multifunctional neural network. When different states are recalled, the neural network exhibits distinct dynamical behaviors. The idea of multifunctional RNNs has been discussed previously with different forms of implementation. In particular, in the index-based approach, an index parameter is introduced to modulate the functionality of the neural networks and store different states with different index values[29,31,32,47,48], but recalling the memorized states was a challenge, for which control may be necessary. To our knowledge, prior to our work, there were no discussions about the scaling law of the memory capacity in the context of storing different states in RNNs. Exploiting multistability in the neural networks represents another approach - the index-free approach[49–52], where multiple attractors coexist in the high-dimensional hidden phase space of the reservoir network. In this regard, previous work focused on storing fixed points[49], and there were also methods based on storing chaotic or periodic states[50–52]. Outstanding issues included the dynamical mechanism underlying the retrieval of the coexisting dynamical states and their basin structure. Here, we shall demonstrate that our reservoir-computing based classifier can distinguish among different recalled states and between a successful and failed recall, both with high accuracy, thereby providing quantitative insights into the open issues.

Our main results can be summarized, as follows. For the location-addressable scenario, a single reservoir computer can "memorize" a number of distinct complex dynamical attractors (also referred to as the complex memory states). Each memory state is embedded in the high-dimensional phase space of the hidden reservoir neural network, is maintained (stored) indefinitely, and can be retrieved whenever needed. A key issue is how successful a transition between two arbitrarily memorized states can be, for which some proper control actions are needed. We calculate the success rate of transition among different states in a reservoir computer, obtain a dynamical understanding behind the failed switching, and articulate several control strategies to significantly enhance the success rate. We also demonstrate the success of memorizing hundreds of different attractors in one single reservoir computer and uncover scaling laws between the size of the reservoir network and the number of memory states. For the content-addressable setting without an index channel, we exploit multistability for retrieval with the aid of some cue signals, where different memory states can be coexisting attractors or long transient states. We find that the stored states can indeed be recalled by short cues and even partial or noisy cues. We uncover a transition phenomenon that arises in the retrieval success rate as the length of the cue signal varies, as well as sophisticated basin structures in the hidden high-dimensional phase space of the reservoir network. A connection between the transition phenomenon of success rate and the basin structure is established, yielding a dynamical understanding of the mechanism behind memory retrieval from the coexisting states. We also find that a natural random itinerancy is possible when there is noise on the neurons in our reservoir computers. These results provide foundational insights into developing machine-learning-based long-term memory devices for complex dynamical states or attractors.

Results

Index-based reservoir-computing memory

In the "location-addressable" or "parameter-addressable" scenario, the stored memory states within the neural network are activated by a specific location address or an index parameter. The stimulus that triggers the system to switch states can be entirely unrelated to the content of the activated state, and the pair linking the memory states and stimuli can be arbitrarily defined. For instance, the stimulus can be an environmental condition such as the temperature, while the corresponding neural network state could represent specific behavioral patterns of an animal. An itinerancy among different states is thus possible, given a fluctuating environmental factor. (The "parameter-addressable" scenario is different from the "content-addressable" scenario that requires some correlated stimulus to activate a memory state.)

Storage of complex dynamical attractors. The architecture of our index-based reservoir computing memory consists of a standard reservoir computer (i.e., echo state network)[11–13] but with one feature specifically designed for location-addressable memory: an index channel, as shown in Fig. 1(A). During training, the time series from a number of dynamical attractors to be memorized are presented as the input signal $\mathbf{u}(t)$ to the reservoir machine, each is associated with a specific value of the index p so that the attractors can be recalled after training using the same index value. This index value p modulates the dynamics of the RC network through an index input matrix W_{index} that projects p to the entire hidden layer with weights defined by the matrix entries. The training process is an open-loop operation with input $\mathbf{u}(t)$ and the corresponding index value p through the index channel. To store or memorize a different attractor, its dynamical trajectory

becomes the input, together with the index value for this attractor. After training, the reservoir output is connected with its input, generating closed-loop operation so that the reservoir machine is now capable of executing self-dynamical evolution starting from a random initial condition, "controlled" by the specific index value. More specifically, to retrieve a desired attractor, we input its index value through the index channel, and the reservoir machine will generate a dynamical trajectory faithfully representing the attractor, as schematically illustrated in Fig. 1(A) with two examples: one periodic and another chaotic attractor. The mathematical details of the training and retrieval processes are given in Supplementary Information (SI).

To demonstrate the workings of our reservoir computing memory, we generate a number of distinct attractors from three-dimensional nonlinear dynamical systems. We first conduct an experiment of memorizing and retrieving six attractors: two periodic and four chaotic attractors, as represented by the blue trajectories in the top row of Fig. 1(B) (the ground truth). The index values for these six attractors are simply chosen to be $p_i = i$ for $i = 1, \ldots, 6$. During

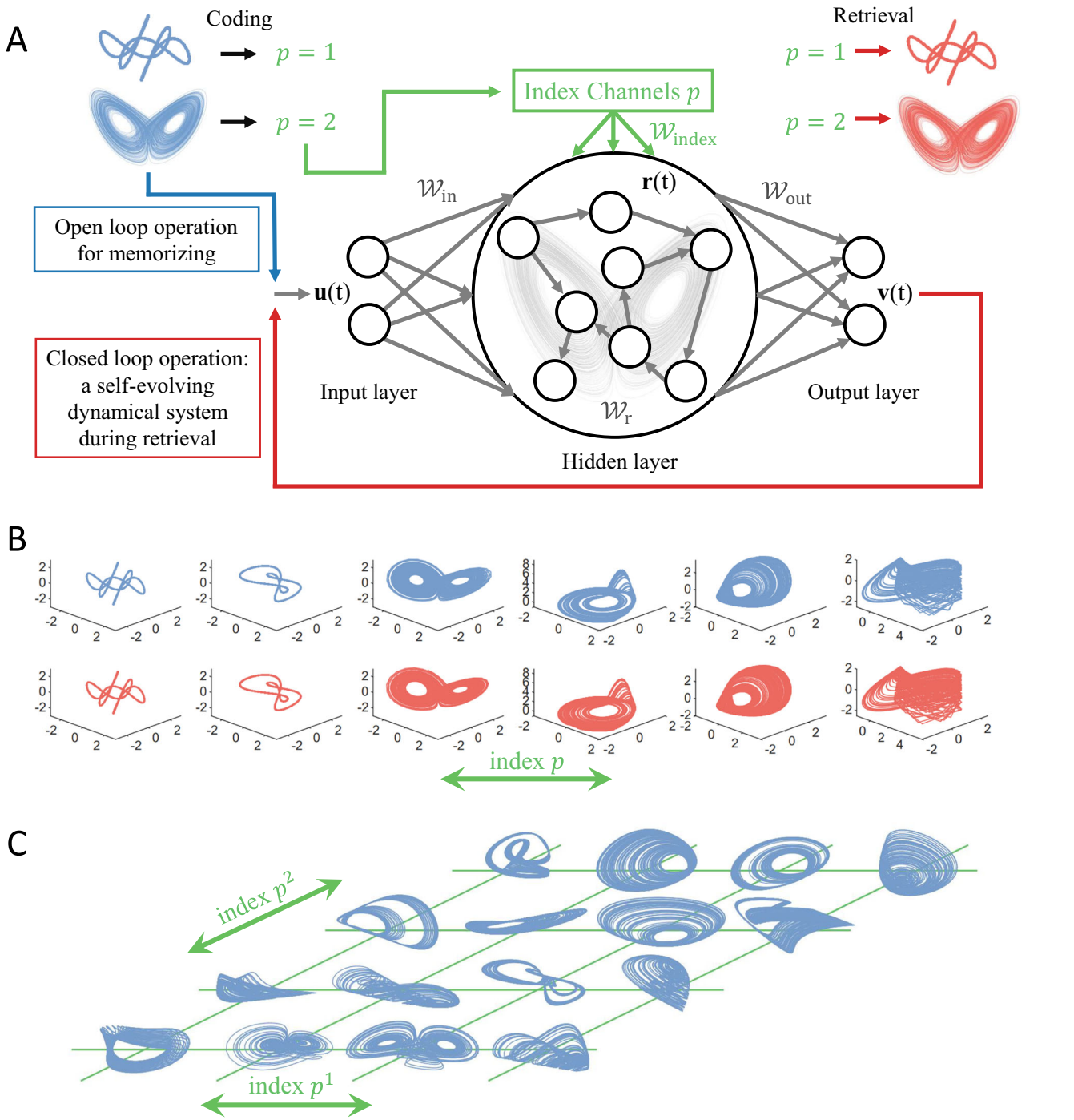

Fig. 1 | Memorizing attractors with index-based reservoir computing. A A schematic illustration of index-based reservoir-computing memory. **B** Retrieval of six memorized chaotic and periodic attractors using a scalar index p_i (one integer value for each memorized dynamical attractor), where the blue and red trajectories in the top and bottom rows, respectively, represent the original and retrieved attractors. Visually, there is little difference between the original and memorized/retrieved attractors. **C** Memorizing 16 chaotic attractors with two indices. The successfully retrieved attractors are presented in SI. All the memorized attractors are dynamically stable states in the closed-loop reservoir dynamical system.

training, the time series of each attractor is injected into the input layer, and the reservoir network receives the specific index value, labeling this attractor to be memorized so that the reservoir-computing memory learns the association of the index value with the attractor to be memorized. To retrieve a specific attractor, we set the index value p to the one that labeled this attractor during training and allow the reservoir machine to execute a closed-loop operation to generate the desired attractor. For the six distinct dynamical attractors stored, the respective retrieved attractors are shown in the second row in Fig. 1(B). The recalled attractors closely resemble the original attractors and for any specific value of the index, the reservoir-computing memory is capable of generating the dynamical trajectory for an arbitrarily long period of time. The fidelity of the recalled states, in the long run, can be assessed through the maximum Lyapunov exponent and the correlation dimension. In particular, we train an ensemble of 100 different memory RCs, recall each of the memory states, and generate outputs continuously for 200,000 steps. The two measures are calculated during this generation process and compared with the ground truth. As an example, for the Lorenz attractor with the true correlation dimension of 2.05 ± 0.01 and largest Lyapunov exponent of about 0.906, 96% and 91% of the retrieved attractors have their correlation dimension values within 2.05 ± 0.02 and exponent values in 0.906 ± 0.015, indicating that the memory RC has learned the long-term "climate" of the attractors and is able to reproduce them (see SI for more results).

For a small number of dynamical patterns (attractors), a scalar index channel suffices for accurate retrieval, as illustrated in Fig. 1(B). When the number of attractors to be memorized becomes large, it is useful to increase the dimension of the index channel. To illustrate this, we generate 16 distinct chaotic attractors from three-dimensional dynamical systems whose velocity fields consist of different combinations of power-series terms[53], as shown in Fig. 1(C). For each attractor, we use a two-dimensional index vector $p_i = (p_i^1, p_i^2)$ to label it, where $p_i^1, p_i^2 \in \{1,2,3,4\}$. The attractors can be successfully memorized and faithfully retrieved by our index-based reservoir computer, with detailed results presented in SI.

Given a number of dynamical patterns to be memorized, there can be many different ways of assigning the indices. In addition to using one and two indices to distinguish the patterns, as shown in Fig. 1(B) and (C), we studied two alternative index-assignment schemes: binary and one-hot code. For the binary assignment scheme, we store K dynamical patterns using $\log_2 K$ number of channels. The index value in each channel can either be one of the two possible values. For the one-hot assignment scheme, the number of index channels is the same as the number K of attractors to be memorized. The index value in each channel is still one of the two possible values, for instance, either 0 or 1. For the one-hot assignment, each attractor is associated with one channel, where only this channel can take the number one for the attractor, while the values associated with all other channels are zero. Similarly, for each channel, the index value can be one if and only if the attractor being trained/recalled is the state with which it is associated. Arranging all the index values as a matrix, one-hot assignment leads to an identity matrix. For a given reservoir hidden-layer network, different ways of assigning indices result in different index-input schemes to the reservoir machine and can thus affect the memory performance and capacities. The effects are negligible if the number of dynamical attractors to be memorized is small, e.g., a dozen or fewer. However, the effects can be pronounced if the number K of patterns becomes larger. One way to determine the optimal assignment rule is to examine, under a given rule, how the memory capacity depends on or scales with the size of the reservoir neural network in the hidden layer, which we will discuss later.

Transition matrices among stored attractors. We study the dynamics of memory toggling among the stored attractors. As each memorized state s_i is trained to be associated with a unique index value p_i, the reservoir network's dynamical state among different attractors can be switched by simply altering the input index value. Several examples are shown in Fig. 2(A) and (B), where p is switched among various values multiple times. The dynamical state activated in the reservoir network switches among the learned attractors accordingly. For instance, at step 800, for a switch from $p = 1$ to $p = 6$, the reservoir state switches from a periodic Lissajous state to a chaotic Hindmarsh-Rose (HR) neuron state. However, not all such transitions can be successful - a failed example is illustrated in the bottom panel of Fig. 2(C). In this case, after the index switching, the reservoir system falls into an undesired state that does not belong to any of the trained states. The probability of failed switchings is small and asymmetric between the two states before and after the switching. A weighted directed network can be defined among all the memorized states where the weights are the success probabilities, which can be represented by a transition matrix. Making index switches at many random time steps and counting the successful switches versus the failed switches, we numerically obtain the transition matrix, as exemplified in Fig. 2(D), from two different randomly generated indexed RCs. Figure 2(E) shows the average transition matrix of an ensemble of 25 indexed RCs. Figure 2(F) shows that the variance of the success rate across different columns (i.e., different destination states) is much larger than the variance across different rows (i.e., different starting states), implying that the success rate is significantly more dependent on the destination attractor than on the starting attractor.

What is the origin of the asymmetric dependence in the transition matrices and the dynamical mechanism behind the switch failures? Two observations are helpful. The first concerns the dynamic consequence of a switch in the index value p. Incorporating this term of p into the reservoir computer is equivalent to adding an adjustable bias term to each neuron in the reservoir hidden layer. Different values of p thus directly result in different bias values on the neurons and different dynamics. The same indexed RC under different p can be treated as different dynamical systems. Second, note that, during a switch, one does not directly interfere with the state input u or the hidden state r of the RC network, but only changes the value of p. Consequently, a switch in p as in Fig. 2(A) switches the dynamical equations of the RC network while keeping the RC states r_{last} and the output v_{last} at the last time step, which is passed from the previous dynamical system to the new system after switching. This pair of r_{last} and v_{last} becomes the initial hidden state and initial input under the new dynamic equation of RC.

The two observations suggest examining the basin of attraction of the trained state in the memory RC hidden space under the corresponding index value, which typically does not fully occupy the entire phase space. Figure 2(I) shows the basin structure of two arbitrary states in an indexed RC trained with 16 attractors. The blue regions, leading to the trained states, leave some space for the orange regions that lead to untrained states and failed to switch. If the RC states at the last time step before switching lay outside the blue region, the RC network will evolve to an unwanted state, and a switch failure will occur. We further plot the points from the attractor before switching in blue (successful) and orange (unsuccessful), as shown in Fig. 2(H). They are the projections of the basin structure of the new attractors onto the previous attractor. This picture provides an explanation for the strong dependence of success rates on the destination states. In particular, the success rate is determined by two factors: the relative size of the basin of the new state after switching and the degree of overlap between the previous attractor and the new basin. While the degree of overlap depends on both the starting and destination states, the relative size of the new state basin is solely determined by the destination state, resulting a strong dependency on the switching success rate of the destination state. [Further illustrations of (i) the basin structure of the memorized states in an indexed RC, (ii) projections of the basin structure of the destination state back to the starting

https://doi.org/10.1038/s41467-024-49190-4

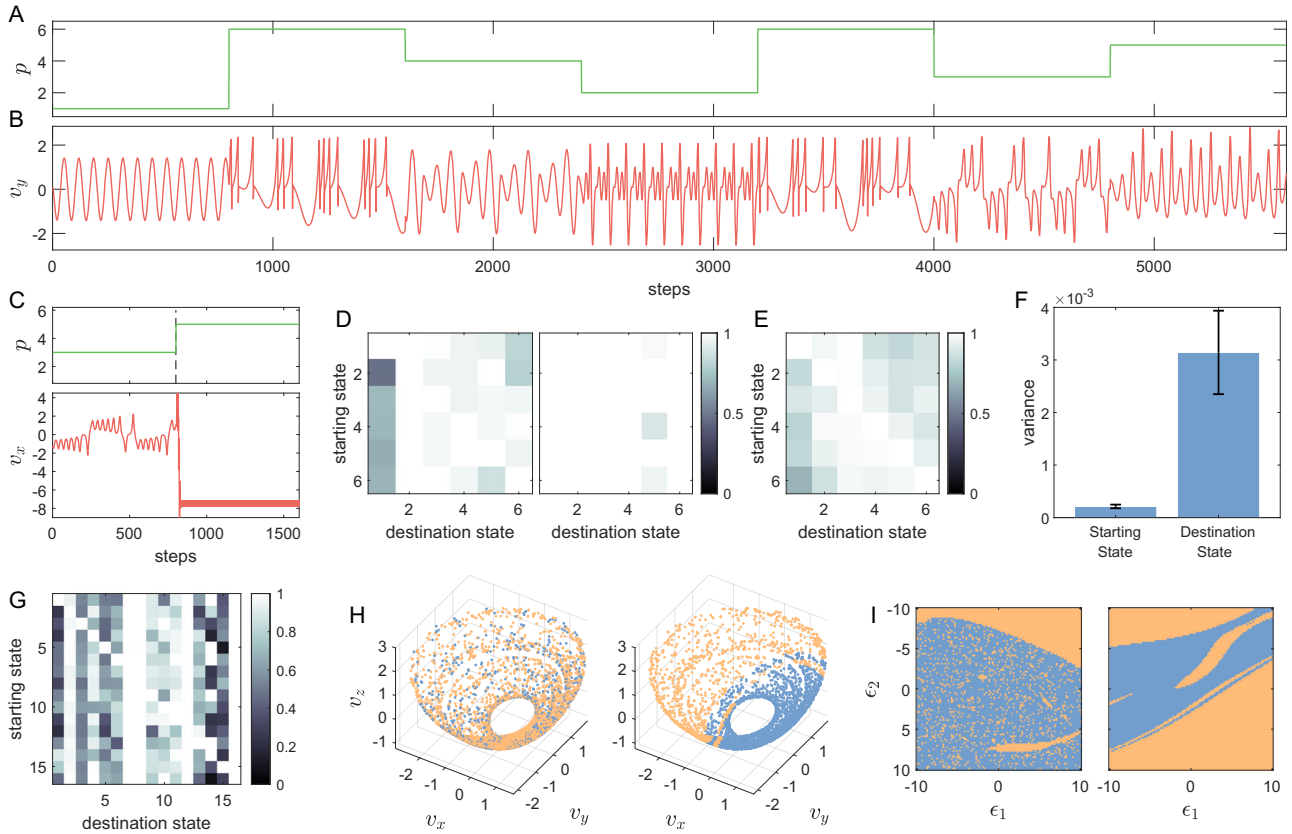

Fig. 2 | Transitions among different memorized attractors in an indexed memory RC. A, **B** By changing the index value p as in panel (**A**), one can toggle among different states shown in panel (**B**) that have been memorized as attractors in the indexed memory RC (Dimensions v_x and v_z are not shown). **C** An instance of a failed switch, where the memory RC evolves to an untrained state after changing p. **D** Two typical transition matrices where six different states [as the ones in Fig. 1(B)] were memorized. **E** The average of transition matrices from 25 memory RCs with different random seeds and the same settings as in panel (**D**). **F** Variances in the success rate with respect to the starting and destination attractors, revealing a stronger dependence on the latter. The result is from the ensemble of 25 RC networks, each trained with ten different chaotic attractors. **G** A deliberately chosen memory RC with 16 memory states has relatively low success rates among many states for visualizing the basin structures shown below. **H** Attracting regions of two different destination states on the manifold of a starting state. The memory RC originally operates on this shell-shaped starting attractor, and the index p is toggled at different time steps when the memory RC is at different positions on this attractor. Such a position can determine whether the switching is successful, where the successful positions are marked blue and failed positions are marked lighter orange. **I** Local two-dimensional slices of the basin structures of the two destination states from panel (**H**) in the high-dimensional RC hidden space. Again, the blue and lighter orange regions mark the basins of the memorized state and of some untrained states leading to failed switches, respectively. The quantities ϵ_1 and ϵ_2 define the scales of perturbations to the RC hidden space in two randomly chosen perpendicular directions. The $\epsilon_1 = \epsilon_2 = 0$ origin points are the RC hidden states taken from random time steps while the memory RC operates at the corresponding destination states.

state attractor, and (iii) how switch success is strongly affected by the basin structure of the destination state are provided in SI.]

Control strategies for achieving high memory transition success rates. For a memory system to be reliable, accessible, and practically useful, a high success rate of switching from one memorized attractor to another upon recalling is essential. As this switching failure issue is rooted in the problem of whether an initial condition is inside the attracting basin of the destination state, such type of failure can be universal for models for memorizing not exactly the (static) training data but the dynamical rules of the target states and regenerating the target states by evolving the memorized dynamical rules. Even if one uses a series of networks to memorize the target states separately, the problem of properly initializing each memory network during a switch or a recall persists. A possible solution to this initial state problem is to optimize the training scheme or the RNN architecture to make the memorized state globally attractive in the entire hidden space - an unsolved problem at present. Here we focus on noninvasive control strategies to enhance the recall or transition success rates without altering the training technique or the RC architecture, assuming a memory RC is already trained and fixed. The established connections or weights will not be modified, ensuring that the memory attractors already in place are preserved.

