Machine-learning potential of a single pendulum

Swarnendu Mandal [1,*] Sudeshna Sinha [2,†] and Manish Dev Shrimali [1,‡]
[1] *Central University of Rajasthan, Ajmer, Rajasthan, India 305817*
[2] *Indian Institute of Science Education and Research Mohali, Punjab, India 140306*

(Received 4 January 2022; revised 2 April 2022; accepted 7 April 2022; published 2 May 2022)

Reservoir computing offers a great computational framework where a physical system can directly be used as computational substrate. Typically a "reservoir" is comprised of a large number of dynamical systems, and is consequently high dimensional. In this work, we use just a *single* simple low-dimensional dynamical system, namely, a driven pendulum, as a potential reservoir to implement reservoir computing. Remarkably we demonstrate, through numerical simulations as well as a proof-of-principle experimental realization, that one can successfully perform learning tasks using this single system. The underlying idea is to utilize the rich intrinsic dynamical patterns of the driven pendulum, especially the transient dynamics which has so far been an untapped resource. This allows even a single system to serve as a suitable candidate for a reservoir. Specifically, we analyze the performance of the single pendulum reservoir for two classes of tasks: temporal and nontemporal data processing. The accuracy and robustness of the performance exhibited by this minimal one-node reservoir in implementing these tasks strongly suggest an alternative direction in designing the reservoir layer from the point of view of efficient applications. Further, the simplicity of our learning system offers an opportunity to better understand the framework of reservoir computing in general and indicates the remarkable machine-learning potential of even a single simple nonlinear system.

DOI: 10.1103/PhysRevE.105.054203

I. INTRODUCTION

The ability of dynamical systems to process information has commanded long-standing interdisciplinary research interest [1–3]. There are several examples of natural systems with the capability to perform different forms of intrinsic computation [4–6]. In the context of machine learning, the overarching question is how ideas from physics or physical systems can enhance existing concepts. On one hand, research directions that can enhance the performance of algorithms in handling data from dynamical systems is a very pertinent question [7–11]. On the other hand, research efforts to utilize physical systems to implement machine-learning learning algorithms have serious implications for new concepts in the field of artificial intelligence. This line of enquiry also has consequences for gauging the information processing capacity of naturally occurring or human-engineered physical, chemical, and biological systems [12].

Here we consider the reservoir computing (RC) technique to exploit a dynamical system for machine learning. RC is a recurrent neural network (RNN) based computational framework, in which the memory capability and rich dynamics of an RNN can be used for computation without actually training the network structure itself. Instead, training the readout layer is sufficient to achieve good performance [13,14]. In this framework, the network is called the *reservoir*, as it stores the input as a high-dimensional spatiotemporal pattern, such that a linear transformation can efficiently extract the desired output in readout. Formally, a low-dimensional temporal input $u(t)$ is transformed into a much higher dimensional state vector $x(t)$ by the reservoir. These state vectors are processed further by the linear readout to get a desired output. For its simplicity, scalability, and lower training costs, reservoir computing has attracted widespread research interest, both in terms of applications [15–24] as well as basic development of the general framework [25–29].

In this article, we show through both numerical simulations and experimentation that a surprisingly simple system, namely, a single forced pendulum, has sufficient richness in its dynamics to process information for intelligent computation. The central idea is that, instead of multiplexing the input in state space, we encode the inputs in the temporal patterns, effectively making it act like a high-dimensional system [30–33]. In this work, we will assess the performance of our reservoir in the arena of both temporal and nontemporal tasks, and demonstrate that both these classes of tasks can be performed using our minimal one-node reservoir with strikingly good performance.