The first strategy, named "tactical detour," utilizes some successful pathways in the transition matrix to build indirect detours to enhance the overall transition success rates. Instead of switching from an initial attractor to a destination attractor directly, going through a few other intermediate states can result in a high success rate, as exemplified in Fig. 3(A). This method can be relatively simple to implement in many scenarios, as all needed is switching the p value a few more times. However, this strategy has two limitations. First, it is necessary to know some information about the transition matrix to search for an appropriate detour and estimate the success rate. Second, the dynamical mechanism behind failed switching suggests that the success rate mostly depends on the relative size of the attracting basin of the destination state, implying that a state that has a small attracting basin is difficult to reach from most other states. As a result, the improvement of the success rate from a detour can fall below 100%. As exemplified in Fig. 3(B), the success rate with detours, which can significantly increase the overall success rate within just a few steps, can saturate at some levels lower than one.

https://doi.org/10.1038/s41467-024-49190-4

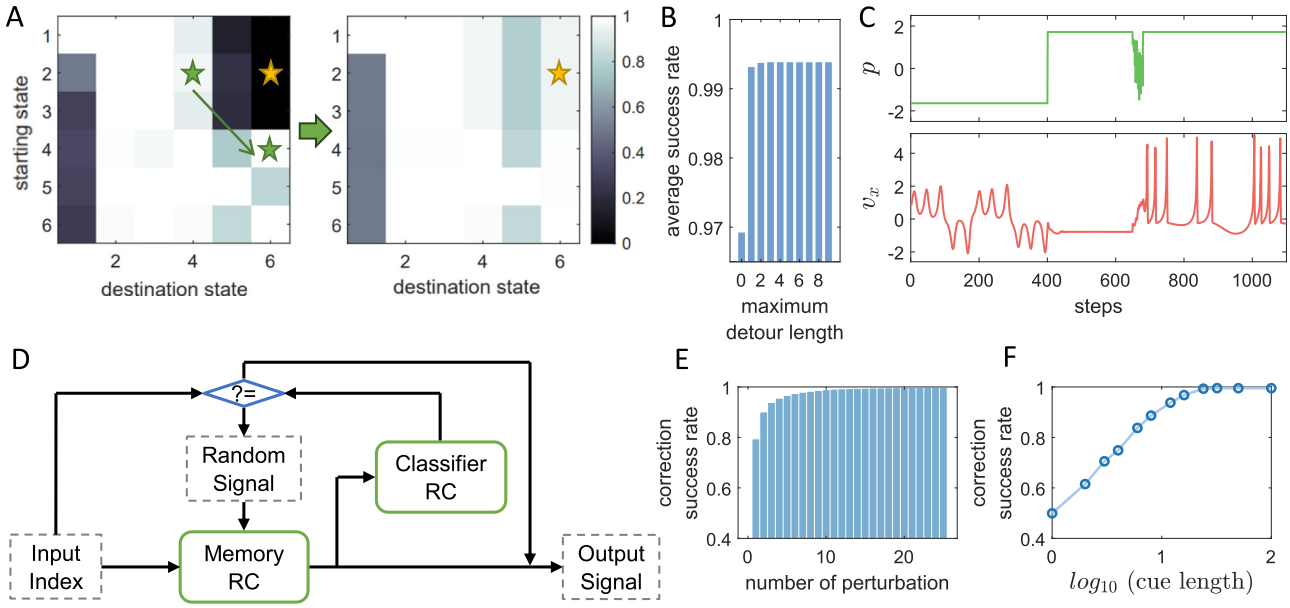

Fig. 3 | Control strategies for achieving high memory transition success rates. **A** Achieving higher switching success rates using detours, where the switching from attractor 2 to attractor 6 via a detour through attractor 4 significantly increases the success rate. The right panel illustrates the improved transition matrix with detours as compared to the original one on the left. **B** Average success rate with different maximum detour lengths. A minimal detour is typically needed - just one single step of detour can reduce more than two-thirds of the failing probability, obtained from the same ensemble of memory RCs as in Fig. 2(F). **C** An instance of feedback control to correct a failed switch, following the scheme in panel (**D**). The switch at the 400th time step leads to an untrained static state. Since the classifier RC cannot recognize the memory RC output as the desired state, a short random signal is applied to perturb the memory RC, which adjusts the state of the memory RC to achieve a successful retrieval. **D** Illustration of the controlled memory retrieval by a classifier RC and random signals, where a classifier RC is used to identify the output from the memory RC. The outcome of this identification decides whether to apply a random perturbation or to use the memory RC output as the final result. **E** The rates of successfully corrected failed switches by the control mechanism illustrated in (**C**) and (**D**). The denominator of this correction success rate is the number of failed switches, not all switches. Each perturbation is implemented by ten steps of standard Gaussian noise. Just one "perturb and classify" cycle can correct about 80% of the failed switches. The results of this approach with different parameter choices are shown in SI. **F** Performance of the cue-based method to correct failed switches with respect to cue length. Within 25 steps of the cue, the correction success rate reaches 99%. Panels (**E**) and (**F**) are generated from the same ensemble of 25 memory RCs, each trained to memorize the six attractors as in Fig. 1.

Our second strategy is of the feedback control type with a classifier reservoir computer and random signals to perturb the memory reservoir computer until the latter reaches the desired memory state, as illustrated and explained in Fig. 3(C, D). This trial-and-error approach is conceptually similar to the random memory search model in psychology[54]. A working example is shown in Fig. 3(C), where the input index p first activates certain outputs from the memory RC, which are fed to a classifier reservoir computer trained to distinguish among the target states and between the target states and non-target states. As detailed in SI, a simple training of this classifier RC can result in a remarkably high accuracy. The output of the classifier RC is compared with the index value. If they agree, the output signal represents the correct attractor; otherwise, a random signal is injected into the memory RC to activate a different outcome. The feedback loop continues to function until the correct attractor is reached. This control strategy can lead to a nearly perfect switch success rate, at the cost of a possibly long switching time. Let T_n be the time duration that the random perturbation lasts, T_c be the time for making a decision about the output of the classifier, and $\eta_{i,j}$ be the switching success rate from a starting attractor s_i to a destination attractor s_j. The estimated time for such a feedback control system to reach the desired state is $T_{\text{reach},i,j} = (T_n + T_c)/\eta_{i,j}$, which defines an asymmetric distance between each pair of memorized attractors.

To provide a quantitative estimate of the performance of this strategy, we run 90,000 switching in 25 different memory RCs trained with the six periodic and chaotic states as illustrated in Fig. 1(B). The size of the reservoir network is 1000, and other hyperparameter values are listed in SI. Among the 90,000 switchings, 8110 trials (about 9%) failed, which we used as a pool to test the control strategies. To make the results generic, we test 12 different control settings with different parameters. We test four different lengths of each noisy perturbation period, including 1 step, 3 steps, 10 steps, and 30 steps, with Gaussian white noise at three different levels (standard deviations: $\sigma = 0.3$, $\sigma = 1$, and $\sigma = 3$). We find that with an appropriate choice of the control parameters, 10 periods of 10 steps of noise perturbations (so 100 steps of perturbations in total) can eliminate 99% of the failed switchings, as illustrated in Fig. 3(E). The full results of the 12 different control settings are demonstrated and discussed in SI.

Our third control strategy is also motivated by daily experience: recalling an object or an event can be facilitated by some relevant, reminding cue signals. We articulate using a cue signal to "warm up" the memory reservoir computer after switching the index p to that associated with the desired attractor to be recalled. The advantage is that the memory access transition matrix is no longer needed, but the success depends on whether the cue is relevant and strong enough. The cue signals are thus state-dependent: different attractors require different cues, so an additional memory device may be needed to restore the warming signals containing less information than the attractors to be recalled. This leads to a hierarchical structure of memory retrieval: (a) a scalar or vector index p, (b) a short warming signal, and (c) the whole memorized attractor, similar to the workings of human memory suggested in ref. 55. We also examine if state-independent cues (uniform cues for all the memory states) can be helpful, and find that in many cases, a simple globally uniform cue can make most memorized attractors almost randomly accessible, but there is also a probability that this cue can almost entirely block a few memory states.

https://doi.org/10.1038/s41467-024-49190-4

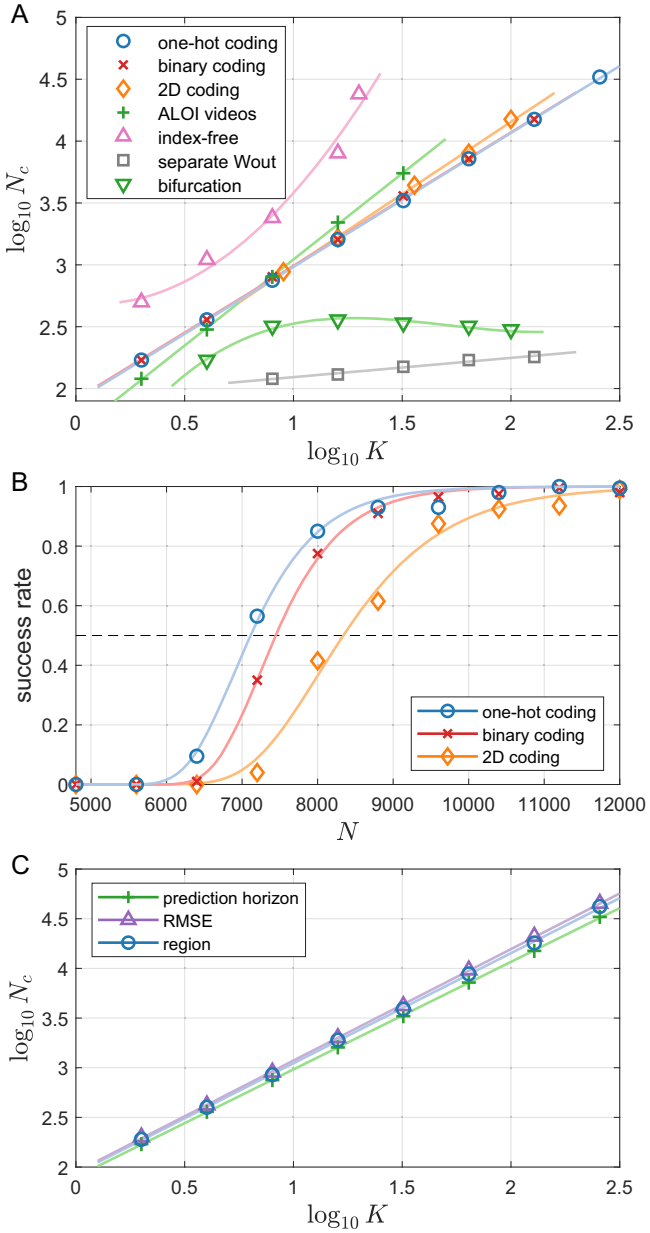

Fig. 4 | Scaling behaviors of the memory capacity. A Algebraic scaling relation between the number K of dynamical patterns to be memorized and the required size N_c of the RC network for various tasks and different memory coding schemes. Details of each curve/task are discussed in the main text. The "separate W_{out}" and "bifurcation" tasks are shown with prediction-horizon-based measures, while all the other curves are plotted with the region-based measure. **B** Examples of how the success rate increases with respect to N and comparisons among different coding schemes, where the success rate of accurate memory recalling (using the region-based performance measure) of three coding schemes on the dataset #1 with $K = 64$ is shown. All three curves have a sigmoid-like shape between zero and one, where the data points closest to the 50% success rate threshold correspond to N_c. All data points in the scaling plots are generated from this type of curve (success rate versus N) to extract N_c. The one-hot coding is the most efficient coding of the three we tested for this task. **C** Comparisons among different recalling performance measures. The three different measures on the dataset #1 with a one-hot coding have indistinguishable scaling behavior with only small differences in a constant factor.

With this third control strategy with the cues, we again employ the 8,110 failed switching pool to test the strategy performance. As shown in Fig. 3(F), with the increase of the cue length, the success rate increases nearly exponentially. Almost 90% of the failed switchings can be eliminated with just 8 steps of cue, and 24 steps of cue can eliminate more than 99% of the failed switchings.

Scaling law for memory capacity. Intuitively, a larger RNN in the hidden layer would enable more dynamical attractors to be memorized. Quantitatively, such a relation can be characterized by a scaling law between the number K of attractors and the network size N. Our numerical method to uncover the scaling law is as follows. For a fixed pair of (K, N) values, we train an ensemble of reservoir networks, each of size N, using the K dynamical attractors. The fraction of the networks with validation errors less than an empirical threshold gives the success rate of memorizing the patterns, which in general increases with N. A critical network size N_c can be defined when the success rate is about 50%. (Here, we use 50% as a threshold to minimize the potential error in N_c due to random fluctuations. In SI, we provide results with 80% as the success rate threshold. The conclusions remain the same under this change.)

We use three different types of validation performance measures. The first is the prediction horizon defined by the prediction length before the deviation of the prediction time series reaches 10% of the oscillation amplitude of the real state (half of the distance of the maximum value minus the minimum value in a sufficiently long time series of that target state). The oscillation amplitude and the corresponding deviation rate are calculated for each dimension of the target state, and the shortest prediction horizon is taken among all the dimensions of the target state. The prediction horizon is rescaled by the length of the period of the target system to facilitate comparison. For a chaotic state, we use the average period defined as the average shortest time between two local maximums in a sufficiently long time series of that target state. The threshold of this prediction horizon for a successful recall is set to be two periods of each memory state. The second performance measure is the mean square root error (RMSE) with a validation time of four periods (or average periods). The third measure is defined by the time before the predicted trajectories go beyond a rectangular-shaped region surrounding the real memory state to an undesired region in the phase space, which is set to be 10% larger than the rectangular region defined by the maximum and minimum values of the memory state in each dimension. For these three measures, the thresholds for a successful recall can be different across datasets but are always fixed within the same task (the same scaling curve). The thresholds for the prediction-horizon-based measures and region-based measures are several periods of oscillation (or average periods for chaotic states) of the target state.

Figure 4(A) shows the resulting scaling law for the various tasks, with different datasets of states, different training approaches, and different coding schemes for the indexed RCs. The first dataset (dataset #1) consists of many thousands of different periodic trajectories. With index-based RCs, we use one-hot coding, binary coding, and two-dimensional coding with this dataset, as shown in Fig. 4(A). They all lead to similar algebraic scaling laws $N_c \propto K^{\gamma}$ that are close to a linear law: $\gamma = 1.08 \pm 0.01$ for both the one-hot coding and binary coding, and $\gamma = 1.17 \pm 0.02$ for the 2D coding. In all the three cases, N_c all grow slightly faster than a linear law. A zoom-in picture comparing the three coding schemes is shown in Fig. 4(B). It can be seen that the one-hot coding is more efficient than the other two. Figure 4(B) also demonstrates how the data points in the scaling laws in Fig. 4(A) are gathered. Similar success rates versus N curves are plotted for each K value for each task and the data point from that curve which is closest to a 50% success rate is taken to be N_c.

Why is the one-hot coding more efficient than the other coding scheme we test? In our index-based approach to memorizing dynamical attractors, an index value p_i is assigned for each attractor s_i and acts as a constant input through the index channel to the reservoir neural network, which effectively modulates the biases to the network. Since the index input p_i is multiplied by a random matrix before

entering the network, each neuron receives a different random level of bias modulation, affecting its dynamical behavior through the nonlinear activation function. In particular, the hyperbolic tangent activation function has a number of distinct regions, including an approximately linear region when the input is small, and there are two nearly constant saturated regions for large inputs. With one specific p_i value injected into the index channel, different neurons in the hidden layer can be distributed to/across different functional regions (detailed demonstration in SI). When the index value is switched to another one, the functional regions for each neuron are redistributed, leading to a different dynamical behavior of the reservoir computer. Among the three coding schemes, the one-hot coding scheme utilizes one column of the random index input matrix, so the bias distributions for different attractors are not related to each other, yielding a minimal correlation among the distributions of the functional regions of the neurons for different values of p_i. For the other two coding schemes, there is a certain degree of overlap among the input matrix entries for different attractors, leading to additional correlations in the bias distributions, thereby reducing the memory capacity.

To make the stored states more realistic, we apply a dataset processed from the ALOI videos[56]. This dataset contains short videos of rotating small everyday objects, such as a toy duck or a pineapple. In each video, a single object rotates horizontally through a full 360-degree turn, returning to its initial angle. Repeating the video generates a periodic dynamical system. To reduce the computational loads due to the high spatial dimensionality of the video frames, we perform dimension reduction to the original videos through the principal component analysis, noting that most pixels are black backgrounds and many pixels of the object are highly correlated. We take the first two principal components of each video as the target states. The results are shown in Fig. 4(A) as the "ALOI videos" task. Similar to the previous three tasks, we obtain an algebraic scaling law $N_c \propto K^{\gamma}$ that is slightly faster than a linear law with $\gamma = 1.39 \pm 0.01$.

The "bifurcation" task is an example to show the potential of our index-based approach, where N_c grows much slower than a linear scale with respect to K and can even decrease in certain cases. It consists of 100 dynamical states gathered from the same chaotic food chain system but with different parameter values. We sweep an interval in the bifurcation diagram of the system with both periodic states of different periods and chaotic states with different average periods. For index coding, we use a one-dimensional index and assign the states by sorting the system parameter values used to generate these states. The index values are in the interval $[-2.5, 2.5]$, and are evenly separated on this interval. As shown in Fig. 4(A), the scaling behavior of this "bifurcation" task is much slower than a linear law, suggesting that when the target states are correlated, and the index values are assigned in a "meaningful" way, the reservoir computer can utilize the similarities among target states for more efficient learning and storing. Moreover, we notice that N_c can even decrease in a certain interval of K, as the amount of total training data increases with larger K, making it possible for the reservoir computer to reach a similar performance with a smaller value of N_c. In other tasks with other training sets, training data from different target states are independent of each other. In summary, training with the index-based approach on this dataset is more resource-efficient than having a series of separate reservoir computers trained with each target state and adding another selection mechanism to switch the reservoir computer while operating switching or itinerary behaviors.

One of the most important features of reservoir computing is that the input layer and the recurrent hidden layer are generated randomly and, thus, are essentially independent of the target state. One way to utilize this convenient feature is to use the same input and hidden layer for all the target states but with a separate output layer trained for each target state. In such an approach, an additional mechanism of selecting the correct output layer (i.e., W_{out}) during retrieval is necessary, unlike the other approaches studied with the same integrated reservoir computer for all the different target states. However, training with separate W_{out} can lower the computational complexity in some scenarios, as shown by the "separate W_{out}" task in Fig. 4(A), where N_c increases as a power law with K that is much slower than a linear law. However, training with separate W_{out} still makes the overall model complexity grow slower than linear, as the same number of independent W_{out} is needed as the number of K. For a scenario such as in the "bifurcation" task, the vanilla index-based approach is more efficient. As the input layer and hidden layer are shared by all the different target states, the issue of failed switching still exists, so control is required.

We further demonstrate the efficiency of our index-based approach compared to the index-free approach. In the "index-free" task shown in Fig. 4(A), dataset #1 was used. While the index-based approach has an almost linear scaling, the scaling of the index-free approach grows much faster than a power law. For instance, the critical network size N_c with $K = 20$ is about $N_c = 2.4 \times 10^4$, which is almost as large as the N_c for $K = 256$ with the index-based memory RC on the same dataset. Moreover, the critical network size N_c for $K = 32$ is larger than 10^5. The comparison of the two approaches on the same task reveals that, while the index-free memory RC has the advantage of having content-addressable memory, it is computationally costly compared with the index-based approach, due to the severe overlapping among different target states for large K values, thereby requiring a large memory RC to distinguish different states.

To test the genericity of the scaling law, we compare the results from the same task (the "one-hot coding" task) with the three different measures, as shown in Fig. 4(C). All three curves have approximately the same scaling behavior, except a minor difference in a constant factor.

Index-free reservoir-computing based memory - advantage of multistability

The RC neural network is a high-dimensional system with rich dynamical behaviors[49–52,57,58], making it possible to exploit multistability for index-free memory. In general, in the high-dimensional phase space, various coexisting basins of attraction can be associated with different attractors or dynamical patterns to be memorized. As we will show, this coexisting pattern can be achieved with a similar training (storing) process to that of index-based memory but without an index. The stored attractors can then be recalled using the content-addressable approach with an appropriate cue related to the target attractor. To retrieve a stored attractor, one can drive the networked dynamical system into the attracting basin of the desired memorized attractor using the cues, mimicking a spontaneous dynamical response characteristic of generalized synchronization between the driving cue and the responding RC[50,59,60].

While the possibility of index-free memory RC was pointed out previously[50–52], a quantitative analysis of the mechanism of the retrieval was lacking. Such an analysis should include an investigation into how the cue signals affect the success rate, what the basin structure of the reservoir computer is, and a dynamical understanding connecting these two. A difficulty is how to efficiently and accurately determine, from a short segment of RC output time series, which memory state is recalled and if any target memory has been recalled at all. For this purpose, short-term performance measures such as the RMSE and prediction horizon are not appropriate, as the ground truth of a specific trajectory of the corresponding attractor is not available, especially for the chaotic states. Moreover, a measure based on calculating the maximum Lyapunov exponents or the correlation dimensions may not be suitable either, due to the requirement of long time series. In realistic scenarios, persisting the memory accurately for dozens of periods can be sufficient, without the requirement for long-term accuracy.

Our solution is to train another RC network (classifier RC) to perform the short-term classification task to determine which

https://doi.org/10.1038/s41467-024-49190-4

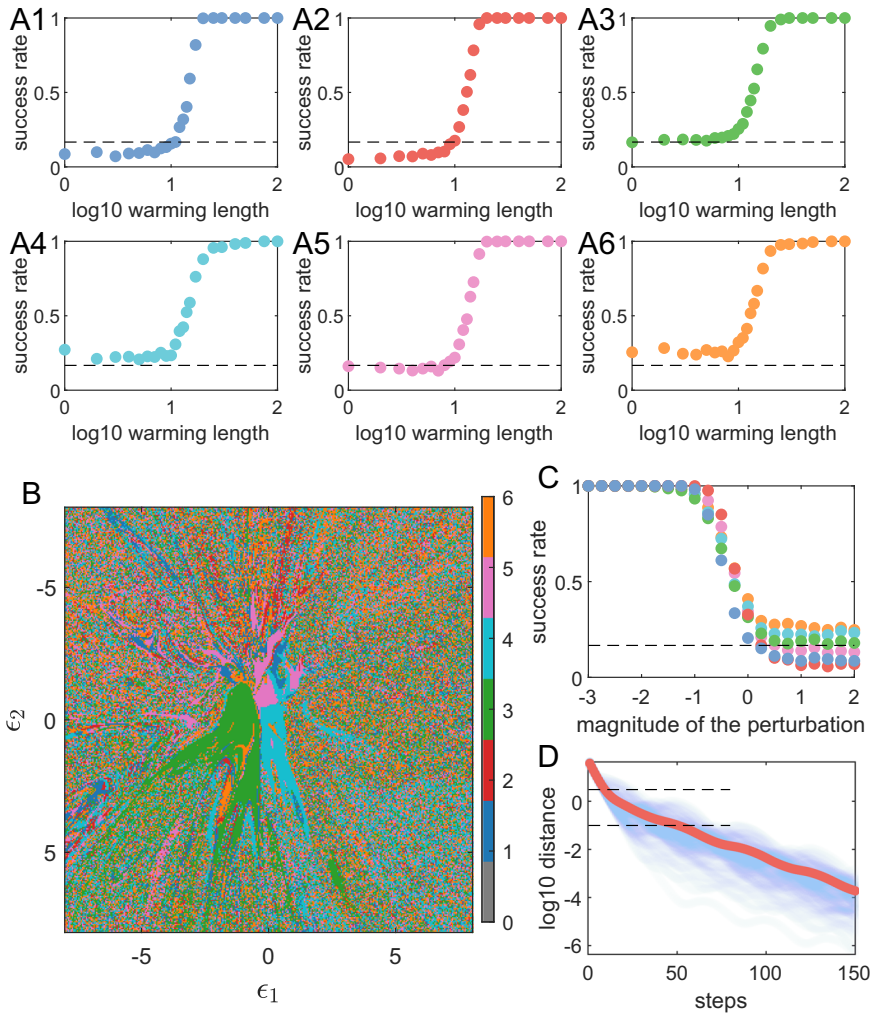

Fig. 5 | Index-free memory recalling. A1–A6 Success rate of memory retrieval of the six attractors in Fig. 1(**B**) versus the length of the warming cue. In the 3D phase space of the original dynamical systems, these attractors are located in approximately the same region and are overlapping. **B** A 2D projection of a typical basin structure of the reservoir dynamical network, where different colors represent the initial conditions leading to different memorized attractors. The central regions have relatively large contiguous areas of uniform color, while the basin structure is fragmented in the surrounding regions. **C** Success rate for the memory reservoir network to keep the desired attractor versus the magnitude of random perturbation. The two plateau regions with ~100% and 1/6 success rates correspond to the two typical basin features shown in panel (**B**). **D** Distance (averaged over an ensemble) between the dynamical state of the reservoir neural network and that of the target attractor versus the cue duration during warming. The cyan curves are obtained from 100 random trials, with the red curve as the average. The two horizontal black dashed lines correspond to the perturbation magnitude of $10^{0.5}$ and −1 in (**C**), and their intersection points with the red curve represent the warming lengths of 10 and 50-time steps, respectively, which are consistent with the transition regions in (**A**), suggesting a connection between the basin structure and the retrieval behavior.

memorized states have been recalled or, if no successful recall has been made at all, with high classification accuracy. We have tested its performance on more than a thousand trials and found only 6 trials out of 1,200 trials of deviation from a human labeler (details in SI), which are intrinsically ambiguous to classify. The classifier RC is also used in the control scheme for index-based memory RC, as shown in Fig. 3(D). The classifier RC provides a tool to understand the retrieval process, e.g., the type of driving signals that can be used to recall a desired memorized attractor. The basic idea is that the cues serve as a kind of reminder of the attractor to be recalled, so a short period of the time series from the attractor serves the purpose. Other cue signals are also possible, such as those based on partial information of the desired attractor to recall the whole or using noisy signals for warming.

We use the six different attractors in Fig. 1(B) as the target states, all located in a three-dimensional phase space. They do not live in distinct regions but have significant overlaps. However, the corresponding hidden states of the reservoir neural network, because of its much higher-dimensional phase space, can possibly live in distinct regions. Using short-term time series of the target attractor as a cue or warming signal to recall it, we can achieve a near 100% success rate of attractor recalling when the length of the warming signal exceeds some critical value, as shown in Fig. 5(A). Before injecting a warming signal, the initial state of the neural network is set to be random, so the success rate should be about 1/6 without the cue. As a cue is applied and its length increases, a relatively abrupt transition to near 100% success rate occurs for all six stored attractors. The remarkable success rate of retrieval suggests that all the six attractors can be trained to successfully coexist in this one high-dimensional dynamical system (i.e., the memory RC) with each of its own separate basin of attraction in the phase space of the neural network hidden state r. The finding that various attractors, despite their very different properties—ranging from being periodic or chaotic, with varying periods or maximal Lyapunov exponents, among others - all share nearly identical thresholds for warming length, also suggests that this threshold is more characteristic of the reservoir computer itself rather than the target states.

https://doi.org/10.1038/s41467-024-49190-4

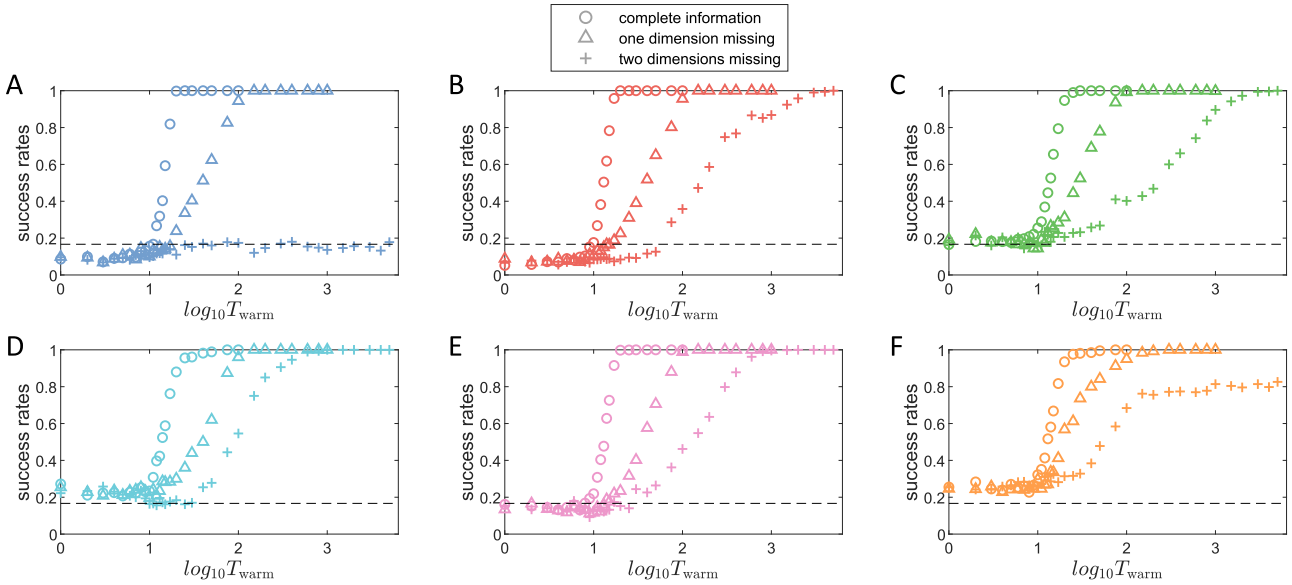

Fig. 6 | Attractor retrieval with partial cues in index-free reservoir-computing memory. A–F Shown is the success rate of memory retrieval of the six different chaotic attractors arising from 3D dynamical systems, respectively, with (i) full 3D cues, (ii) 2D cues with one dimension missing, and (iii) 1D cues with two dimensions missing. For 2D cues, a longer warming time is needed to achieve 100% success rate. For 1D cues, in most cases 100% success rate can be achieved with a longer warming time, but it can also occur that the perfect success rate cannot be achieved, e.g., (**A**), (**F**).

It is possible to arbitrarily pick up a "menu" of the attractors and train them to coexist in a memory RC as one single dynamical system. A question is: what are the typical structures of the basins of attraction of the memorized attractors? An example of the 2D projection of the basin structure is shown in Fig. 5(B). In a 2D plane, there are open areas of different colors, corresponding to the basins of different attractors, which are critical for the stability of the stored attractors. More specifically, when the basin of attraction of a memorized attractor contains an open set, it is unlikely for a dynamical trajectory from the attractor to be "kicked" out of the basin by small perturbations, ensuring stability. Figure 5(C) shows the probability (success rate) for a dynamical trajectory of the reservoir network to stay in the basin of a specific attractor after being "kicked" by a random perturbation. For a small perturbation, the probability is approximately one, indicating that the trajectory will always stay near the original (correct) attractor, leading essentially to zero retrieval error in the dynamical memory. Only when the perturbation is sufficiently large will the probability decrease to the value of 1/6, signifying random transition among the attractors and, consequently, failure of the system as a memory. We note that there are quite recent works on basin structures in a high-dimensional Kuramoto model where a large number of attractors coexist[61], which bears similar patterns to the ones observed in our memory RCs. Further investigation is required to study if these patterns can be truly generic across different dynamical systems.

The echo state property of a reservoir computer stipulates that a trajectory from an attractor in its original phase space corresponds to a unique trajectory in the high-dimensional phase space of the RNN in the hidden layer[11,62–64]. That is, the target attractor is be embedded in the RC hidden space. Figure 5(D) shows how the RC state approaches this embedded target state during the warming by the cue. This panel shows the distances (cyan curves) between each of the 100 trajectories of the reservoir neural network and the embedded memorized target attractor versus the cue duration during warming. (The details of how this distance is calculated are given in SI.) The ensemble-averaged distance (the red curve) decreases approximately exponentially with the cue length, indicating that the RC rapidly approaches the memory state's embedding.

The results in Fig. 5 suggest the following picture. The basin structure in Fig. 5(B) indicates that, when a trajectory approaches the target attractor, it usually begins in a riddled-like region that contains points belonging to the basins of different memorized attractors before landing in the open area containing the target attractor. The result in Fig. 5(C) can be interpreted, as follows. The recall success rate when the RC is away from the open areas and still in the riddled-like region is almost purely random (1/*K*). In an intermediate range of distance where two types of regions are mixed (as the open areas do not appear to have a uniform radius), the recall success rate increases rapidly before entering the pure open region and reaches 100% success rate. This is further verified by Fig. 5(D), where the two horizontal black dashed lines indicate the two distances that equal the magnitudes of perturbation in Fig. 5(C) under which the rapid decline of success rate begins and ends. That is, the interval between these two black dashed lines in Fig. 5(D) is the interval where the RC dynamical state traverses the mixed region. The cue lengths at the two ends of the curve in Fig. 5(D) in this interval are 10 and 50 time steps, which agree well with the transition interval in Fig. 5(A). Taken together, during the warming by the cue, the memory RC begins from the riddled-like region travels through a mixed region and finally reaches the open areas. This journey is reflected in the changes in recall success rate in Fig. 5(A).

It is worth studying if partial information can also successfully recall memorized states in an index-free memory RC. In particular, for the results in Fig. 5(A1–A6), the cue signals used to retrieve any stored chaotic attractor have the full dimensions as that of the original dynamical system that generates the attractor. What if the cues are partial with some missing dimension? For example, if the attractor is from a 3D system, then a full cue signal has three components. If one is missing, the actual input cue signal is 2D. However, since the reservoir network still has three input channels, the missing component can be compensated by the corresponding output component of the memory RC as a feedback loop. (This rewiring scheme was suggested previously[50].) Figure 6 shows, for the six attractors in Fig. 5(A1–A6), the success rate of retrieval versus the cue length or warming signal, for three distinct cases: full 3D cue signals, partial 2D cue signals with one missing dimension, and partial 1D cue signals with two missing dimensions. It can be seen that 100% success rates can still be achieved in most scenarios, albeit longer signals are required. The thresholds of cue length, where the success rate begins to increase significantly

https://doi.org/10.1038/s41467-024-49190-4

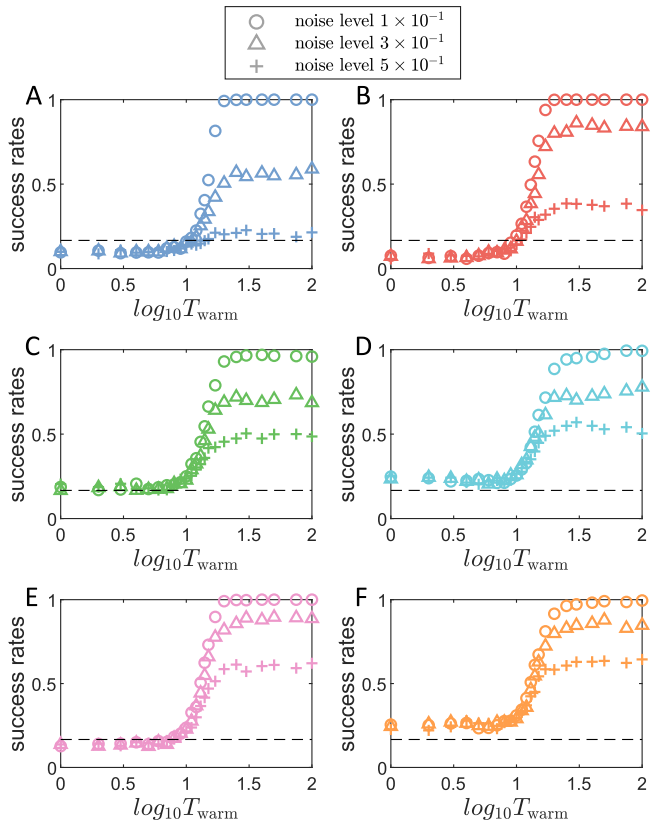

Fig. 7 | Attractor retrieval with noisy cues from index-free reservoir memories. **A**–**F** The success retrieval rate of the six distinct chaotic attractors in Fig. 1(**B**) (from left to right), respectively, versus the cue length.

larger than a random recall (around 1/6), are postponed to a similar degree for all the 2D cases among different attractors. These thresholds are also postponed to another similar degree for all the 1D cases. This result further suggests that the threshold of cue length is a feature of the RC structure and properties rather than a feature of the specific target attractor, with the same way of rewiring feedback loops yielding the same thresholds. (A discussion taking into account all the possible combinations of missed dimensions is provided in SI.)

A further question is: what if the warming signals for the index-free reservoir-computing memory contain random errors or are noisy? Figure 7 shows the retrieval success rate versus the cue length for three different levels of Gaussian white noise in the 3D cue signal. For all six chaotic attractors, a 100% success rate can still be achieved for relatively small noise, but the achievable success rate will decrease as the noise level increases. Different from the scenarios with partial dimensionality, the threshold of required cue length where the success rate jumps significantly from random memory retrieval (1/6) remains the same with different noise levels. We conclude that noisy cues can lower the saturated success rate, while partial information (with rewired feedback loops) can slow down the transition process from random retrieval to saturation in success rate and sometimes lower the saturated success rate.

Discussion

Traditional neural network models of artificial associative memories, such as the celebrated Hopfield neural networks, are designed to store and retrieve static patterns. To memorize and recall complex dynamical patterns such as chaotic attractors, machine learning based on the reservoir-computing type of RNNs is appropriate because of their ability to produce closed-loop, self-dynamical evolution. In a typical reservoir machine, the intrinsic dynamics are produced by a complex network of a large number of nonlinear nodes and are high-dimensional. Suppose the attractors to be stored and retrieved, even if they are chaotic, come from relatively low-dimensional dynamical systems. It is then possible for the high-dimensional reservoir system to "accommodate" the attractors in different regions of the phase space. Through appropriate training, each target attractor to be memorized can be made to live in a subregion with its own basin of attraction in the sense that there exists a dynamical invariant set in the subregion which, when recalled, will generate the target low-dimensional attractor, making reservoir-computing based memory possible. The reservoir memory system so designed is dynamic: the intrinsic high-dimensional state vector or the trajectories on the distinct dynamical invariant sets evolve in time continuously. This is the general principle underlying our present work.

The workings of our reservoir-computing-based memory consist of two phases: training for storing the attractors and testing to retrieve any desired attractor. During the training phase, time-series signals from all the attractors to be memorized are used as input to the reservoir neural network to determine certain connection weights in the neural architecture. Successful training serves to place different attractors in different regions of the high-dimensional phase space of the reservoir dynamical network, where each attractor corresponds to a dynamical invariant set, together with its basin of attraction, in some regions of the phase space. After training, the output vector of the reservoir system becomes its input, leaving the system in its own closed-loop dynamical evolution. To recall or retrieve an attractor, either an index-based scheme labeling different attractors or certain cue signals from the attractor to be recalled can be used to drive the dynamical evolution of the reservoir network to output a dynamical trajectory from the invariant set corresponding to the target attractor. The index-based scheme essentially defines a location-addressable memory, while cue-signal-based recalling effectively turns the reservoir-computing system into a content-addressable memory. We demonstrated that, for the location-addressable scenario, the system can memorize a large number of dynamical attractors, including chaotic attractors. Various scaling relations were uncovered between the number of attractors that can be memorized and the size of the reservoir neural network. For the content-addressable scenario, the stored attractors can be recalled using relatively short cue signals even when a certain dimension of the cue signal is missing, and there is noise.

The learning scenario in our work is batch learning with reservoir computing. We have demonstrated that it is computationally efficient and can successfully memorize and recall hundreds of dynamical attractors. The learning scheme is different from classic sequential learning in neuropsychology and in some machine-learning applications, where training is done by one memory state after another[65]. A difficulty with sequential learning is the possible occurrence of catastrophic forgetting[66], where the capability of performing some previously learned tasks can diminish due to changes in the network weights. There were methods for mitigating catastrophic learning[66–69], but developing RNN-based memory for complex dynamical attractors through sequential learning remains to be an open problem.

Itinerancy between attractor states in neural systems is a phenomenon that has attracted much attention[70,71]. As there is multi-functionality in our memory RCs, it is possible that such phenomena can emerge in our systems. There are three different ways of observing itinerancy, which are induced by a fluctuating input signal, transient behaviors, or noises. In Fig. 2(A), we have demonstrated itinerancy among different states under a fluctuating input signal. In an index-free memory RC, we can also observe itinerancy among transient states, with an example shown in SI in Fig. S5. However, how to properly train the memory RC to have the desired itinerancy behavior among transients needs to be investigated. For a real-world neural network, either a biological one or one realized by a physical system (e.g., by analog electronics[72], by a photonic system[73], or by morphological computing

on soft materials[74]), noises on the neurons are usually inevitable. In SI, we show how noises on the neurons in memory reservoir computers can result in spontaneous itinerancy among the attractor states. We hope our results, as dynamical models of itinerancy, can provide useful insights.

The focus of this work is on applying reservoir computing to store and recall a set of countable patterns. In the real brain, a continuum of memory states can exist[41,75]. For attractors arising from a single dynamical system, previous works on adaptable reservoir computing proposed using a continuous bifurcation-parameter channel to anticipate the future dynamical states of the target system[29–33,48], with training based on time series from a small number of parameter values. In Fig. 4(A), the "bifurcation" task sheds light on the application of developing reservoir computers to store and retrieve a continuum of attractors from different dynamical systems.

Methods

Several different types of dynamical states are used as the target states for the memory RCs to memorize. They include periodic or chaotic attractors generated from numerical integration of corresponding nonlinear ordinary differential equations, periodic trajectories defined in explicit forms as functions of time, and trajectories processed from short videos. In the latter two cases, the original trajectories do not come from dynamical systems, but they can still be trained as attractors in our reservoir computers.

The six attractors

The six attractors in Fig. 1(B), Fig. 2(A–E), Fig. 3(A), Fig. 5, Fig. 6 and Fig. 7 are: (1) a Lissajous trajectory, (2) a periodic attractor from a Sprott system[53], (3) the classic, chaotic Lorenz attractor, (4) the classic chaotic Rössler attractor (5) a chaotic attractor from a food-chain system[76], and (6) a chaotic attractor from the Hindmarsh-Rose neuron system[77]. The detailed equations and definitions of these memory attractors used in this work are given in SI.

The 16 Sprott attractors

All 16 chaotic attractors are illustrated in Figs. 1(C) and 2(G, H, I) are generated from the Sprott systems[53]. The time step for numerical integration is $dt = 0.01$, and the time step (temporal resolution) used in the final time series for training and testing is $\Delta t = 0.1$. All three dimensions are memorized by the memory RCs. These data are normalized such that the time series for each dimension has zero mean and unit standard deviation. The time scales of these attractors have a similar order of magnitude, where each oscillation cycle requires 30–60 time steps. If the natural time scales of the attractors to be memorized are drastically different, introducing heterogeneous leakages[78] or time delay[72] into the dynamics of the artificial neuron in the reservoir network can be effective for achieving the training goal.

Dataset #1

To obtain the scaling law between the number K of attractors that can be memorized and the size N of the reservoir network, hundreds of distinct attractors are needed to ensure at least two orders of magnitude of variation in K. The actual number of the attractors in this pool should be larger than the largest number K used to calculate the scaling law to reduce statistical fluctuations caused by some special features of certain attractors. We set out to generate 10,000 distinct attractors. To find such a large pool of distinct attractors is extremely challenging. Our procedure is as follows. We sample 2–5 random points in a 2D plane with 200 time steps and a constant height corresponding to the interval of $[-1, 1]$. We then fit a fourth-order Fourier series to these random points to obtain a continuous curve with a period of 200-time steps. Each dimension of any attractor is generated independently, and all the 10,000 attractors are also generated independently.

ALOI videos

This dataset[56] contains 1,000 short videos of rotating objects on a black background. Each video has 72 frames and forms a loop as the object rotates an entire circle back to the initial state. Each frame is a 384 times 288-pixel gray-scale image. We only use the odd number frames as our training and testing data to make our task easier and save computational time. We then perform a dimension reduction based on principal component analysis, as most pixels in the video are just the black background, and most of the remaining pixels are highly correlated. We take the first two principal components as the training and testing data. Therefore, each dynamical state is a two-dimensional periodic state with a period of 36 time steps.