II. RESERVOIR DYNAMICS

Specifically, we consider a pendulum of length l with a bob of mass m, periodically driven with a force of amplitude F, depicted schematically in Fig. 1. Given a damping coefficient b of the medium, the equation of motion can be written as

$$\frac{d^2x}{dt^2} = -\frac{g}{l}\sin(x) - k\frac{dx}{dt} + f\,\mathrm{sign}\,[\sin(\omega t)], \qquad (1)$$

where the sign[·] function represents a square wave that toggles between $+1$ and -1 according to the argument being positive or negative, respectively. Here $f = \frac{F}{m}$ and $k = \frac{b}{m}$ are

*swarnenduphy35@gmail.com
†sudeshna@iisermohali.ac.in
‡shrimali@curaj.ac.in

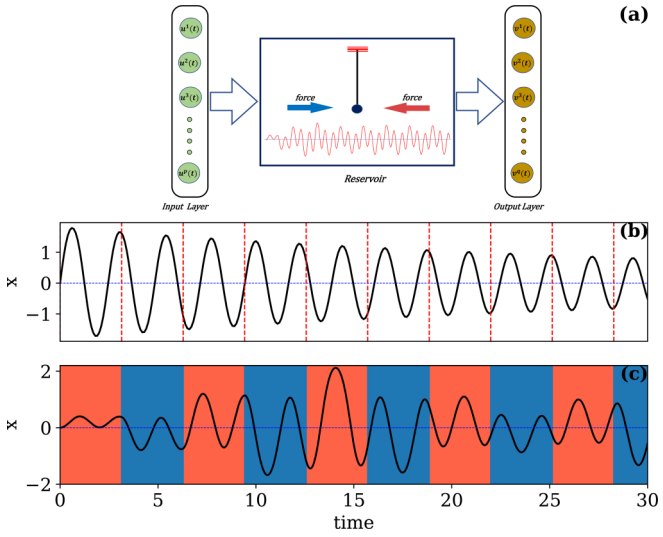

FIG. 1. (a) Schematic of the reservoir, (b) dynamics of an underdamped pendulum without forcing (solid black line) with the period of the driving force shown by dotted red lines; and (c) transient dynamics of the system (solid black line) under periodic forcing (indicated by alternating red and blue backgrounds). In the presence of periodic driving, the dynamics is quasiperiodic here, as the frequency of force is not equal to the natural frequency of the pendulum.

the amplitude of the force and damping coefficient per unit mass. This system can yield quasiperiodicity (as depicted in Fig. 1) and has been studied extensively [34–36]. With no loss of generality, we have considered $k = 5 \times 10^{-2}$ and $l = 1.0$ for the numerical simulations presented in this study.

The unique dynamics arising at each point in f-ω parameter space is evident from the bifurcation diagrams shown in Fig. 2. Unlike most studies, here we examine the temporal patterns arising, not just in the asymptotic case but in the transient

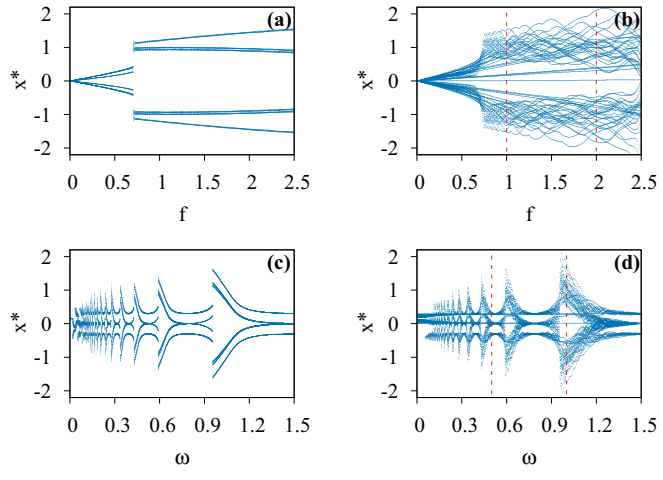

FIG. 2. Bifurcation diagram of the reservoir dynamics, (a, b) with respect to amplitude f of the driving force, with $\omega = 1.0$, and (c, d) with respect to the driving frequency ω, with $f = 1.5$. The first column (a, c) represents the asymptotic dynamics, and the second column (b, d) shows the transient dynamics starting from initial state $[x, \dot{x}] = [0, 0]$. The regions between dashed red lines are used to encode the input to the reservoir.

dynamics as well. The comparative features of the transient reservoir dynamics and the asymptotic dynamics can be seen from the two columns displayed in the figure. Clearly, the transient dynamics provides a richer repertoire of nonlinear patterns than the asymptotic behavior, and we will crucially use this aspect to encode information more efficiently.

III. INPUT ENCODING

Multiplexing the input signal efficiently into the reservoir dynamics is a crucial step for reservoir computing. The complete information should be stored into the reservoir. For our case, we have three possible choices to feed the input to the system. One option is to encode the input information with the initial condition. But this is not an efficient choice. As different trajectories can evolve to the same attractor, the preimages are not unique after transience, with different initial conditions producing the same asymptotic trajectories. Hence this will lead to input information loss and hinder robust and consistent input encoding. Alternatively, one can encode the input using the two system parameters: amplitude (f) and frequency (ω) of the applied force. Either of these two options is a better choice than input encoding with initial conditions, as each point in f-ω space gives rise to a unique dynamical sequence.

In this work we focus on tasks involving one-dimensional inputs, and hence only one parameter needs to be varied for input encoding while the other parameter can be kept fixed. We will consider input encoding using both the amplitude and the frequency of forcing in order to compare the efficacy of these two alternative methods of input encoding for different tasks, including their robustness in the presence of noise. We will denote the scheme of input encoding using forcing amplitude f as *amplitude encoding*, while the scheme where inputs are encoded using the forcing frequency ω will be simply referred to as *frequency encoding*.

First consider the amplitude encoding scheme where we multiplex the input with the amplitude of force f. In this scheme we need to choose a range of the parameter, say $f \in [f_{\min}, f_{\max}]$, and we then need to scale all input points into this range. This scaling transformation ($u \to f$) can be expressed as $f = f_{\min} + (f_{\max} - f_{\min})u$, where u is the normalized input in range [0,1]. Formally, $u = \frac{\min[\tilde{u}(t)] + \tilde{u}(t)}{\max[\tilde{u}(t)] - \min[\tilde{u}(t)]}$, $\tilde{u}(t)$ being the original input signal. Specifically, the range of f for this scheme is taken to be [1,2] as shown in Fig. 2(b). For the frequency encoding scheme, we can proceed in a similar fashion, with parameter ω replacing parameter f in the formalism.

As a testbed to gauge the performance of our system we will consider two distinct classes of tasks: one task will involve processing nontemporal signals and another task will consider processing temporal signals. In general, reservoir computing has proven to be successful in solving time-dependent data processing, stemming from the nonlinear memory effect of the reservoir. But for nontemporal tasks, we need to remove the memory effect. This can be achieved by resetting the reservoir to a fixed point after feeding an input data point to it. Specifically for our case, the reservoir is set to $[x, \dot{x}] = [0, 0]$ after each input. A step-wise detailed

discussion of the training procedure can be found in Appendix A 1 of this article.

IV. RESERVOIR STATE AND REGRESSION

The transient dynamics of the reservoir is stored in a discretized form as the state vector. Only the transient part of the dynamics is considered as it produces a richer nonlinear repertoire than its asymptotic behavior. More formally, the state vector is the set of variables $x(t)$ recorded at a fixed sampling rate $\kappa\Omega$ [$\kappa = 1, 2, 3, \ldots$], an integer multiple of the sampling cycle frequency Ω; i.e., for each sampling cycle we record κ values of $x(t)$. Thus if we store the data for N cycles for each input, states to be stored can be written as $S = [x(0),\ x(\tau),\ x(2\tau), \ldots,\ x(\kappa N\tau)]$, where $\tau = \frac{2\pi}{\kappa\Omega}$ is the sampling interval. For each point $\tilde{u}(t_i)$, $i = 1, 2, 3, \ldots$, of input signal, one reservoir state S_i is noted. Note that, after driving the reservoir with each input point therewith noting corresponding S, we reset the reservoir to $[x, \dot{x}] = [0, 0]$.