The "bifurcation" task dataset

All the states are generated by the following dynamical equations:

$$\frac{dR}{dt} = R\left(1 - \frac{R}{K_f}\right) - \frac{x_c y_c C R}{R + R_0}, \tag{1}$$

$$\frac{dC}{dt} = x_c C\left[\frac{y_c R}{R + R_0} - 1\right] - \frac{x_p y_p P C}{C + C_0}, \tag{2}$$

$$\frac{dP}{dt} = x_p P\left(\frac{y_p C}{C + C_0} - 1\right), \tag{3}$$

with $x_c = 0.4, y_c = 2.009, x_p = 0.08, y_p = 2.876, R_0 = 0.16129$ and $C_0 = 0.5$. The differences among the states are created by varying the value of K_f in the interval of $[0.92, 1]$, where multiple bifurcations happen, and there are periodic regions and chaotic regions mixed in this interval. A bifurcation diagram of this system in this interval can be found in Fig. 2 of ref. 29. The time step (temporal resolution) of the final time series used as training and testing data is $\Delta t = 1$. All three dimensions are memorized by the memory RC.

Data availability

The data generated in this study, including both the training time series (as the dynamical states to be memorized) and weights of the reservoir computers, can be found in this OSF repository: https://osf.io/yxm2v/[79].

Code availability

The codes used in this paper can be found in the repository: https://github.com/lw-kong/Long-Term-Memory-in-RC[80].

References

1. Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci. (USA)* **79**, 2554–2558 (1982).
2. Hopfield, J. J. & Tank, D. W. Computing with neural circuits: a model. *Science* **233**, 625–633 (1986).
3. Aoyagi, T. Network of neural oscillators for retrieving phase information. *Phys. Rev. Lett.* **74**, 4075–4078 (1995).
4. Aonishi, T. Phase transitions of an oscillator neural network with a standard hebb learning rule. *Phys. Rev. E* **58**, 4865–4871 (1998).
5. Aonishi, T., Kurata, K. & Okada, M. Statistical mechanics of an oscillator associative memory with scattered natural frequencies. *Phys. Rev. Lett.* **82**, 2800–2803 (1999).
6. Yoshioka, M. & Shiino, M. Associative memory storing an extensive number of patterns based on a network of oscillators with distributed natural frequencies in the presence of external white noise. *Phys. Rev. E* **61**, 4732–4744 (2000).
7. Nishikawa, T., Lai, Y.-C. & Hoppensteadt, F. C. Capacity of oscillatory associative-memory networks with error-free retrieval. *Phys. Rev. Lett.* **92**, 108101 (2004).

8. Nishikawa, T., Hoppensteadt, F. C. & Lai, Y.-C. Oscillatory associative memory network with perfect retrieval, etrieval. *Phys. D.* **197**, 134–148 (2004).
9. Radhakrishnan, A., Belkin, M. & Uhler, C. Overparameterized neural networks implement associative memory. *Proc. Nat. Acad. Sci. (USA)* **117**, 27162–27170 (2020).
10. Ott, E.*Chaos in Dynamical Systems*. second edn (Cambridge University Press, Cambridge, UK, 2002).
11. Jaeger, H. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn., Ger.: Ger. Natl Res. Cent. Inf. Technol. GMD Tech. Rep.* **148**, 13 (2001).
12. Mass, W., Nachtschlaeger, T. & Markram, H. Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neur. Comp.* **14**, 2531–2560 (2002).
13. Jaeger, H. & Haas, H. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. *Science* **304**, 78–80 (2004).
14. Haynes, N. D., Soriano, M. C., Rosin, D. P., Fischer, I. & Gauthier, D. J. Reservoir computing with a single time-delay autonomous Boolean node. *Phys. Rev. E* **91**, 020801 (2015).
15. Larger, L. et al. High-speed photonic reservoir computing using a time-delay-based architecture: million words per second classification. *Phys. Rev. X* **7**, 011015 (2017).
16. Pathak, J., Lu, Z., Hunt, B., Girvan, M. & Ott, E. Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data. *Chaos* **27**, 121102 (2017).
17. Lu, Z. et al. Reservoir observers: Model-free inference of unmeasured variables in chaotic systems. *Chaos* **27**, 041102 (2017).
18. Pathak, J., Hunt, B., Girvan, M., Lu, Z. & Ott, E. Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach. *Phys. Rev. Lett.* **120**, 024102 (2018).
19. Carroll, T. L. Using reservoir computers to distinguish chaotic signals. *Phys. Rev. E* **98**, 052209 (2018).
20. Nakai, K. & Saiki, Y. Machine-learning inference of fluid variables from data using reservoir computing. *Phys. Rev. E* **98**, 023111 (2018).
21. Roland, Z. S. & Parlitz, U. Observing spatio-temporal dynamics of excitable media using reservoir computing. *Chaos* **28**, 043118 (2018).
22. Griffith, A., Pomerance, A. & Gauthier, D. J. Forecasting chaotic systems with very low connectivity reservoir computers. *Chaos* **29**, 123108 (2019).
23. Jiang, J. & Lai, Y.-C. Model-free prediction of spatiotemporal dynamical systems with recurrent neural networks: Role of network spectral radius. *Phys. Rev. Res.* **1**, 033056 (2019).
24. Tanaka, G. et al. Recent advances in physical reservoir computing: a review. *Neu. Net.* **115**, 100–123 (2019).
25. Fan, H., Jiang, J., Zhang, C., Wang, X. & Lai, Y.-C. Long-term prediction of chaotic systems with machine learning. *Phys. Rev. Res.* **2**, 012080 (2020).
26. Zhang, C., Jiang, J., Qu, S.-X. & Lai, Y.-C. Predicting phase and sensing phase coherence in chaotic systems with machine learning. *Chaos* **30**, 083114 (2020).
27. Klos, C., Kossio, Y. F. K., Goedeke, S., Gilra, A. & Memmesheimer, R.-M. Dynamical learning of dynamics. *Phys. Rev. Lett.* **125**, 088103 (2020).
28. Chen, P., Liu, R., Aihara, K. & Chen, L. Autoreservoir computing for multistep ahead prediction based on the spatiotemporal information transformation. *Nat. Commun.* **11**, 4568 (2020).
29. Kong, L.-W., Fan, H.-W., Grebogi, C. & Lai, Y.-C. Machine learning prediction of critical transition and system collapse. *Phys. Rev. Res.* **3**, 013090 (2021).
30. Patel, D., Canaday, D., Girvan, M., Pomerance, A. & Ott, E. 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).
31. Kim, J. Z., Lu, Z., Nozari, E., Pappas, G. J. & Bassett, D. S. Teaching recurrent neural networks to infer global temporal structure from local examples. *Nat. Mach. Intell.* **3**, 316–323 (2021).
32. Fan, H., Kong, L.-W., Lai, Y.-C. & Wang, X. Anticipating synchronization with machine learning. *Phys. Rev. Resesearch* **3**, 023237 (2021).
33. Kong, L.-W., Fan, H., Grebogi, C. & Lai, Y.-C. Emergence of transient chaos and intermittency in machine learning. *J. Phys. Complex.* **2**, 035014 (2021).
34. Bollt, E. On explaining the surprising success of reservoir computing forecaster of chaos? the universal machine learning dynamical system with contrast to var and dmd. *Chaos* **31**, 013108 (2021).
35. Gauthier, D. J., Bollt, E., Griffith, A. & Barbosa, W. A. Next generation reservoir computing. *Nat. Commun.* **12**, 1–8 (2021).
36. Carroll, T. L. Optimizing memory in reservoir computers. *Chaos* **32**, https://doi.org/10.48550/arXiv.2201.01605 (2022).
37. Zhai, Z.-M. et al. Model-free tracking control of complex dynamical trajectories with machine learning. *Nat. Commun.* **14**, 5698 (2023).
38. Yan, M. et al. Emerging opportunities and challenges for the future of reservoir computing. *Nat. Commun.* **15**, 2056 (2024).
39. Kim, J. Z. & Bassett, D. S. A neural machine code and programming framework for the reservoir computer. *Nat. Mach. Intell.* **5**, 622–630 (2023).
40. French, R. M. Catastrophic forgetting in connectionist networks. *Trends Cogn. Sci.* **3**, 128–135 (1999).
41. Chaudhuri, R. & Fiete, I. Computational principles of memory. *Nat. Neurosci.* **19**, 394–403 (2016).
42. James, W. *The Principles of Psychology*, vol. 1 (Cosimo, Inc., 2007).
43. Scoville, W. B. & Milner, B. Loss of recent memory after bilateral hippocampal lesions. *J. Neurol. Neurosurg. Psychi.* **20**, 11–21 (1957).
44. Tetzlaff, C., Kolodziejski, C., Markelic, I. & Wörgötter, F. Time scales of memory, learning, and plasticity. *Biol. Cybern.* **106**, 715–726 (2012).
45. Bailey, C. H., Kandel, E. R. & Harris, K. M. Structural components of synaptic plasticity and memory consolidation. *Cold Spring Harb. Pers. Biol.* **7**, a021758 (2015).
46. Shiffrin, R. M. & Atkinson, R. C. Storage and retrieval processes in long-term memory. *Psychol. Rev.* **76**, 179 (1969).
47. Inoue, K., Nakajima, K. & Kuniyoshi, Y. Designing spontaneous behavioral switching via chaotic itinerancy. *Sci. Adv.* **6**, eabb3989 (2020).
48. Kong, L.-W., Weng, Y., Glaz, B., Haile, M. & Lai, Y.-C. Reservoir computing as digital twins for nonlinear dynamical systems. *Chaos: an Interdisciplinary Journal of Nonlinear Science* **33**, 033111 (2023).
49. Ceni, A., Ashwin, P. & Livi, L. Interpreting recurrent neural networks behaviour via excitable network attractors. *Cogn. Comp.* **12**, 330–356 (2020).
50. Lu, Z. & Bassett, D. S. Invertible generalized synchronization: a putative mechanism for implicit learning in neural systems. *Chaos* **30**, 063133 (2020).
51. Flynn, A., Tsachouridis, V. A. & Amann, A. Multifunctionality in a reservoir computer. *Chaos* **31**, 013125 (2021).
52. Flynn, A. et al. Exploring the limits of multifunctionality across different reservoir computers. In *2022 International Joint Conference on Neural Networks (IJCNN)*, 1–8 (IEEE, 2022).
53. Sprott, J. C. Some simple chaotic flows. *Phys. Rev. E* **50**, R647–R650 (1994).
54. Raaijmakers, J. G. & Shiffrin, R. M. Search of associative memory. *Psychological Rev.* **88**, 93 (1981).
55. Unsworth, N. Exploring the retrieval dynamics of delayed and final free recall: further evidence for temporal-contextual search. *J. Mem. Lang.* **59**, 223–236 (2008).
56. Geusebroek, J.-M., Burghouts, G. J. & Smeulders, A. W. The amsterdam library of object images. *Int. J. Computer Vis.* **61**, 103–112 (2005).

57. Röhm, A., Gauthier, D. J. & Fischer, I. Model-free inference of unseen attractors: reconstructing phase space features from a single noisy trajectory using reservoir computing. *Chaos* **31**, 103127 (2021).
58. Roy, M. et al. Model-free prediction of multistability using echo state network. *Chaos* **32**, 101104 (2022).
59. Rulkov, N. F., Sushchik, M. M., Tsimring, L. S. & Abarbanel, H. D. I. Generalized synchronization of chaos in directionally coupled chaotic systems. *Phys. Rev. E* **51**, 980–994 (1995).
60. Lymburn, T., Walker, D. M., Small, M. & Jüngling, T. The reservoir's perspective on generalized synchronization. *Chaos* **29**, 093133 (2019).
61. Zhang, Y. & Strogatz, S. H. Basins with tentacles. *Phys. Rev. Lett.* **127**, 194101 (2021).
62. Jaeger, H. Echo state network. *Scholarpedia* **2**, 2330 (2007).
63. Yildiz, I. B., Jaeger, H. & Kiebel, S. J. Re-visiting the echo state property. *Neu. Net.* **35**, 1–9 (2012).
64. Hart, A., Hook, J. & Dawes, J. Embedding and approximation theorems for echo state networks. *Neu. Net.* **128**, 234–247 (2020).
65. Botvinick, M. M. & Plaut, D. C. Short-term memory for serial order: a recurrent neural network model. *Psychol. Rev.* **113**, 201–233 (2006).
66. Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A. & Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv Preprint arXiv:1312.6211* (2013).
67. Kirkpatrick, J. et al. Overcoming catastrophic forgetting in neural networks. *Proc. Nat. Acad. Sci. (USA)* **114**, 3521–3526 (2017).
68. Coop, R. & Arel, I. Mitigation of catastrophic forgetting in recurrent neural networks using a fixed expansion layer. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 1–7 (IEEE, 2013).
69. Kobayashi, T. & Sugino, T. Continual learning exploiting structure of fractal reservoir computing. In *Artificial Neural Networks and Machine Learning–ICANN 2019: Workshop and Special Sessions: 28th International Conference on Artificial Neural Networks, Munich, Germany, Proceedings 28*, 35–47 (Springer, 2019).
70. Tsuda, I. Chaotic itinerancy and its roles in cognitive neurodynamics. *Curr. Opin. Neurobiol.* **31**, 67–71 (2015).
71. Miller, P. Itinerancy between attractor states in neural systems. *Curr. Opin. Neurobiol.* **40**, 14–22 (2016).
72. Appeltant, L. et al. Information processing using a single dynamical node as complex system. *Nat. Commun.* **2**, 1–6 (2011).
73. Woods, D. & Naughton, T. J. Photonic neural networks. *Nat. Phys.* **8**, 257–259 (2012).
74. Nakajima, K., Hauser, H., Li, T. & Pfeifer, R. Information processing via physical soft body. *Sci. Rep.* **5**, 10487 (2015).
75. Kim, S. S., Rouault, H., Druckmann, S. & Jayaraman, V. Ring attractor dynamics in the drosophila central brain. *Science* **356**, 849–853 (2017).
76. Blasius, B., Huppert, A. & Stone, L. Complex dynamics and phase synchronization in spatially extended ecological systems. *Nature* **399**, 354–359 (1999).
77. Hindmarsh, J. L. & Rose, R. M. A model of neuronal bursting using three coupled first order differential equations. *Proc. R. Soc. Lon. Ser. B Biol. Sci.* **221**, 87–102 (1984).
78. Tanaka, G., Matsumori, T., Yoshida, H. & Aihara, K. Reservoir computing with diverse timescales for prediction of multiscale dynamics. *Phys. Rev. Res.* **4**, L032014 (2022).
79. Kong, L.-W. Reservoir-computing based associative memory and itinerancy for complex dynamical attractors https://doi.org/10.17605/OSF.IO/YXM2V (2024).
80. Kong, L.-W. Codes. GitHub: https://github.com/lw-kong/Long-Term-Memory-in-RC (2024).

Acknowledgements

We thank Dr. J.-J. Jiang for stimulating discussions during the initial phase of the study. We also thank Dr. Andrew Flynn for insightful discussions and comments. This work was supported by the Air Force Office of Scientific Research under Grant No. FA9550-21-1-0438 (to Y.-C.L.). This work was also supported by the Eric and Wendy Schmidt AI in Science Postdoctoral Fellowship, a Schmidt Futures program (to L.-W.K.), and by the U.S. Army Research Institute under Award No. W911NF2310300 (to G.A.B.).

Author contributions

L.-W. K., G.A.B., and Y.-C. L. designed the research project, the models, and methods. L.-W. K. performed the computations. L.-W. K., G.A.B., and Y.-C. L. analyzed the data. L.-W. K. and Y.-C. L. wrote and edited the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41467-024-49190-4.

Correspondence and requests for materials should be addressed to Ying-Cheng Lai.

Peer review information *Nature Communications* thanks Jason Kim, and the other, anonymous, reviewer(s) for their contribution to the peer review of this work. A peer review file is available.

Supplementary Information for

Reservoir-computing based associative memory and itinerancy for complex dynamical attractors

Ling-Wei Kong, Gene A. Brewer, and Ying-Cheng Lai

Corresponding author: Ying-Cheng Lai (Ying-Cheng.Lai@asu.edu)

CONTENTS

Supplementary Note 1. RESERVOIR COMPUTING

We employ two types of RC networks for associative memory of complex dynamical attractors: index-based and index-free. Their architectures are not identical, but their training and testing methods (for storing and retrieving attractors) bear only minor differences. Therefore, in the following, we will describe the training and testing methods of both index-based and index-free schemes together. The codes for training and predicting with both architectures can be found at Ref. [1].

As shown in Fig. 1 in the main text, a reservoir computer consists of three layers: an input layer, a hidden recurrent layer, and an output layer. The major advantage of reservoir computing compared with most RNN architectures is that not all the weights in the layers need to be trained, and the training usually does not require backpropagation. Rather, a regularized linear regression suffices to train only the weights W_{out} of the output layer. This training scheme not only significantly lowers the training computational cost, but also helps mitigate issues such as catastrophic forgetting and vanishing/exploding gradient. These features make reservoir computing particularly suitable for our multifunctional tasks, which require training with a large set of training data sequences with distinct behaviors without forgetting.

The training of a reservoir neural network for memorizing dynamical attractors, whether index-based or index-free, can be divided into three steps. In the first step, we generate the input and recurrent hidden layers. They are generated with random values for the entries and fixed once generated. Specifically, in an index-based RC, there are two input matrices: the state input matrix W_{u} that projects the low-dimensional state vector $\mathbf{u}$ characterizing the target attractor to be memorized to the hidden layer, and the index input matrix W_{index} that injects the index value p associated with each attractor to be memorized into the reservoir neural network. The entries of both W_{u} and W_{index} are randomly generated by a uniformed distribution in the interval $[-c_{\text{in}}, c_{\text{in}}]$. Here c_{in} is a hyperparameter to be optimized. When we input the index value $p(t)$, since the specific values of p can be chosen rather arbitrarily, we introduce a linear transformation of it $k_p(p(t) + b_p)$ before injecting to the RC network to make sure it is not way too large or way too small. Here k_p and b_p are two hyperparameters. For index-free RCs, we only have W_{u} in the input layer, and there is no $p(t)$, W_{index}, k_p, or b_p. The RNN in the hidden layer has N neurons connected with each other by a network called "the reservoir". This reservoir network can be represented by the matrix W_{r}, with N as the network size. A basic parameter characterizing the network connectivity is the probability that a random pair of nodes is connected. We write this connectivity coefficient as s_r, which is another hyperparameter. The reservoir network is usually a sparse network with a small s_r, as in all the RCs used in this study. This sparsity significantly reduces the computational costs in both training and testing. The reservoir network is directed and weighted, where all the connecting weights are initially independently generated by a uniform distribution in the interval of $[0, 1]$. The entire matrix W_{r} is then rescaled so that the network spectral radius ρ (another hyperparameter) equals the desired value we obtain through the hyperparameter optimization.

In the second step, which is often called the "echoing step", the time series of each attractor is the input vector signal $\mathbf{u}(t)$ into the input layer and generates a response vector signal $\mathbf{r}(t)$ of equal length in the hidden layer. We call this process as the echoing process as we are simply observing the echoing of the input training signals in hidden layer (the reservoir). For the index-based memory, the corresponding index value $p(t)$ (a scalar piecewise constant function) is simultaneously

injected into the hidden-layer network. The equation of iteration in this echoing step is as follows:

$$\begin{aligned}\mathbf{r}(t) = (1-\alpha)\mathbf{r}(t-\Delta t) + \alpha \tanh\{W_{\text{r}} \cdot \mathbf{r}(t-\Delta t) \\ + W_{\text{u}} \cdot [\mathbf{u}(t) + \sigma_{\text{train}}\xi(t)] + W_{\text{index}} k_p[p(t) + b_p]\},\end{aligned} \tag{S1}$$

where α is the leakage parameter, and Δt is the time step of network dynamical evolution. For index-free memory retrieval, the term $W_{\text{index}} k_p[p(t) + b_p]$ is absent. Training noise is applied to better stabilize the memory states [2], and we have $\xi(t)$ term in the iteration equation as a standard Gaussian white noise added to the input data. The hyperparameter σ_{train} controls the magnitude of this noise. Since we are training with multiple target states, we do the echoing process for each of them sequentially. At the beginning of the echoing process for each target state, $\mathbf{r}(t)$ is initiated with all zeros. We then have a 10-step washing-out period immediately after the initialization to exclude the transient behaviors. After this washing-out period, all the hidden state $\mathbf{r}(t)$ are recorded to be used later in the second step. At the end of this first step (echoing step), we should have gathered a huge collection of hidden state $\mathbf{r}(t)$ with a number of the number of target states multiplied by the subtraction of the training length by washing out length.

In the third step, a regularized linear regression is carried out between the target state and the reservoir hidden state $\mathbf{r}(t)$ we have collected from the first step. In our cases, as we want the RC network to learn the dynamic rules of the target states, the training target is the same as the training inputs but with one time step forward. The RC network is thus essentially trained to make a one-step-ahead prediction of the target state.

Compared with a more standard approach where only one attractor/state is trained, the trick with our approach (for both indexed and index-free schemes) is that we then concatenate the records of the hidden states $\mathbf{r}(t)$ from different target states together in the temporal dimension to form one (potentially very long) time series $R(t)$. The training target is processed in the same way, where $\mathbf{v}(t)$ from different states are concatenated in the temporal dimension to form one (also potentially very long) time series $V(t)$. This "concatenating in time" scheme has been used in several previous work in the context of reservoir computing [3–5] when more than one state/attractor is trained. It is pretty interesting that this simple method works at all, especially given that the different states it concatenates can have very different dynamical features, from simple periodic oscillations to various chaotic trajectories with different forms of nonlinearity. The high dimensionality of the RC system and the simplicity of linear regression allow this way of merging different dynamics into one RNN. This is particularly helpful for our tasks with multiple states to memorize as no forgetting issue would arise.

One final treatment before the actual linear regression is that we follow the trick from Ref. [6, 7] to take a square of the even rows/neurons of $R(t)$ to form $R'(t)$ to exclude undesired potential symmetries in the RC system. Then, finally, a ridge regression is performed between $V(t)$ and $R(t)$, by the following equation:

$$W_{\text{out}} = V \cdot R'^{T}(R' \cdot R'^{T} + \beta I)^{-1}, \tag{S2}$$

where β is the l-2 regularization coefficient, another hyperparameter of the reservoir computer, and I is an identity matrix. Now we have our readout matrix from the reservoir W_{out}, and the training is finished.

A network so trained can serve as a closed-loop dynamical system capable of generating arbitrarily long trajectories of a memorized attractor after a successful recall according to the following

iterative dynamical equations:

$$\mathbf{r}(t) = (1-\alpha)\mathbf{r}(t-\Delta t) + \alpha \tanh[W_{\text{r}} \cdot \mathbf{r}(t-\Delta t) + W_{\text{u}} \cdot \mathbf{u}(t) + W_{\text{index}} k_p(p(t)+b_p)], \tag{S3}$$

$$\mathbf{v}(t) = W_{\text{out}} \cdot \mathbf{r}'(t), \tag{S4}$$

$$\mathbf{v}(t) \to \mathbf{u}(t+\Delta t), \tag{S5}$$

where $\mathbf{v}(t)$ is the output of the reservoir computer and should be the target state we want if the retrieval is successful. In the loop of generating a trajectory, the output $\mathbf{v}(t)$ becomes the state input $\mathbf{u}(t+\Delta t)$ of the next time step so that the iteration can continue for an arbitrary length. (This does not necessarily mean this arbitrarily long prediction is always accurate.) Again, the term $W_{\text{index}} k_p(p(t)+b_p)$ will disappear if we are operating an index-free reservoir computer.

For reservoir computing, hyperparameter optimization is often necessary. The hyperparameters that need optimization are the leakage α, the regularization coefficient β of the linear regression in training, the scale of the input matrix c_{in}, the spectral radius ρ of W_{r}, and the reservoir network connectivity s_r, and the strength of training noise σ_{train}. We use a Bayesian optimization algorithm (*surrogateopt* in Matlab). In such an optimization process, we go through a loop of iterations. In each iteration, we test a set of hyperparameters' values and receive a validation result. The algorithm uses the results we collect from previous iterations to fit a performance landscape in the space of the hyperparameters and use such a fitting to guide our future search for optimal hyperparameters. After a maximum iteration number is reached, we stop the iteration and select the set of hyperparameters with the best validation performance that we have tested. Compared with this Bayesian approach, a random search is very inefficient as each iteration is independent and history information is not utilized. A grid search is also not applicable as we have many different hyperparameters, so the parameter space's dimensionality is too high.

Table S1 lists all the specific values of all the hyperparameters that we get from the optimization and use in this paper. With different tasks and different training approaches, we have multiple groups of reservoir computers. The reservoir computers in the same group use the same set of hyperparameters. To make things clear, we assign each group a name, which is shown in the first column of Tab. S1. The group Indexed #1 refers to the indexed memory RCs that are trained with the six different memory states shown in Fig. 1(B) in the main text and Fig. S1. Results that use this set are shown in Fig. 1(B), Figs. 2(A, B, C, D, E, F), and Fig. 3 in the main text. Results shown in Figs. S2, S7, S9, and S15 also use this set. The group Indexed #2 refers to the indexed memory RCs that are trained to have 16 Sprott chaotic states, as shown in Fig. 1(C) in the main text. Results that use this set are shown in Figs. 2(G, H, I) in the main text, Figs. S3, S4, and S8. For the results in Figs. 2(G, H, I) in the main text and Fig. S8, the training length is $T_{\text{train}} = 4,000$ steps for each attractor. and the network size N is $N = 2,000$. For the results in Figs. S3 and S4, the training length is $T_{\text{train}} = 5,000$ steps for each attractor, and the network size is $N = 3,000$. The group Indexed #3 refers to the indexed memory RCs that are used in generating some of the scaling laws. More specifically, it is used in the "one-hot coding" task, the "binary coding" task, the "separate W_{out}" task, and the "bifurcation task" in Fig. 4(A) in the main text. It is also used in all the curves in Figs. 4(B,C) in the main text, Figs. S10 and S11. The training length T_{train} is 1,000 steps for all these results except the "bifurcation" task where the training length T_{train} is 2,000 steps. The group Indexed #4 refers to the indexed memory RCs that are used in the "ALOI" task in Fig. 1(A) in the main text. The group Index-Free #1 refers to the index-free memory RCs that are trained with the

RC Group	N	T_{train}	ρ	c_{in}	α	$\log_{10}\beta$	s_r	$\log_{10}\sigma_{\text{train}}$	k_p	b_p
Indexed #1	1,000	6,000	0.78	0.85	0.37	-7.5	0.21	-3.1	1.12	-1.08
Indexed #2	2,000 or 3,000	4,000 or 5,000	0.39	0.91	0.64	-6.5	0.4	-3	3.3	-10
Indexed #3	-	1,000 or 2,000	0.39	0.91	0.64	-6.5	0.005	-3	-	-
Indexed #4	-	1,440	0.3	0.9	0.6	-6.5	0.005	-2	1	0
Index-Free #1	4,000	6,000	1.47	1.13	1	-6.4	0.19	-2.9	-	-
Index-Free #2	-	1,000	1.92	2.82	0.42	-7.1	0.0027	-4.5	-	-

TABLE S1. Hyperparameters of the reservoir computers used in this paper. The parts of results in the paper that used each RC group are discussed in the Methods section, with more details on the specific choices of training length T_{train} and reservoir size N. The training length T_{train} refers to the number of steps used in the training for each target state.

six different memory states shown in Fig. 1B) in the main text and Fig. S1. Results that use this set are shown in Figs. 5, 6, and 7 in the main text, as well as Figs. S12, S13, S14, and S16. The group Index-Free #1 refers to the index-free memory RCs used in the "index-free" task in Fig. 1(A) in the main text.

Supplementary Note 2. DETAILS OF THE DYNAMICAL ATTRACTORS TO BE MEMORIZED

We describe how the training and testing data for the target attractors to be memorized are generated. All the data used in work can be found under the 'data' folder in our GitHub repository [8].

The training data $\mathbf{u}(t) = [u_x(t), u_y(t), u_z(t)]^T$ for the six attractors in Fig. 1(B) in the main text (illustrated in Fig. S1), are generated as follows.

- A periodic Lissajous system with the frequency ratio as 1:3:5, where $\mathbf{u}(t)$ is generated by the following formulas with time step $\Delta t = 1$.

$$\begin{aligned} u_x(t) &= \sin(\pi t/100), \\ u_y(t) &= \sin(3\pi t/100 + \pi/2), \\ u_z(t) &= \sin(\pi t/20). \end{aligned} \tag{S6}$$

- A periodic attractor from the Sprott system [9] is generated by the following equations:

$$\begin{aligned} du_x/dt &= 0.3u_x + u_z, \\ du_y/dt &= u_x u_z - u_y, \\ du_z/dt &= -u_x + u_y. \end{aligned} \tag{S7}$$

  The time step (temporal resolution) in the training data is $\Delta t = 0.015$.

- The classic chaotic Lorenz attractor is generated from the equations

$$\begin{aligned} du_x/dt &= 10(u_y - u_x), \\ du_y/dt &= u_x(28 - u_z) - u_y, \\ du_z/dt &= u_x u_y - 8/3u_z. \end{aligned} \tag{S8}$$

  The time step (temporal resolution) in the training data is $\Delta t = 0.02$.

- The classic chaotic Rössler system is generated from

$$\begin{aligned} du_x/dt &= -u_y - u_z, \\ du_y/dt &= u_x + 0.2u_y, \\ du_z/dt &= u_z(u_x - 5.7) + 0.2. \end{aligned} \tag{S9}$$

  The time step (temporal resolution) in the training data is $\Delta t = 0.1$.

- A chaotic attractor from the food chain system [10] is generated by the following equations:

$$\begin{aligned} du_x/dt &= u_x - \frac{0.2u_x u_y}{1 + 0.05u_x}, \\ du_y/dt &= -u_y + \frac{0.2u_x u_y}{1 + 0.05u_x} - u_y u_z, \\ du_z/dt &= -10(u_z - 0.006) + u_y u_z. \end{aligned} \tag{S10}$$

  The time step (temporal resolution) in the training data is $\Delta t = 0.15$.

- A chaotic attractor from the Hindmarsh-Rose (HR) neuron system [11] is generated from the following equations:

$$\begin{aligned} du_x/dt &= u_y - u_x^3 + 3u_x^2 - u_z + 3.25, \\ du_y/dt &= 1 - 5u_x^2 - u_y, \\ du_z/dt &= 0.006(4(u_x + 8/5) - u_z). \end{aligned} \tag{S11}$$

  The time step (temporal resolution) in the training data is $\Delta t = 0.6$.

Supplementary Note 3. FIDELITY OF RECALLED ATTRACTORS IN THE LONG TERM

In the main text, we discuss how we evaluate the maximum Lyapunov exponents and the correlation dimensions of the reconstructed chaotic attractors in the retrieval phase to validate the fidelity of the reconstructed attractor accuracy. Here we provide more results on this point, by calculating the maximum Lyapunov exponents of the reconstructed attractors under different levels of training noise σ_{train}. The result is shown in Fig. S2. We show that there is an optimal training noise region where the maximum Lyapunov exponents of most of the reconstructed chaotic attractors agree well with the ground truth values.

Note that this "fidelity of recalled attractors in the long term" is a separate issue from achieving "long-term memory", although both expressions have something to do with a long time scale. It is also indeed true, though, that our framework satisfies both "long-term" criteria. The former criterion requires that once a memory state is recalled, it can persist for a long term without losing or deviating from the crucial dynamical features of the target memory state (such as the maximum Lyapunov exponent). The latter criterion requires that the dynamical information is stored in the weights and connections of the Rc network, not the hidden state, so that the memory states can be recalled with proper cues or other recalling methods, even with a random initial hidden state. Thus, a recalled state that only persists in the memory device for several periods may still be considered as "long-term memory". In other words, the "term" in the name "long-term memory" actually refers to the time *between* memorizing and recalling, not to the time length *of* recalling.

Supplementary Note 4. PERFORMANCE OF INDEX-BASED RESERVOIR MEMORY FOR MEMORIZING 16 CHAOTIC ATTRACTORS

In Fig. 1(C) in the main text, the 16 chaotic attractors to be memorized and the 2D index values are displayed. Here we provide the results of testing (retrieval), as shown in Fig. S3, where the blue and red trajectories are the ground truth and the outputs of reservoir memory, respectively. The length of the recalled trajectories can be arbitrarily long.

In Fig. S4, we test if a randomized assignment of the index values will harm the performance. We show that, in most cases, the reservoir computer can successfully recall the target memory states and accurately persist the state for at least four average periods. This result suggests that our approach can work on this dataset regardless of how the index values are assigned.

Supplementary Note 5. RESERVOIR-COMPUTING BASED ATTRACTOR CLASSIFIER

We use a classifier reservoir computer to classify the outputs of the reservoir memory system in an automated way and to check if the output trajectory agrees with that of the desired attractor. In the main text, the classifier RC is used in two tasks. The first is in the application of random perturbation and feedback control strategy to enhance the switching success rate with the index-based reservoir memory. The classifier reservoir computer, as described in Figs. 3(D), is deployed inside the feedback loop to determine if the desired attractor has been reached and if further perturbation is required. The second task is with index-free reservoir computers, where a classifier is used to distinguish the retrieved trajectories and to produce the retrieval success rate. A classifier is necessary here as no explicit ground truth trajectory can be found for the chaotic target states, so we cannot use measurements such as RMSE or prediction horizon.

A reservoir-computing-based classifier has three layers: an input layer, a recurrent hidden layer, and an output layer, but without any index channel. The input signal is the three-dimensional time series from the training data or the output of the reservoir memory, where the latter is an m-dimensional one-hot vector $v_c(t)$ at each time step. Each dimension of the output of the classifier is associated with one memorized attractor. For a well-trained classifier, the ith entry of $v_c(t)$ being approximately one implies that the time series is classified as in the ith memorized attractor at this time t. On the contrary, if the ith entry of $v_c(t)$ is approximately zero, then the time series will be classified as not belonging to the ith memorized attractor at this time. More specifically, the hyperparameters of the classifier RC are shown in Tab. 1. The training is performed on each of the six target states ten individual times, each time with a training length of 500 steps. The training procedure is the same as training an index-free memory RC, except that the output target is changed to one-hot coding vectors.

After training, we apply the trained classifier RC to the output time series of the memory RCs that we collect from the two tasks discussed above. These output time series of the memory RCs become the inputs of the classifier RC. Some exemplary inputs and outputs of the classifier RC are shown in Fig. S5. If the time series is an accurate reconstruction of the target state, we should observe a dark stripe in the output of classifier RC at the correct row. A failed retrieval is shown as the second example in the fourth row where, after recalling, the correct Rössler chaotic attractor appeared for a short period of transient time before the reservoir memory switched to a different state. It can be seen that the stripe in the output at index $p = 4$ breaks, and a new but wrong stripe at index $p = 6$ is formed. Our criterion in distinguishing an accurate and stable reconstruction of

any of the target states versus a failed one (Either an untrained state is activated, or the retrieval is too unstable and does not stay in one state for the majority of the classifying window.) is as follows. We take the row of the classifier RC output that has the maximum mean value, and test if this mean value is within the interval $[0.75, 1.25]$ around the ideal value 1. We then also look at the mean values of other rows, to see if none of these mean values are larger than 0.25. If the answers to both of the tests are true, then the classifier has classified this recall as an accurate and stable one of the target states that corresponds to the row with the maximum mean value. The testing window is always set to 300 steps, with a 50-step washing-out period.

A confusion matrix is demonstrated in Fig. S6, with 1,200 trials collected from the outputs from 8 different index-free memory reservoir computers. The accuracy is very high, where only in 6 trials there is a disagreement between the classifier RC and the human labeler. All these 6 trials are caused by the boundary between successful recalls (of any target state) and failed recalls (none of the target states is accurately and stably reconstructed during the classifying window). This boundary is indeed hard to define. No confusion among the target states is found in all the 1,200 trials.

Supplementary Note 6. EFFECTS OF INDEX VALUES ON THE FUNCTIONAL REGIONS OF ARTIFICIAL NEURONS IN THE RESERVOIR NETWORK

It is shown in the main text that the index values modify the bias terms in the artificial neurons in the reservoir network, thus affecting the functional regions in the activation function. Here, we explicitly demonstrate this effect.

The attractors to be memorized are dynamical trajectories with periodic or chaotic oscillations. When an oscillatory signal is fed into the recurrent neural network in the hidden layer, most neurons will be excited to oscillate as well. For the index-based reservoir memory, the oscillatory patterns of the neurons can be tuned by the index values, as shown in Fig. S7. Corresponding to the case of memorizing six attractors with a one-dimensional index, we calculate the maximum/median/minimum values of the oscillatory pattern of each neuron for two different attractors, displayed in the left and right columns of Fig. S7, respectively. It can be seen that, for neurons with different p_i values, the maximum/median/minimum values of their oscillations are different, indicating that the oscillatory patterns of the neurons can indeed be tuned by the index value (through W_{index}). Remarkably, Figs. S7(B) and S7(E) show that $\tanh(W_{\text{index}} p_i)$ fits the median values of the oscillating neurons quite well. Overall, changing the index value can alter the oscillatory patterns in the reservoir network, making storage of independent attractors possible without the need for multistablity in the high-dimensional phase space of the dynamical network in the hidden layer.

Note that for the index-free reservoir memory system, because of the absence of index values, multistablity is necessary to realize any memory capacity. The occurrence of multistablity typically requires larger networks for the same task than index-based reservoir memory.

Supplementary Note 7. BASIN STRUCTURES IN INDEX-BASED RESERVOIR MEMORY AND SWITCHING SUCCESS RATES

Figure 2(F) in the main text shows that the variance among the average rates of each column $\text{Var}(\sum_i \eta_{i,j}/K)$ is much larger than those among the rows $\text{Var}(\sum_j \eta_{i,j}/K)$, where $\eta_{i,k}$ is

the success rate of switching from attractor s_i to attractor s_j. This indicates that the success rate of switching among memorized attractors depends more on the destination attractor than on the starting attractor.

To understand this dependence, we use the example Fig. 2(G) in the main text with 16 stored chaotic attractors. Choosing attractor $\#11$ as an example, we locate the regions in the 3D attractor generated by the reservoir memory where switching starts and distinguish those with successful (darker blue) and failed (orange) switchings, as shown in Fig. S8(A), where the ten panels correspond to ten different destination attractors (attractors Nos. 1, 2, 3, 4, 5, 6, 10, 13, 15, and 16, respectively). The success rate of switching can be estimated as the ratio between the numbers of blue and orange points. It can be seen that, on the starting attractor ($\#11$), the regions leading to successful switching to different destination attractors are different with distinct relative sizes and structures. For instance, the third panel in the first row gives a riddled structure, while the second panel in the second row has a smooth boundary between the blue and orange regions. These patterns are related to the basin structures of the destination attractors.

It is useful to further investigate the basin structures of the memorized attractors in the high-dimensional phase space of the reservoir network. The results are summarized in Fig. S8(B), where the thirty panels are organized as ten vertical columns, each corresponding to a panel (a distinct destination attractor) in Fig. S8(A), in the same order. In each column, the three panels show three 2D slices of the basin structure in the N-dimensional phase space, respectively, where the dark blue region belongs to the basin of the destination attractor. The basin structures are computed, as follows. In each panel, the center point, with position $\vec{c_0}$ in the phase space, is chosen to be within the basin of attraction and is thus blue. Different perturbations are then applied: $\vec{h} = \epsilon_1\vec{h}_1 + \epsilon_2\vec{h}_2$, where $\epsilon_1, \epsilon_2 \in [-10, 10]$ to the center points. Whether the reservoir memory stays in the destination attractor after the perturbation determines if the point at (ϵ_1, ϵ_2) (corresponding to the point $\vec{c_0} + \vec{h}$ in the high-dimensional phase space) is within the basin or not. It can be seen that, in terms of the relative sizes of the blue and orange regions as well as their structures, there is strong correlation between each panel in Fig. S8(A) and the corresponding three-panel column in Fig. S8(B). The structural correspondence is particularly remarkable. For example, the riddled structure in the third panel in the first row of Fig. S8(A) also appears in the third column of three panels in Fig. S8(B), and the "clean" boundary in the first panel in the second row of Fig. S8(A) can also be observed in the corresponding panels in Fig. S8(B).

The structural correlations suggest that the successful switching regions in Fig. S8(A) are the projections of the basin of attraction of the destination attractors. Furthermore, as the memorized attractors are confined in similar three-dimensional phase-space regions of their original dynamical systems, the corresponding regions that they reside in the high-dimensional phase space of the reservoir network should also be similar. However, the results in Fig. S8(B) indicate that the basins of attractions of different attractors can be quite different in terms of their sizes and structures. It is these differences that determine the success rate of memory switching.

Supplementary Note 8. PERFORMANCE OF OUR FEEDBACK CONTROL STRATEGY FOR HIGHER SWITCH SUCCESS RATES UNDER DIFFERENT PARAMETERS

In the main text, we present the performance of our feedback control strategy in Fig. 3 (E) with a moderate set of control parameters. The entire recall procedure under this control strategy is essentially a loop of trial and error. Within each iteration, there is a perturbation phase and a

classification phase. We apply random noises with a certain amplitude and temporal length to the memory RC in the perturbation phase, and then use the classifier RC to provide feedback information on whether the target state is reached. Figure S9 shows the performance of this strategy for indexed memory RC under different parameters. We vary the temporal length of random perturbation and the noise levels in each iteration of the trial and error loop. All the random perturbations are implemented by Gaussian white noise with standard deviation σ_p. The tests are run on the same ensemble of 25 memory RCs as in the main text, each trained with 6 attractors. Figure S9 (C2) is also the panel shown in the main text. It can be considered as the optimal one in all the 12 different combinations of strategy parameters. A moderate time length (around 10 steps) enables it to utilize almost the full potential of this method without taking too much time or computational resources. A moderate noise level (around $\sigma_p = 1$) also leads to a high overall correction success rate.

Supplementary Note 9. DYNAMICAL PROCESS OF RETRIEVAL IN INDEX-FREE RESERVOIR-COMPUTING MEMORY

We provide a further demonstration of the dynamic process of retrieval in index-free reservoir memory systems. As discussed in the main text, the goal of the process is to reach a target trajectory $g[u(t)]$ from a random initial state of the memory dynamical system, as illustrated in Fig. S12. In particular, in Fig. S12 (A), the four panels show four cases of random initial state (dashed red curves) approaching the target trajectory $g[u(t)]$ (solid purple curves) during the warming phase. The dashed red curves can approach the purple curves after one or two dozen steps, in agreement with the threshold values in Fig. 5(A) in the main text. Figure S12(B) shows how the warming data for the dashed red and solid purple curves in Fig. S12(A) are prepared. The echo state property of reservoir computing stipulates that the dynamical output trajectory can approach the target attractor after sufficient warming. The purple curves in Fig. S12(A) are simulated by the trajectories of the reservoir memory system after 400 steps of warming.

Supplementary Note 10. DEPENDENCY OF MEMORY RETRIEVAL IN INDEX-FREE RESERVOIR COMPUTERS WITH PARTIAL INFORMATION ON THE SPECIFIC MISSING DIMENSIONS

In the main text, we demonstrate how our index-free memory RC can still function and recall target states with partial cues missing some dimensions. However, we only show one missing-dimension scenario for each dimensionality in the main text, while there are more possible combinations of missing dimensions in the three-dimensional target states we test. It could be interesting to see how the retrieval performance depends on the specific combinations of dimensions that are missing. Here, we demonstrate comprehensive results on all the possibilities of missing dimensions with all the six attractors we test. The results are shown in Fig. S13 and Fig. S14. Comparing the six attractors, we observe that the thresholds of the cue length where the success rate begins to rise significantly larger than a random recalling are postponed similarly to the results shown in Fig. 6 in the main text. This again verifies that the threshold of cue length is a feature of the RC structure and properties rather than a feature of the specific target attractor, with the same way of rewiring feedback loops yielding the same thresholds. A decrease in the saturated success rate is

also observed in multiple cases, and is more frequent for the 1D cues than the 2D cues, as more information is lost. Among the four chaotic attractors, the Lorenz attractor (in panel (C)) can always reach a 100% success rate. Both the chaotic Rossler system and the chaotic food chain system suffer no decrease in the saturated success rate except in the sole case where only the third dimension is lost. The most intriguing case is with the periodic Sprott system, where missing the second dimension alone would result in a worse saturated success rate than missing two dimensions. This suggests that having more dimensions hidden during the retrieval does not always make the success rate worse; the relationship between missing dimensions and the change in success rate is much more complicated and system-dependent. Further research is necessary to unveil a possible generic understanding of these interesting phenomena.