Now, for the regression, a state vector X_i is formed corresponding to each input point $u(t_i)$ using the set of noted S_i. For nontemporal tasks, X_i is a function of only the current state S_i. For that, we produce the state vector simply as the column matrix $X_i = [S_i]^T$, where $[\cdot]^T$ represents the transpose. For temporal tasks, we form the state vector corresponding to any particular input with the current states as well as states corresponding to a certain number of previous input points; i.e., we take $X_i = [w_0 S_{i-m},\ w_1 S_{i-m-1}, \ldots,\ w_{m-1} S_{i-1},\ w_m S_i]^T$, where w_j, $j = 0, 1, 2, \ldots, m$, are the weights of previous input states, following a linear distribution in the range [0,1]. The linear distribution was found to work well in this case [37–39]. Please refer to Appendix A 2 for details. Here m is the finite *memory*, which can be considered as a hyperparameter to be tuned for different kinds of temporal tasks and allows us to achieve the required *fading memory* to process temporal data. In our numerical simulations, we have considered $m = 100$.

For the two schemes of input encoding the value of Ω is different. In the amplitude encoding scheme we take $\Omega = \omega$, the frequency of driving force, and for the frequency encoding scheme we consider $\Omega = \omega_0$, the natural frequency of the oscillator. In general, Ω can be treated as another hyperparameter for both schemes.

Thus, for a complete input signal $\tilde{u}(t)$ one has the reservoir state vector matrix $\Re = [X_1,\ X_2,\ X_3, \ldots, X_L]$, L being the length of the input signal. So the matrix $\Re$ has the dimension $\kappa N \times L$ for task I and $m\kappa N \times L$ for task II, where κ and N are hyperparameters that can be optimized for best results.

Now if the corresponding output for $\tilde{u}(t)$ is $\tilde{v}(t)$, the linear transformation between the output signal and the reservoir state vector matrix can be written as $\tilde{v} = W\Re$, where W is the $(1 \times \kappa N)$-dimensional connection matrix. This matrix can be evaluated using a training data set by a regression method as $W = \tilde{v}\Re^{-1}$. Specifically for this purpose, we have used the Moore-Penrose pseudoinverse [40].

V. MACHINE-LEARNING TASKS

To check the performance of the reservoir we consider two tasks. The first task is nontemporal, and involves the learning of a high-degree polynomial. The second task involves temporal data processing, and considers the difficult task of using data from one state variable to infer another state variable in a chaotic system.

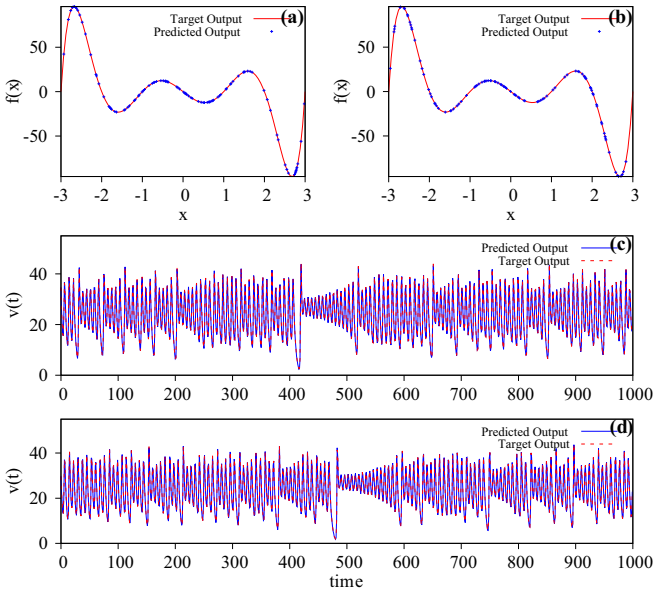

FIG. 3. The comparison of predicted output with target for (a, b) task I and (c, d) task II. (a) and (c) are the results obtained with the amplitude encoding scheme and (b) and (d) are those obtained with frequency encoding.

Specifically, the aim of task I is to approximate a seventh-degree polynomial given by $f(x) = (x-3)(x-2)(x-1)x(x+1)(x+2)(x+3)$ in the range $x \in [-3, 3]$. As this task corresponds to nontemporal input processing, one input point x is necessary and sufficient to get the corresponding output $f(x)$.