Supplementary Note 11. EFFECTS OF NOISE AND RANDOM ITINERANCY

We apply independent Gaussian white noise of standard deviation of σ_n to each neuron in the reservoir network, for both index-based and index-free memory systems. For small noise, the output of the reservoir memory contains small random fluctuations. For large noise, the dynamics of the reservoir network are stochastic without any distinguishable dynamical patterns. For intermediate noise, an intermittent behavior between the memorized attractor and some random states arises for the index-based reservoir memory. For the index-free memory, because all the memorized attractors coexist in the phase space of the reservoir network, there is a random itinerary among the attractors. These results are exemplified in Figs. S15 and S16 for the index-based and index-free reservoir memory systems, respectively.

SUPPLEMENTARY FIGURES

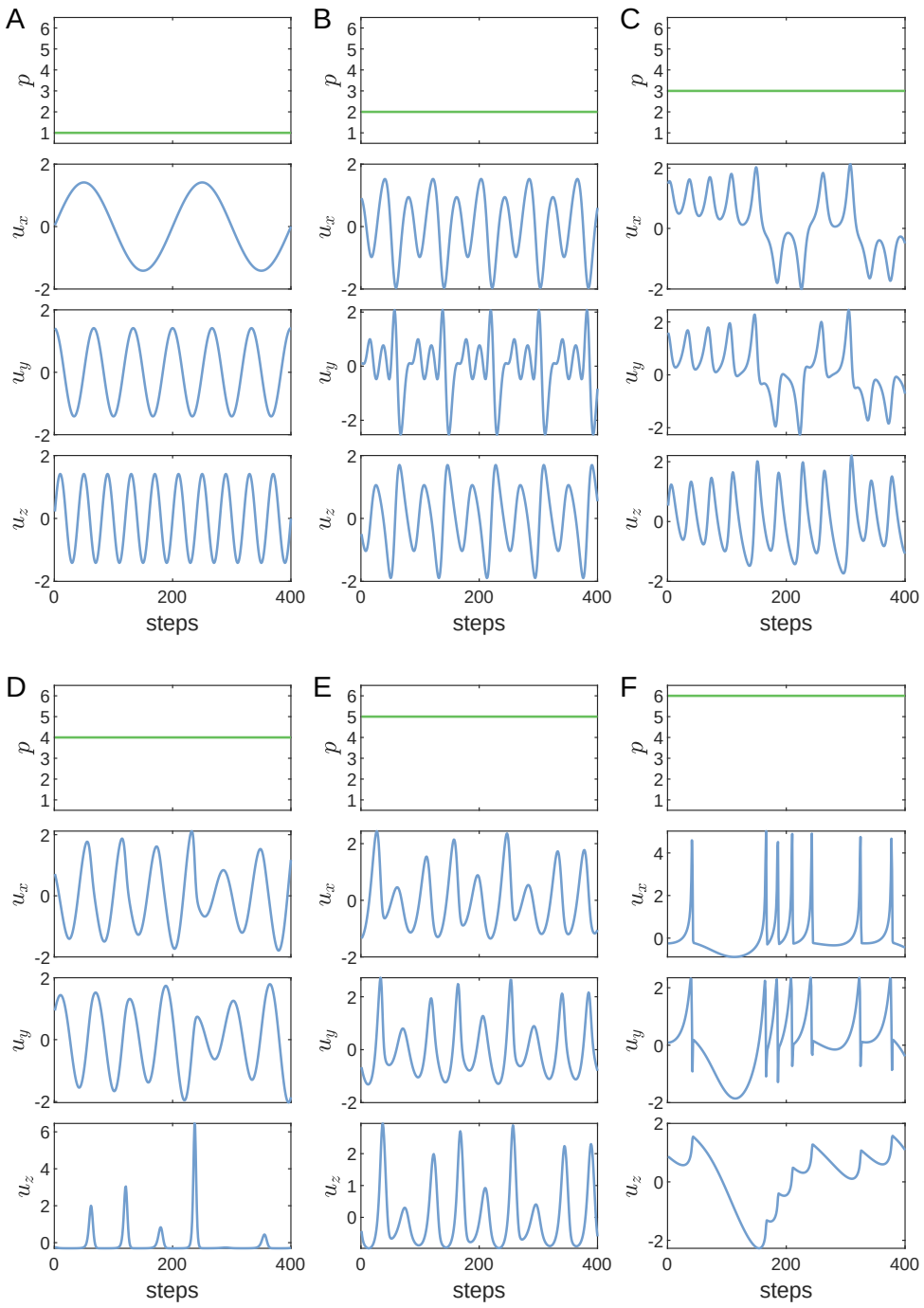

FIG. S1. Training data for the six attractors in Fig. 1(B) in the main text. The attractors are (A) a periodic Lissajous attractor, (B) a periodic attractor of the Sprott system, (C) the classic chaotic Lorenz attractor, (D) the classic chaotic Rössler attractor, (E) a chaotic attractor from food chain system, and (F) a chaotic attractor from the HR neuron system. The phase space for all six attractors is three-dimensional. During training of each attractor, a constant index value p is injected into the recurrent neural network through the index channel for index-based reservoir-computing memory, where no index channel is needed for index-free memory.

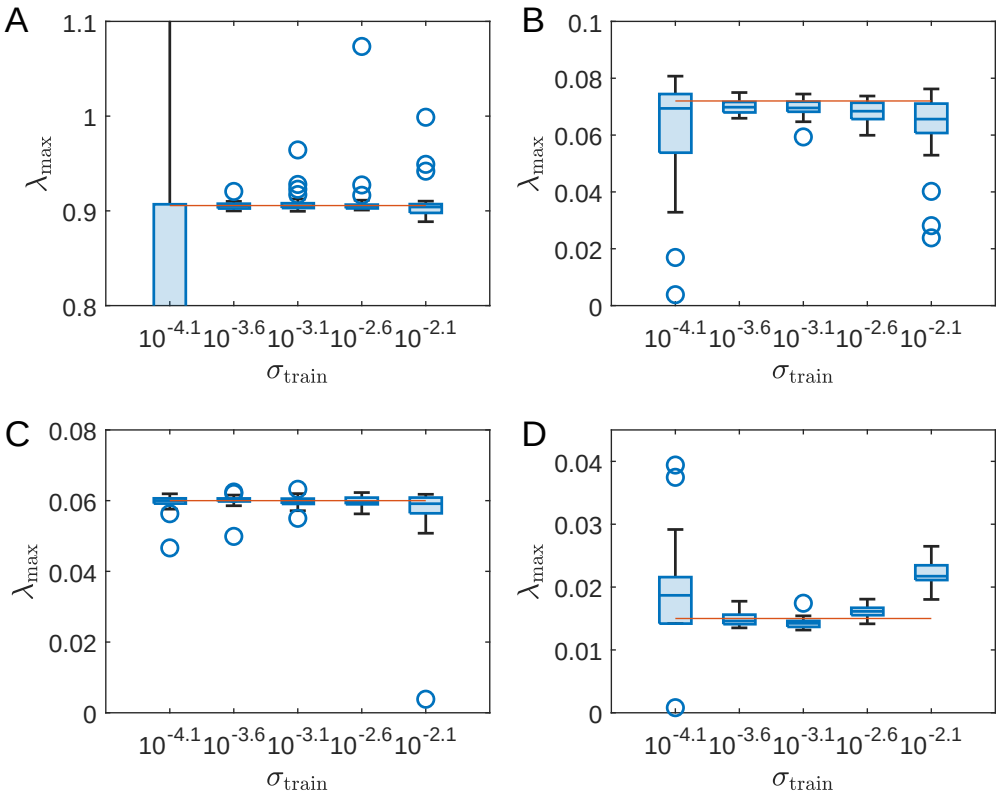

FIG. S2. Maximum Lyapunov exponent $\lambda_{\max}$ of the chaotic attractors reconstructed from the recalls of the memory RCs under different levels of training noise σ_{train}. The chaotic target states in each panel are (A) the Lorenz system, (B) the Rossler system, (C) the chaotic food chain system, and (D) the HR system. The red horizontal line represents the ground truth value of $\lambda_{\max}$ for each system. The upper and lower edges of the boxes represent the upper and lower quartiles of the resulting $\lambda_{\max}$ from 30 different memory RCs. The horizontal lines inside the boxes represent the median values of these resulting $\lambda_{\max}$. The outliers are shown by the circles, which are values that are more than 1.5 times of the interquartile range away from the top or bottom of the box. The upper whisker connects the upper quartile to the nonoutlier maximum (the maximum data value that is not an outlier), and the lower whisker connects the lower quartile to the nonoutlier minimum (the minimum data value that is not an outlier). The optimal noise level we have from the hyperparameter optimization is $\sigma_{\text{train}} = 10^{-3.1}$. It appears that when the noise level is around the optimal level, the majority of the $\lambda_{\max}$ values from the memory RCs agree with the ground truth values well. The index-based memory RCs tested here use the hyperparameter set Indexed #1 with $N = 1,200$. The training length for each target state is 6,000 steps. The maximum Lyapunov exponents $\lambda_{\max}$ are calculated from running the recalled state for 200,000 steps of iteration.

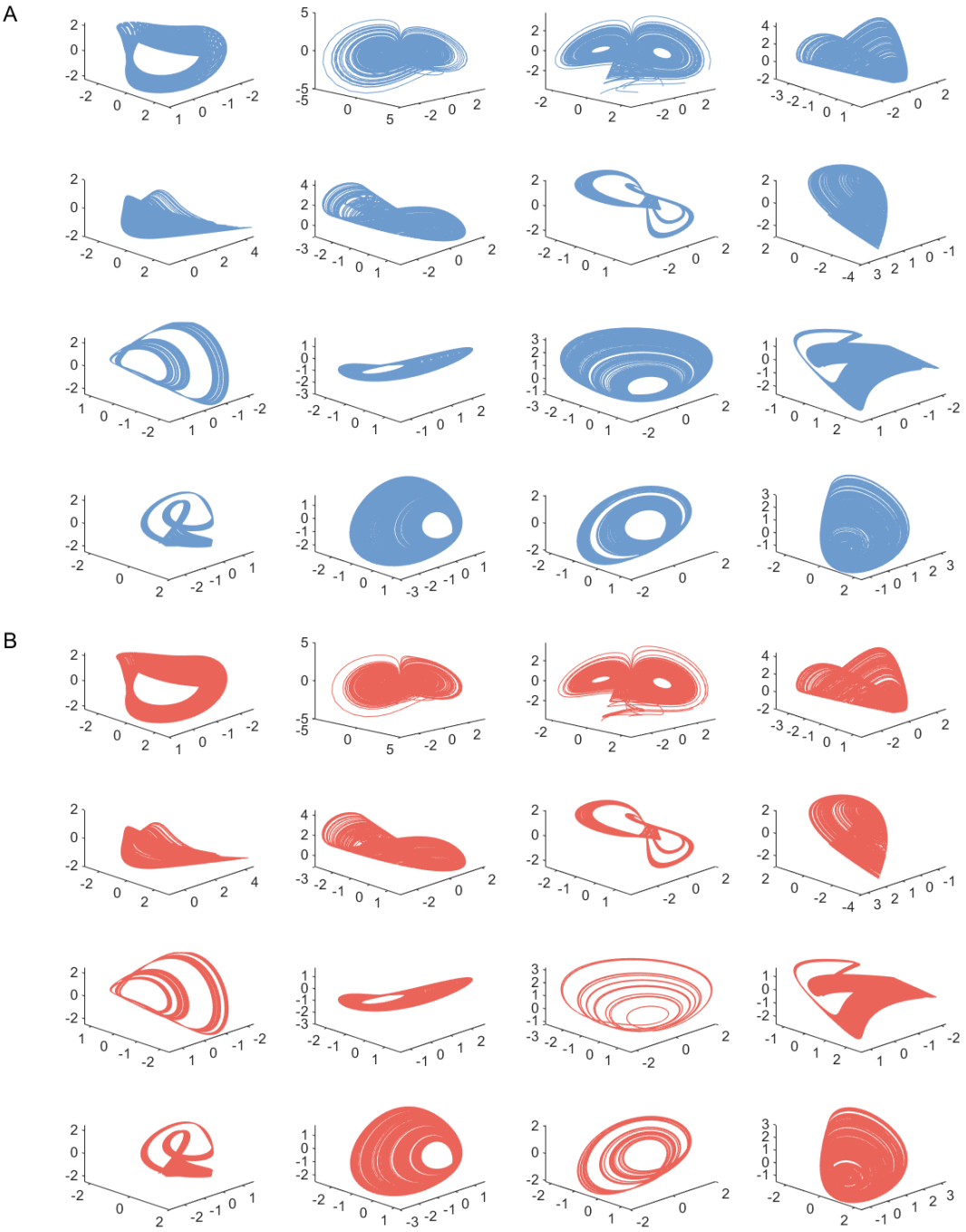

FIG. S3. Performance of index-based reservoir memory for memorizing 16 chaotic attractors. (A, B) Target (ground truth) and retrieved attractors, respectively. The 16 attractors are those in Fig. 1(C) in the main text. All the chaotic attractors can be successfully stored and faithfully recalled. Upon retrieval of any attractor, the reservoir system can generate an arbitrarily long trajectory on the attractor.

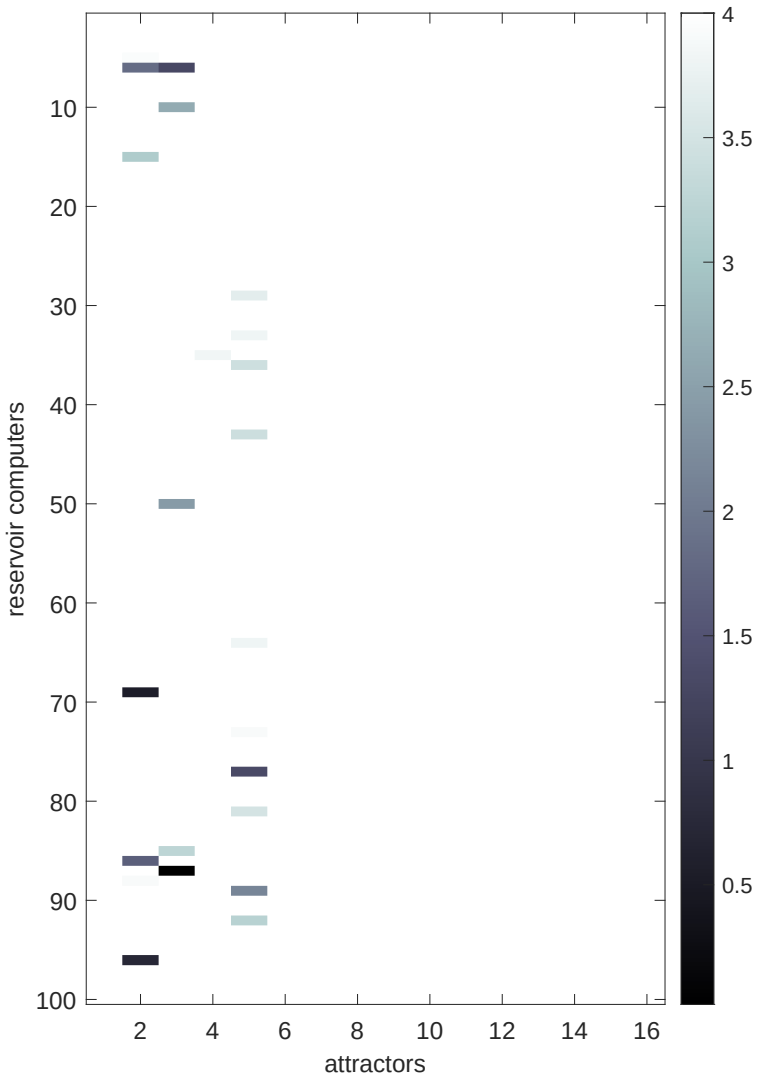

FIG. S4. Performance of index-based reservoir memory for memorizing 16 chaotic attractors, with a two-dimensional encoding where the order of the coding is randomized. The color represents the prediction horizon (by the unit of average period, which is the average temporal distance between two local maximums in the target state). We train and test 100 different reservoir computers. The 16 chaotic attractors are fixed (as the ones shown in Fig. S3), but which index each attractor is assigned is randomized. The prediction horizon is defined as the maximum temporal length during a recall testing that, in none of the dimensions of the target system, the deviation between the RC-generated trajectory and the ground truth target state is larger than 10% of the maximum value minus the minimum value in the target state. Our result suggests that our approach can successfully memorize and recall almost all the attractors regardless of how the index values are assigned.

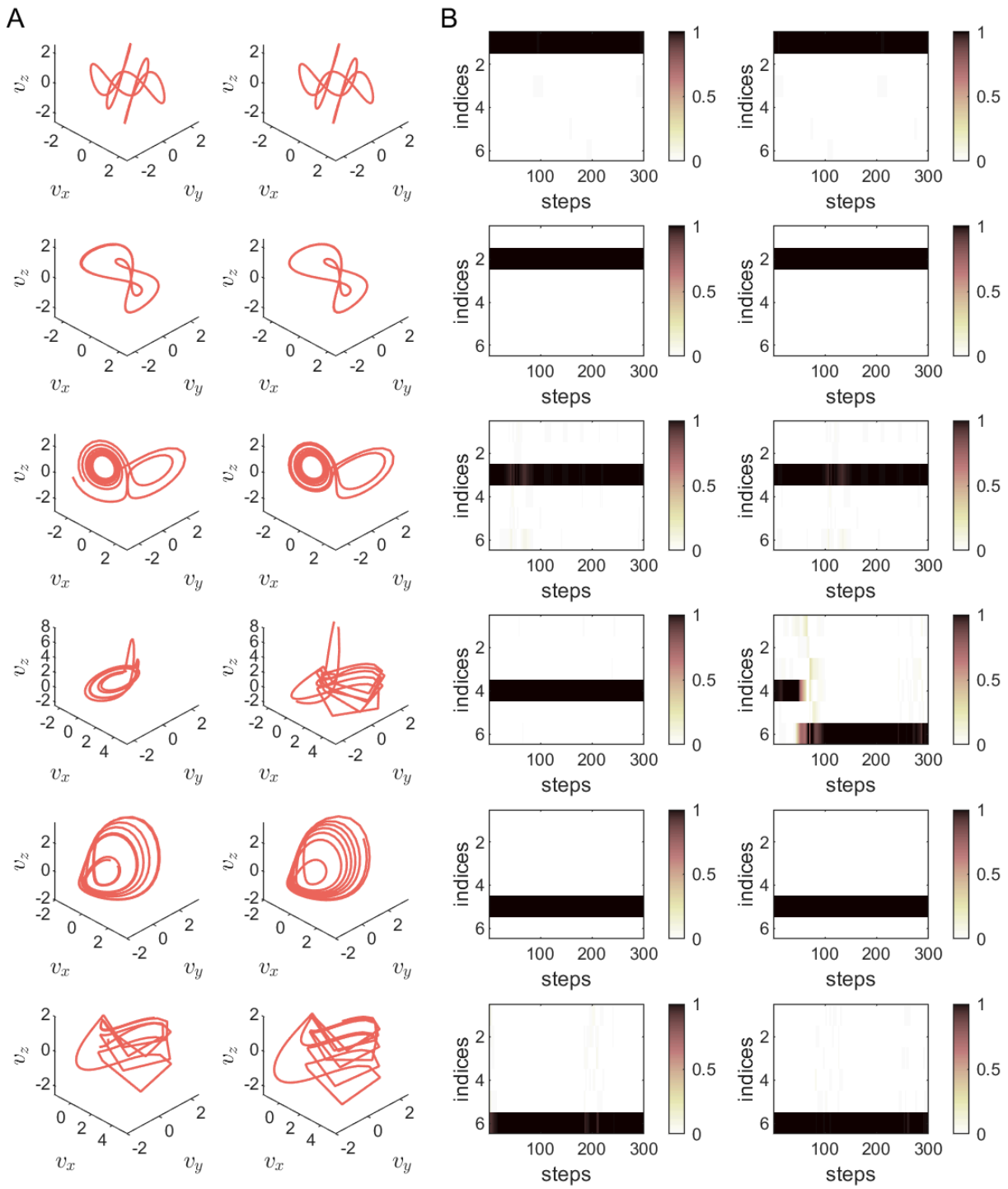

FIG. S5. Working example of the reservoir-computing-based classifier. (A, B) Exemplary input time series and output classifying results, respectively. The time series are generated by the reservoir memory system during the retrieval process. A dark stripe in the correct row of the classifier output indicates that the time series tested is from the correct memorized attractor. An example of failed retrieval is shown in the fourth row where, after recalling the correct Rössler attractor for a short period of transient time, the system switches to the sixth attractor, as shown in (A). In the corresponding panel in (B), the dark stripe in the output at index $p = 4$ breaks and a new stripe at index $p = 6$ is formed.

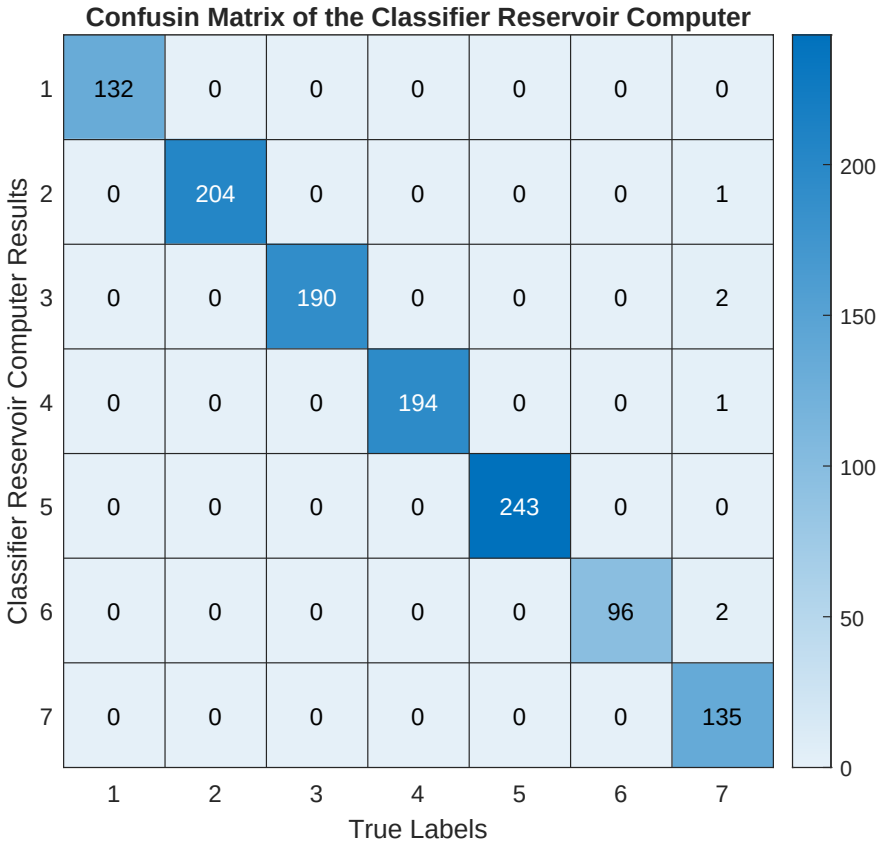

FIG. S6. Confusion matrix of the reservoir-computing-based classifier. Labels 1 to 6 represent the six target states, while label 7 represents an untrained state in a failed recall. The RC classifiers show high accuracy in classifying different target states as well as distinguishing untrained states. Among all the 1200 trials among 8 different memory RCs, there are only 6 trials where the classifier RC results are different from the human labeler. All these 6 trials are caused by the rather ambiguous and hard-to-define boundary between a successfully recalled target state and a failed one.

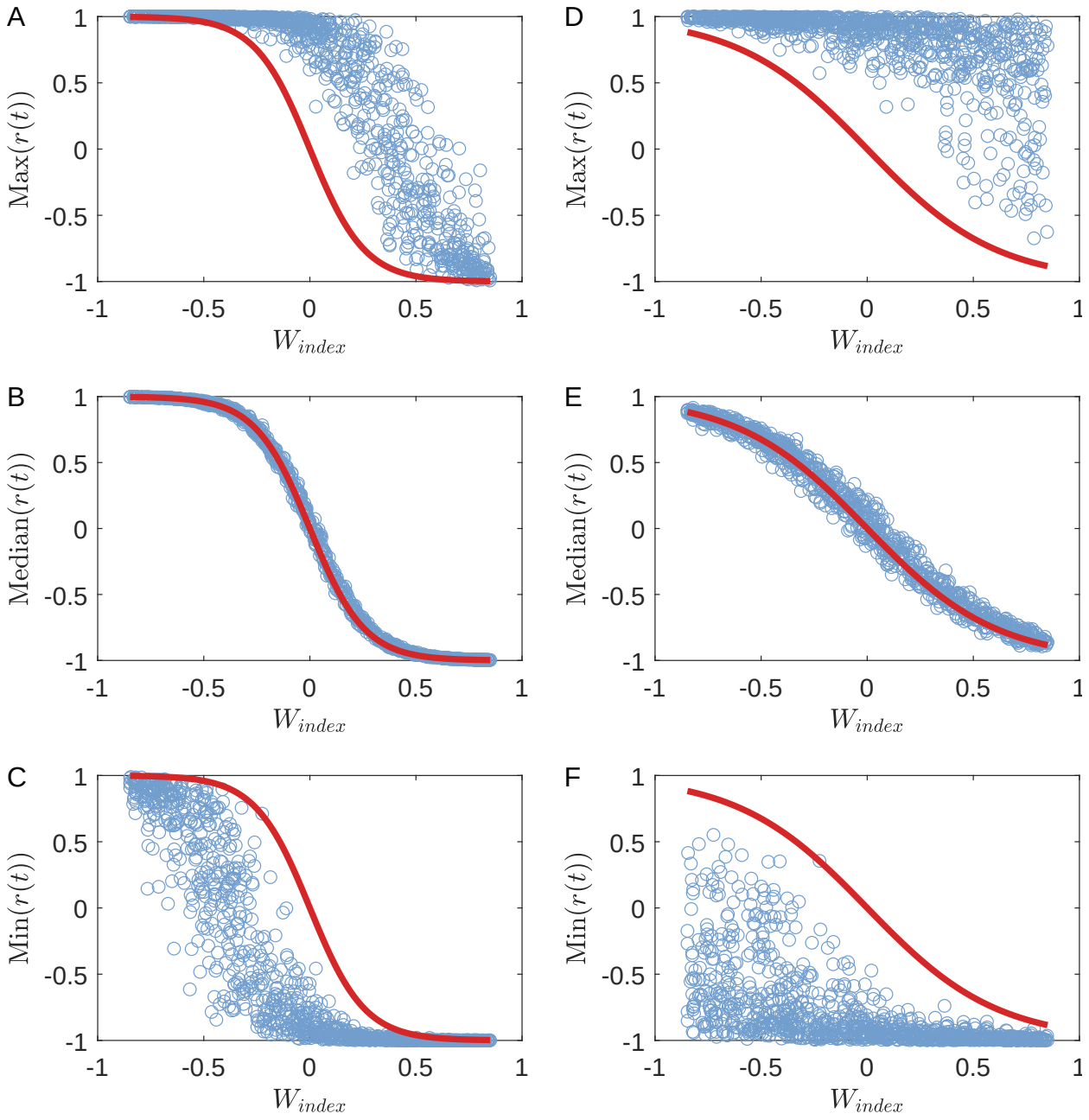

FIG. S7. Effects of index values on the functional regions of artificial neurons in the reservoir network. (A, D) The maximum (B, E) median, and (C, F) minimum values of each neuron for two different target attractors in an index-based reservoir memory trained to store the six attractors displayed in Fig. 1(B) in the main text, where the left and right columns are for attractors 1 and 3, respectively. Each blue circle represents the state of a neuron in the reservoir network, the horizontal coordinate of which is the value of the entry in W_{index} connected to that neuron. The red curves represent the function $\tanh(W_{\text{index}}p_i)$, which fits well the median values of the oscillating neurons [(B) and (E)]. The results demonstrate how the oscillatory patterns of the neurons in the index-based reservoir memory are tuned by the index value through W_{index}.

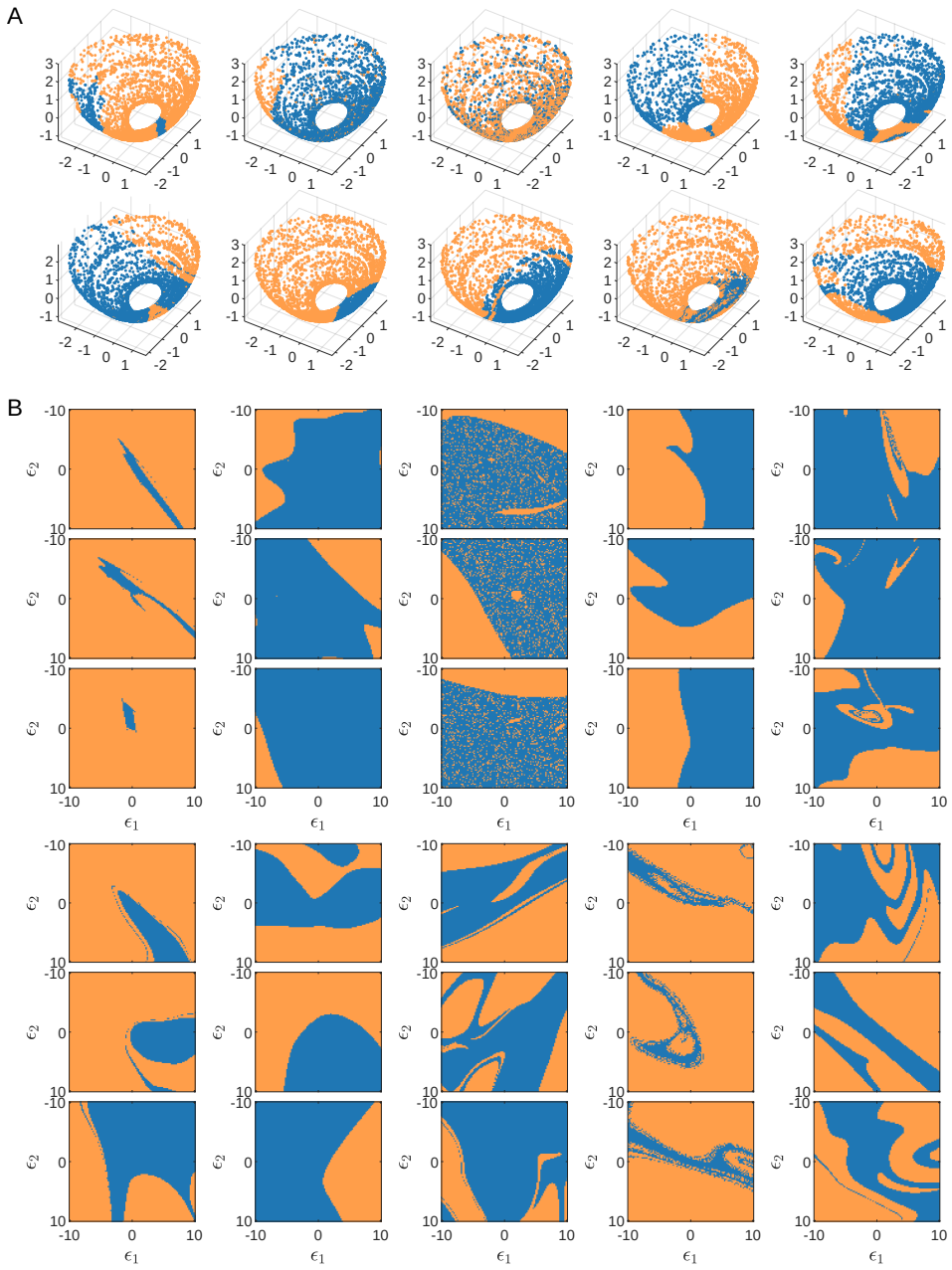

FIG. S8. Switching success/failure landscape and basin structures in index-based reservoir computer with multiple memory states. (A) Regions of successful and failed switching from the same starting attractor (No. 11 in Fig. 3(C) in the main text) to ten different destination attractors, where each dot is the point at which the switching begins. The darker blue and orange dots correspond to successful and failed switchings, respectively. (B) The corresponding basin structures of the ten different destination attractors in the high-dimensional phase space of the reservoir network from the same starting attractor in (A), where each panel in (A) corresponds to a column of three different 2D slices (panels) in (b), in the same order. In each panel, the darker blue regions denote the attracting basin of the corresponding destination attractor, while the orange regions do not belong to the basin of attraction and lead to failed switching as the reservoir output can be some irrelevant dynamical states (e.g., a fixed point).

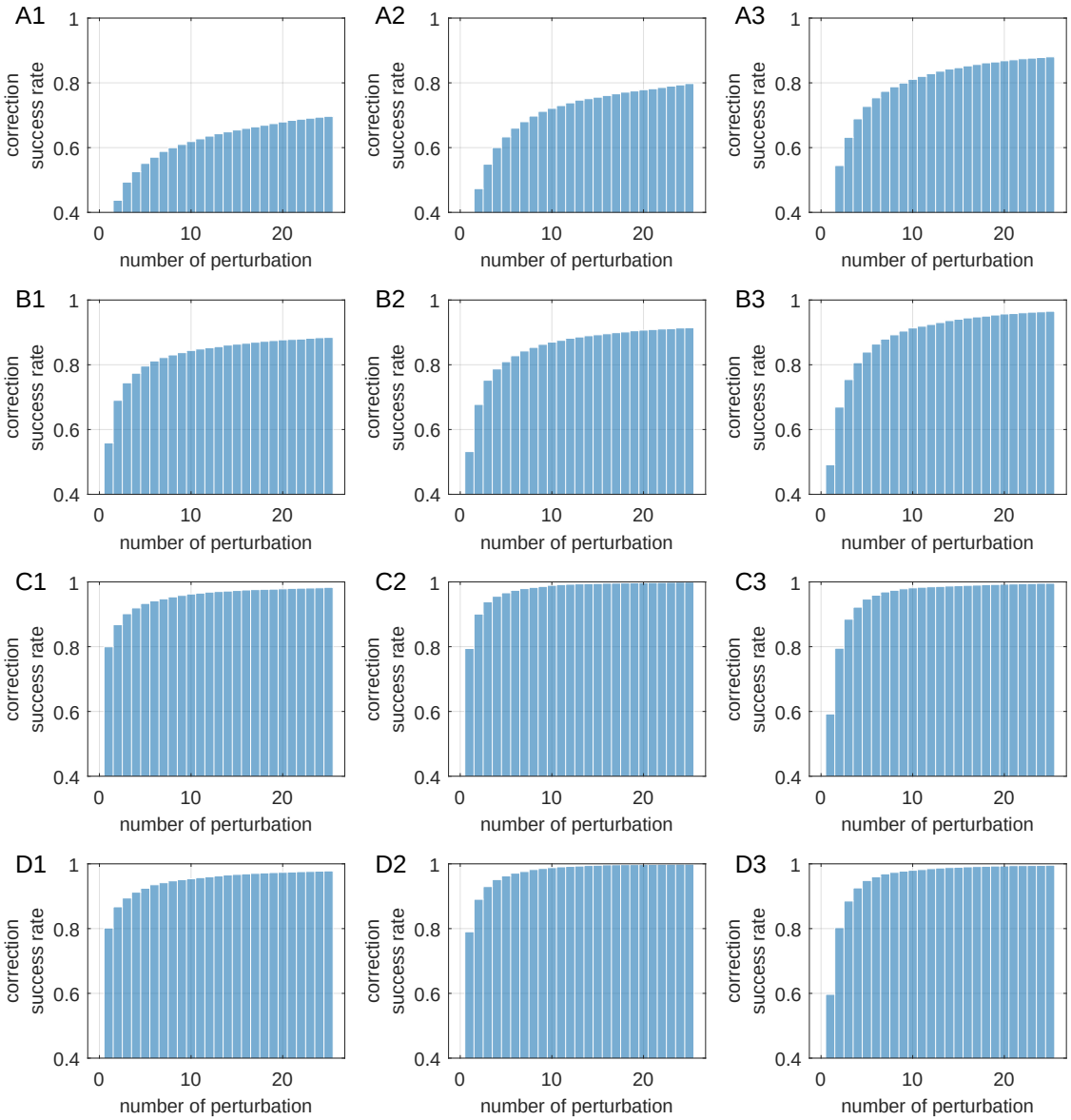

FIG. S9. Performance of our second control strategy (with a classifier RC and random perturbations) under different setting parameters. The length of a single run of random perturbation is (A1, A2, A3) 1 step, (B1, B2, B3) 3 steps, (C1, C2, C3) 10 steps, and (D1, D2, D3) 30 steps. The noise levels are (A1, B1, C1, D1) $\sigma_p = 0.3$, (A2, B2, C2, D2) $\sigma_p = 1$, and (A3, B3, C3, D3) $\sigma_p = 3$. We observe that there is little difference between the (C1, C2, C3) row and the (D1, D2, D3) row, while the (B1, B2, B3) and (A1, A2, A3) rows have significantly worse performance. This result suggests that one needs the perturbation length to be not way too short, but does not need it to be very long either as it will not enhance the performance much. The comparisons among the three columns also suggest the existence of an optimal moderate noise level.

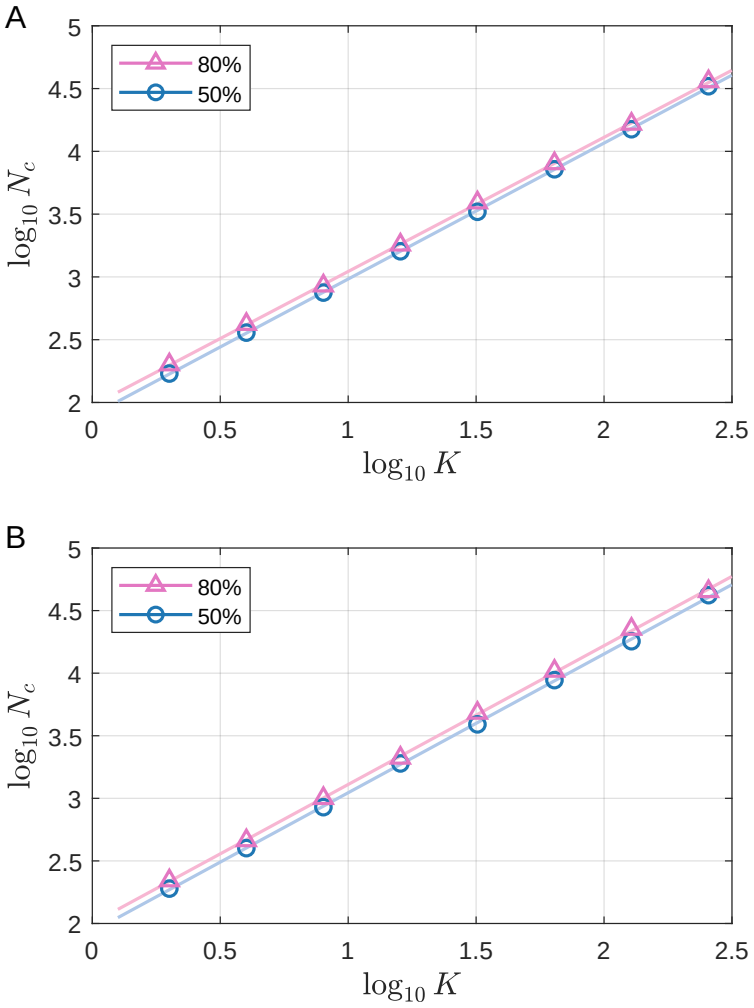

FIG. S10. Comparisons between the scaling laws plotted by the 50% success rate versus the 80% success rate. Shown are two pairs of examples with indexed memory RC with the on-hot coding on Dataset #1 with (A) the region-based performance measure and (B) the prediction-horizon-based performance measure. The scaling laws do not appear to differ with different success rates, except for a constant factor.

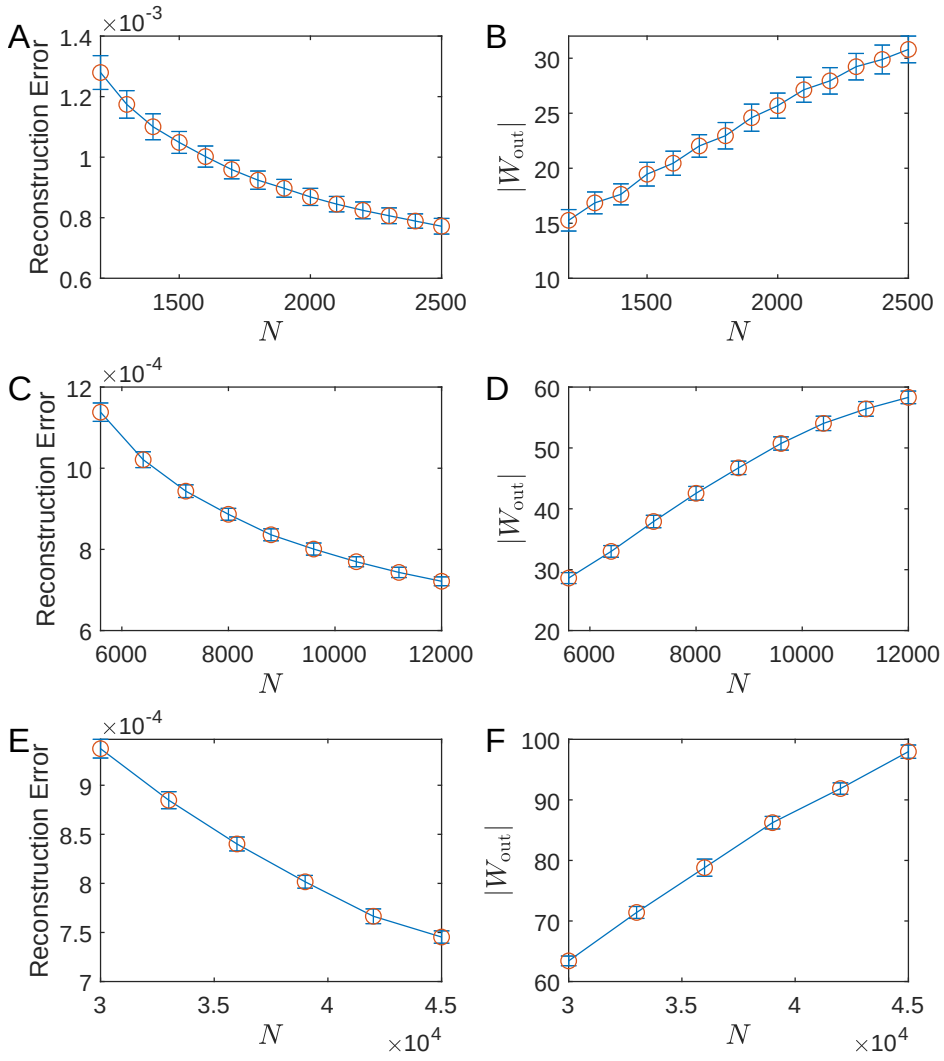

FIG. S11. Reconstruction error (A, C, E) and 2-norm of the W_{out} (B, D, F) of the index-based memory RC near the critical network size N_c. All the memory RCs in this figure are trained on Dataset #1 with a one-hot coding. Panels (A, B) are from memory RCs trained with $K = 16$ attractors, with $N_c = 1,600$ with the region-based measure and $N_c = 1,900$ with the prediction-horizon-based measure. The results are averaged over 200 random RCs, and the error bar represents the standard deviation within this ensemble. Panels (C, D) are from memory RCs trained with $K = 64$ attractors, with $N_c = 7,200$ with the region-based measure and $N_c = 8,800$ with the prediction-horizon-based measure. The results are averaged over 120 random RCs, and the error bar represents the standard deviation within this ensemble. Panels (E, F) are from memory RCs trained with $K = 256$ attractors, with $N_c = 33,000$ with the region-based measure and $N_c = 42,000$ with the prediction-horizon-based measure. Here, the reconstruction error is calculated by the RMSE on the training data of all target states after training. The results are averaged over 25 random RCs, and the error bar represents the standard deviation within this ensemble. As the number of memory states K increases for more than an order of magnitude, the reconstruction error around N_c is always around 8×10^{-4} to 1×10^{-3}. The 2-norm of the W_{out} around N_c is increasing, but not as fast as K.