Our second task (task II) pertains to the inferring of a missing variable of a chaotic attractor dealing with temporal data processing. As an illustrative example, here we consider the state variable $x(t)$ of a chaotic Lorenz system [$\dot{x} = 10(y - x)$, $\dot{y} = x(28 - z) - y$, $\dot{z} = xy - 8z/3$] [41,42] as input to infer another state variable, $y(t)$ or $z(t)$, as output.

This class of tasks is considered to be a benchmark to test the performance of a reservoir [43,44].

VI. RESULTS

The efficiency of the reservoir is analyzed by estimating the accuracy of the tasks it performs, quantified by root mean square error (RMSE) of the predicted output with reference to the target one. We find that the reservoir works with great accuracy for both temporal and nontemporal tasks, for both schemes, as is clearly discernible from Fig. 3. The success of our single-node reservoir is also evident quantitatively from Table I, which lists the order of accuracy obtained for the tasks.

For task I, we find that a reservoir trained with only 500 data points can approximate the polynomial with RMSE of the order of 10^{-10}. Further, smaller training data sets do not significantly degrade the accuracy obtained. For instance, even

TABLE I. Comparison of performance, as quantified by the RMSE, under two different schemes of input encoding, with one method using the forcing amplitude f and the other method using forcing frequency ω to encode inputs. The first value reports the testing phase result as the order of average RMSE obtained with 100 time series samples, for noise-free systems, while the second value gives that obtained in the presence of noise.

Input encoding	Task I	Task II
f	10^{-10}, 10^{-2}	10^{-5}, 10^{-5}
ω	10^{-8}, 10^{-3}	10^{-5}, 10^{-5}

training data sets with size as low as 100 yields accuracy of the order of 10^{-6}.

For the case of task II, the reservoir was trained with data set of length 5000, and it yielded an accuracy of the order of 10^{-5}. While the prediction of both variables $y(t)$ and $z(t)$ was found to be very good, we present the result of the $x(t) \to z(t)$ prediction in Table I. For the tasks we have considered, the prediction accuracy of the trained reservoir is independent of the testing data length, and for the numerical results listed in Table I we have taken the testing data length to be the same as the training data length. So from the results it is clear that, even for the tasks involving intensive and complex information processing, the one-node reservoir predicts the output successfully.

A. Performance in presence of noise

We now assess the robustness of the performance in the presence of a noise floor. In order to examine the effect of noise on the performance, we have perturbed each state variable with a random noise, uniformly distributed in the range $[-0.01 : 0.01]$. The results are displayed in Table I. It is clear that the performance is reasonably stable even in the presence of such significantly large noise. Further, we notice that encoding inputs via the frequency of the drive is more robust and accurate than encoding inputs via the amplitude of forcing. This suggests that for optimal and most robust implementation, different control parameters for encoding information should be investigated, as the nature of the dynamics could be quite different under variation of different parameters, leading to different robustness in the presence of noise.

B. Comparison of performance with multinode reservoirs

We have also compared the efficiency of our single-node reservoir with the multinode reservoirs utilized in earlier studies. As a representative example, we have considered the reservoir of an environmentally coupled Lorenz oscillator network [45] to show the comparison in terms of the similar tasks performed by the two reservoirs. The tasks considered are the attractor reconstruction of chaotic Rössler and chaotic Chua systems, and the filtering of a Mackey-Glass time series. Table II lists the accuracy obtained for these tasks performed by both the reservoirs when trained with same training data. These results suggest that our single-node reservoir has the potential to perform better than a reservoir comprised of a large network of dynamical systems.

TABLE II. Comparison of performance, as reflected by the RMSE obtained from reservoir computing implemented by a multinode network reservoir and a single-node pendulum reservoir, for three illustrative tasks (from left to right): attractor reconstruction of a chaotic Rössler system, attractor reconstruction of a chaotic Chua system, and filtering of a Mackey-Glass time series.