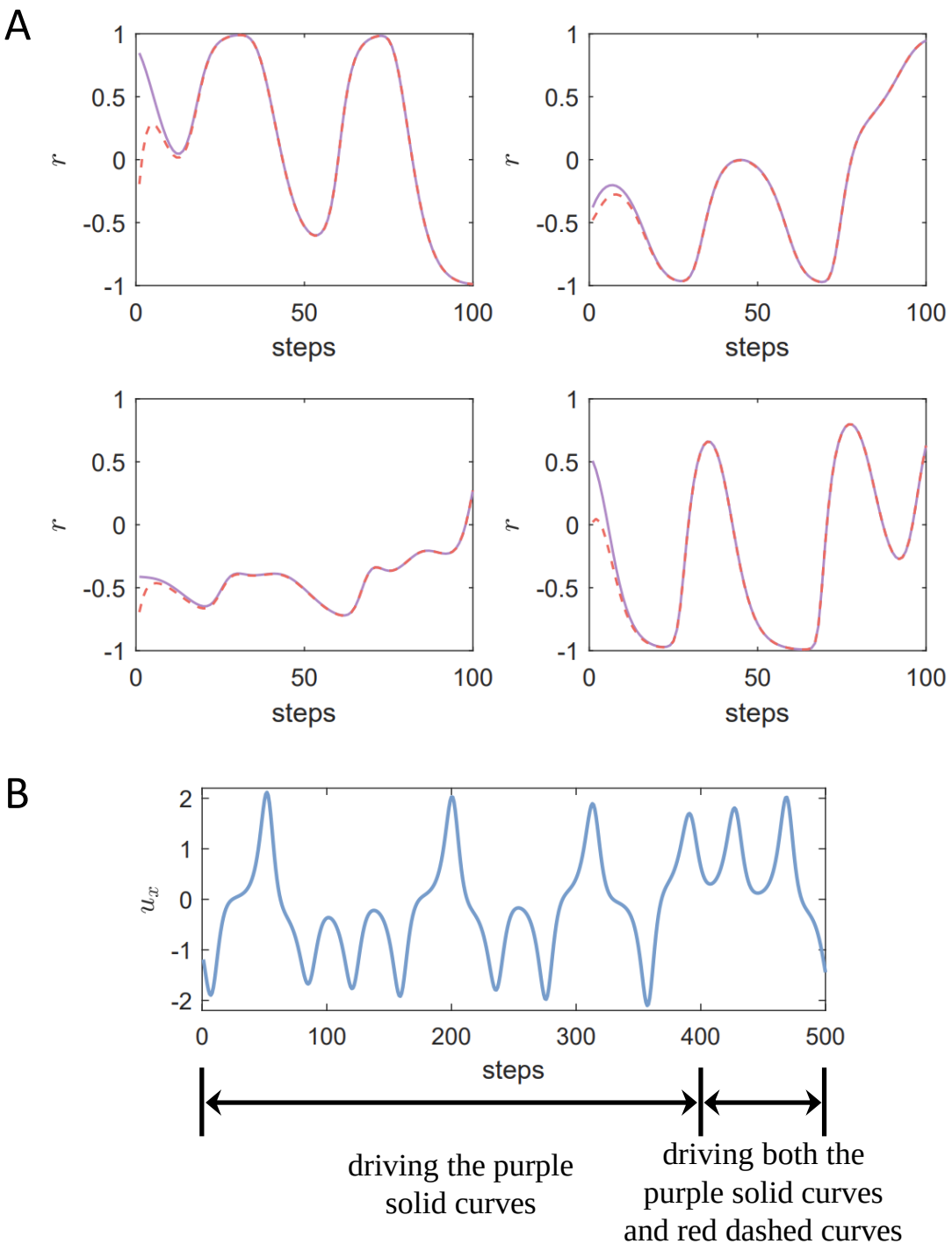

FIG. S12. Dynamical process of retrieval in index-free reservoir-computing memory. Presented is a demonstration of how the network state of index-free reservoir memory approaches the target trajectories. (A) Four random examples of the network dynamical state (dashed red curves) approaching the target trajectories (solid purple curves) after one or two dozen steps. (B) Warming data for the dashed red and solid purple curves in (A).

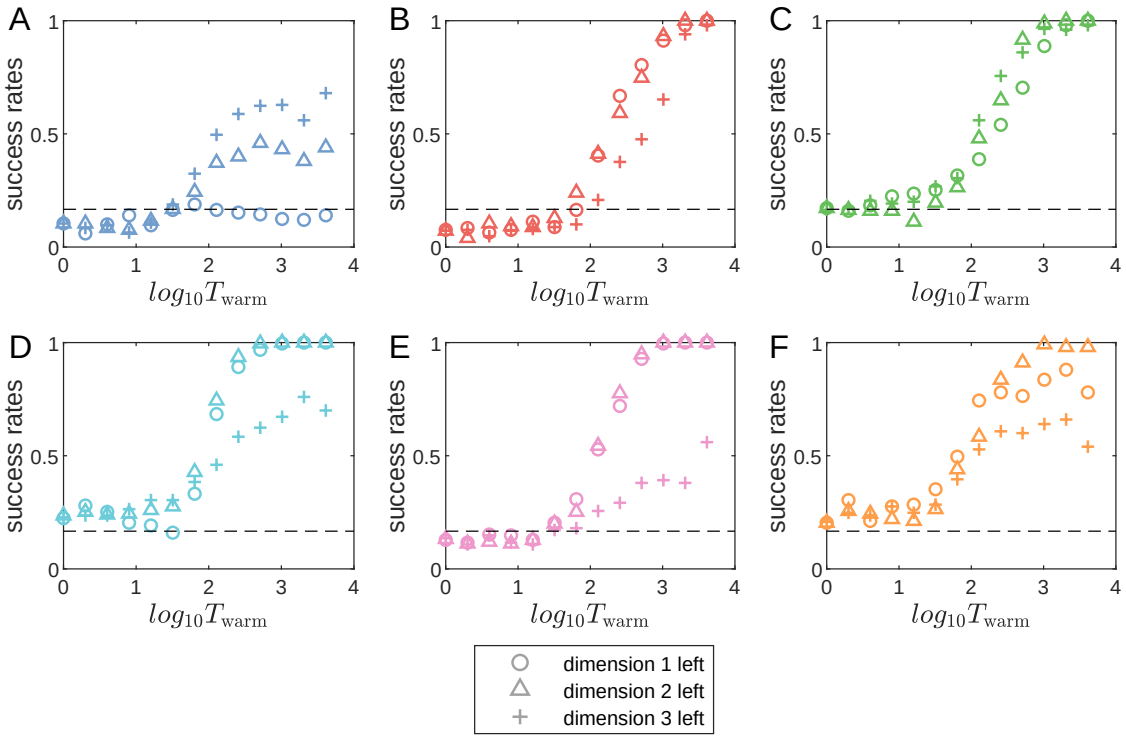

FIG. S13. Attractor retrieval with partial cues in index-free memory RCs for all possible scenarios with only one dimension left in the originally three-dimensional cues. (A-F) Success rate of retrieval versus the cue length for the six attractors in Fig. 1B in the main text (from left to right)). All values of the hyperparameters, training settings, and target memory states are the same as those in Fig. 6 in the main text.

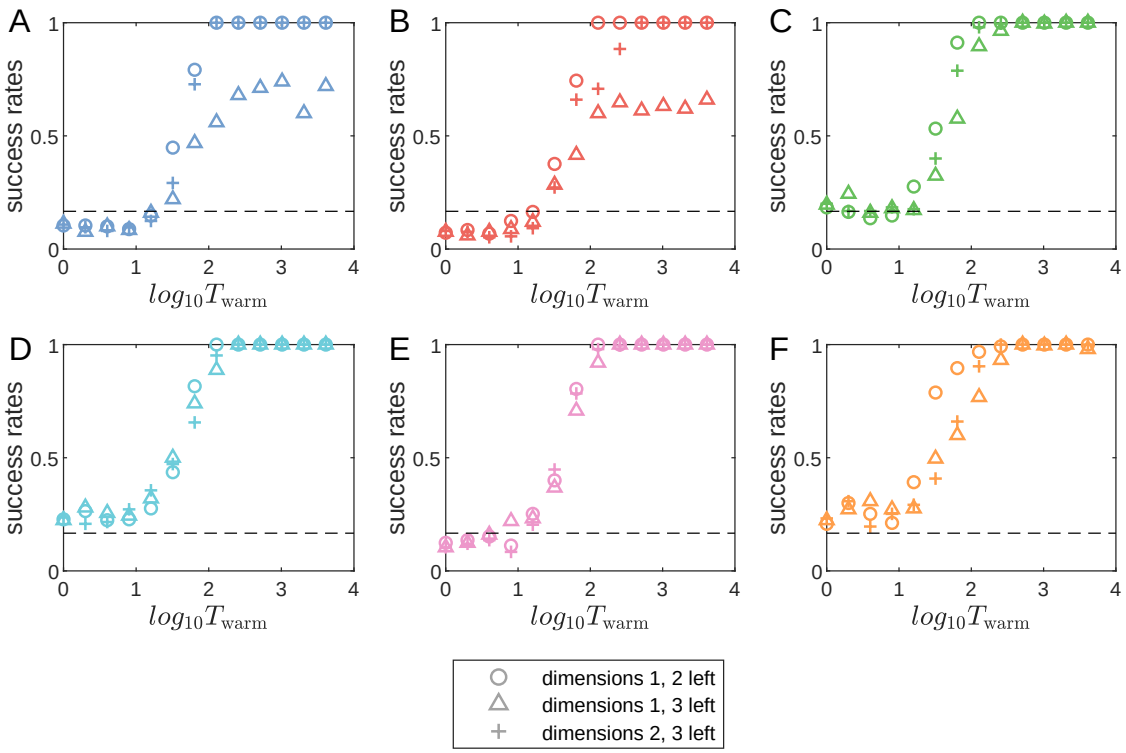

FIG. S14. Attractor retrieval with partial cues in index-free memory RCs for all possible scenarios with two dimensions left in the originally three-dimensional cues. (A-F) Success rate of retrieval versus the cue length for the six attractors in Fig. 1B in the main text (from left to right)). All values of the hyperparameters, training settings, and target memory states are the same as those in Fig. 6 in the main text.

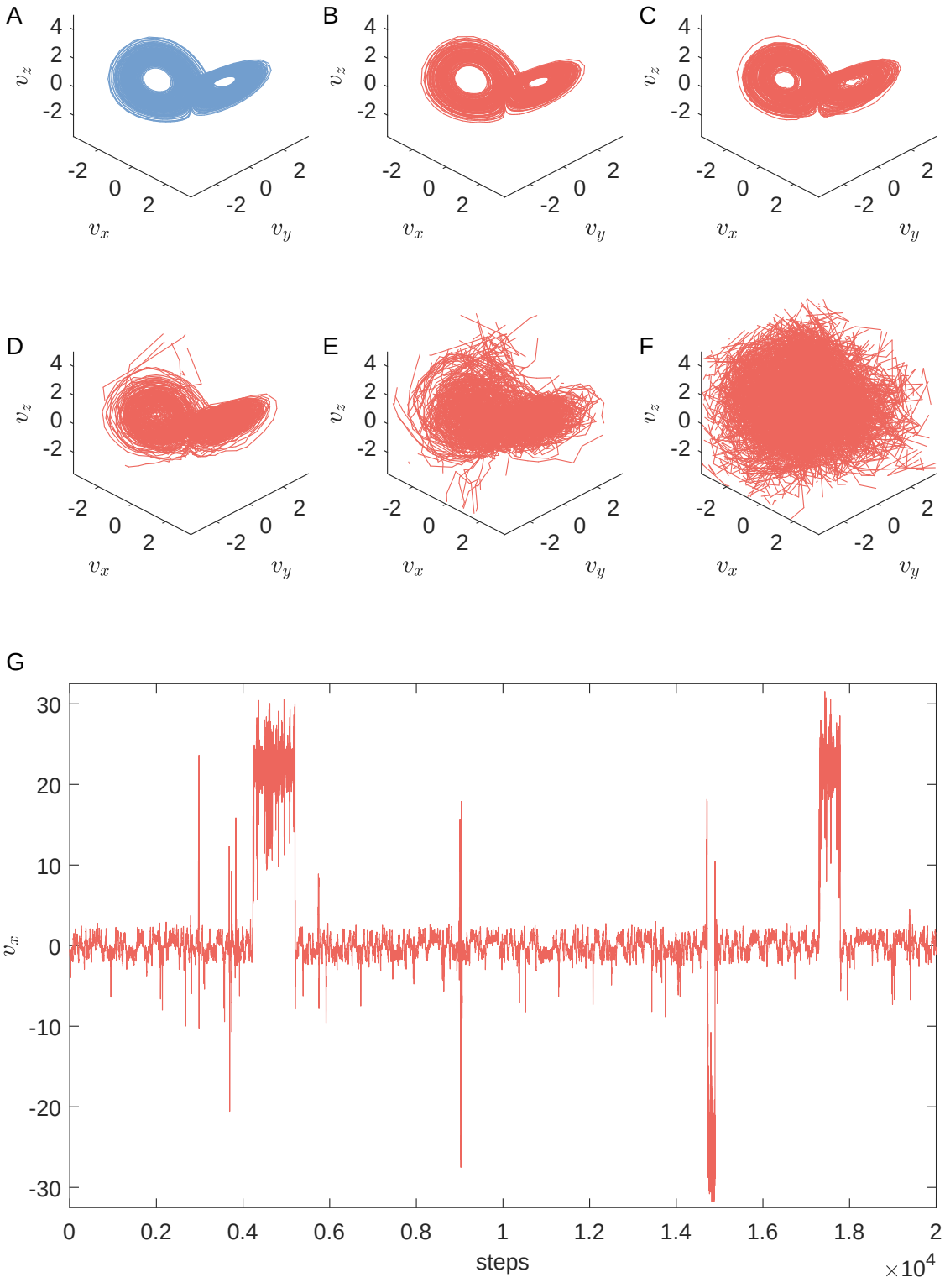

FIG. S15. Effects of noise in index-based reservoir memory system. (A) Original attractor (ground truth). (B-F) Recalled attractors under different levels of noise applied to each neuron in the reservoir network: (B) $\sigma_n = 10^{-4}$, (C) $\sigma_n = 10^{-3.5}$, (D) $\sigma_n = 10^{-3}$, (E) $\sigma_n = 10^{-2.5}$, and (F) $\sigma_n = 10^{-2}$. (G) Intermittency between the memorized attractor and some random untrained states for $\sigma_n = 10^{-2.5}$. When the output v_x is within the interval $(-3, 3)$, the output trajectory is close to the true memorized attractor.

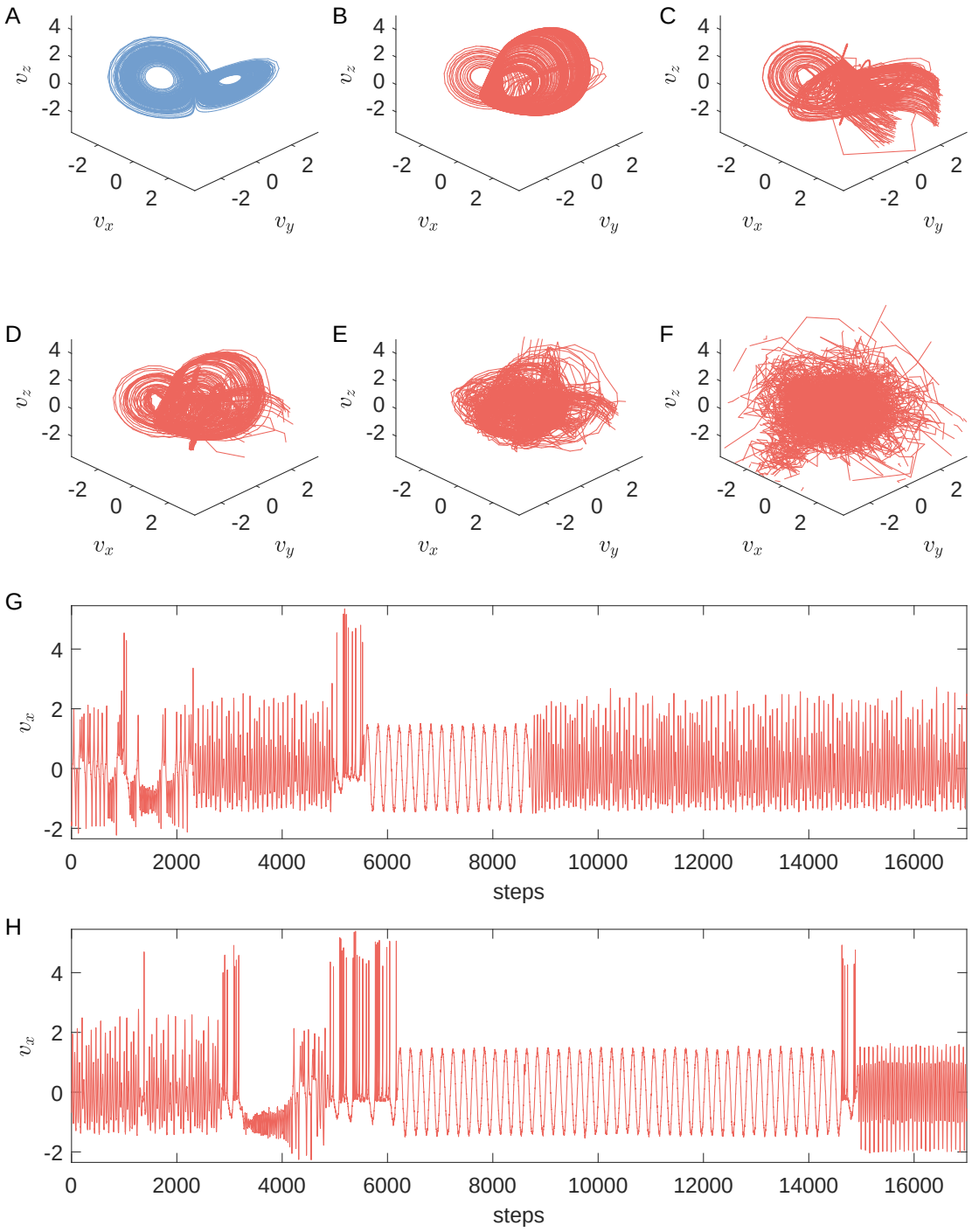

FIG. S16. Effects of noise in index-free reservoir memory. (A) Original attractor (ground truth). (B-F) Recalled attractors under different levels of noise: (B) $\sigma_n = 10^{-4.5}$, (C) $\sigma_n = 10^{-4}$, (D) $\sigma_n = 10^{-3.5}$, (E) $\sigma_n = 10^{-3}$, and (F) $\sigma_n = 10^{-2.5}$. (G-H) Two examples of random itinerary among the memorized attractors for $\sigma_n = 10^{-3.5}$. In (G), the itinerary order is: chaotic Lorenz attractor $\rightarrow$ chaotic food-chain attractor $\rightarrow$ chaotic HR neuron attractor $\rightarrow$ Lissajous attractor $\rightarrow$ chaotic food-chain attractor. In (H), the itinerary is: chaotic food-chain attractor $\rightarrow$ HR neuron attractor $\rightarrow$ chaotic Lorenz attractor $\rightarrow$ chaotic Rössler attractor $\rightarrow$ chaotic HR neuron attractor $\rightarrow$ Lissajous attractor $\rightarrow$ chaotic HR neuron attractor $\rightarrow$ a periodic attractor.

SUPPLEMENTARY REFERENCES

[1] Kong, L.-W. Codes. GitHub: https://github.com/lw-kong/Long-Term-Memory-in-RC (2024).

[2] Zhai, Z.-M., Kong, L.-W. & Lai, Y.-C. Emergence of a resonance in machine learning. *Phys. Rev. Res.* **5**, 033127 (2023).

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

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

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

[6] Lu, Z. *et al.* Reservoir observers: Model-free inference of unmeasured variables in chaotic systems. *Chaos* **27**, 041102 (2017).

[7] Flynn, A., Herteux, J., Tsachouridis, V. A., Räth, C. & Amann, A. Symmetry kills the square in a multifunctional reservoir computer. *Chaos: An Interdisciplinary Journal of Nonlinear Science* **31**, 073122 (2021).

[8] Kong, L.-W. Data and codes. GitHub: https://github.com/lw-kong/Long-Term-Memory-in-RC (2023).

[9] Sprott, J. C. Some simple chaotic flows. *Phys. Rev. E* **50**, R647–R650 (1994).

[10] Blasius, B., Huppert, A. & Stone, L. Complex dynamics and phase synchronization in spatially extended ecological systems. *Nature* **399**, 354–359 (1999).

[11] Hindmarsh, J. L. & Rose, R. M. A model of neuronal bursting using three coupled first order differential equations. *Proc. R. Soc. Lon. Ser. B Biol. Sci.* **221**, 87–102 (1984).