Reservoir/Tasks	Rössler	Chua	Mackey-Glass
Multinode network	10^{-9}	10^{-6}	10^{-4}
Single pendulum	10^{-14}	10^{-6}	10^{-11}

C. Utilizing a single Duffing oscillator for machine learning

In order to explore the generality of our results we have investigated another low-dimensional nonlinear system that can be readily implemented in the laboratory. We have also chosen a system which allows us to gauge the comparative performance of the systems as the reservoir with different dynamical complexity. Specifically we implement reservoir computing with a single Duffing oscillator serving as a "reservoir." The dynamics is given by the evolution equations

$$\begin{aligned} \dot{x} &= y, \\ \dot{y} &= -\delta y - \beta x - \alpha x^3 + f\cos(\omega t), \end{aligned} \tag{2}$$

with parameter set chosen as follows: $\delta = 0.2, \beta = -1.0, \alpha = 1.0$, and $\omega = 1.0$. The comparison of transient and asymptotic dynamics at complexity is depicted in Fig. 4.

The results of Table III offer us a testbed for gauging the comparative performance of systems with different dynamical complexity serving as a single-node reservoir. The crucial feature we exploit here is that transient periodic and quasiperiodic behavior offers a rich repertoire of temporal sequences, while not suffering from the extreme sensitivity to initial conditions that comes alongside the complexity of chaos. So we find that the combination of stability and complexity offered by periodic and quasiperiodic transient dynamics makes this class of dynamical behavior most suited as a reservoir.

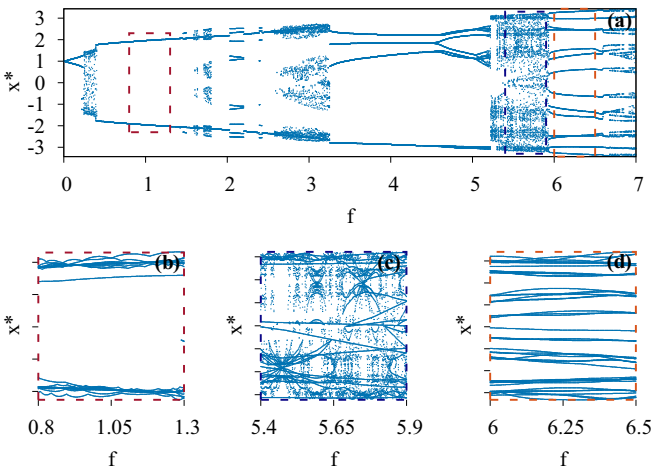

FIG. 4. (a) Bifurcation diagram of the asymptotic dynamics of the Duffing oscillator. (b–d) Bifurcation diagrams for the transient counterpart in the periodic, chaotic, and quasiperiodic regions, respectively, as marked in (a) by dashed rectangles.

TABLE III. Comparison of performance quantified by RMSE, obtained by reservoir computing implemented using a single periodic, quasiperiodic, and chaotic Duffing oscillator as a reservoir. The first row presents results obtained using asymptotic dynamics, and the second row presents results obtained using transient dynamics.

	Periodic		Quasiperiodic		Chaotic	
	Task I	Task II	Task I	Task II	Task I	Task II
Asymptotic	2×10^{-7}	3×10^{-3}	1×10^{-7}	1×10^{-2}	1×10^{0}	2×10^{0}
Transient	6×10^{-12}	2×10^{-3}	2×10^{-8}	1×10^{-3}	1×10^{-8}	7×10^{-3}

D. Proof-of-principle experiment

We have also investigated the performance of a single-node reservoir that utilizes actual laboratory data from an experimental realization of a forced pendulum. Remarkably, even this simple experimental system yields very good performance, as seen from the results displayed in Fig. 5. The detailed discussion of experimental setup and procedure is listed in Appendix B.

VII. CONCLUSIONS

In summary, we have successfully demonstrated that a single simple dynamical system, such as a pendulum, can be used effectively as a reservoir in reservoir computing. Specifically, we exploited the rich dynamics of a driven pendulum for a single-node reservoir to perform complex artificial intelligence tasks. The efficacy of our idea is demonstrated through a range of tasks, and further verified using an experimental system as a reservoir computer. Earlier attempts to implement machine learning with single classical systems have employed *time-delay systems* [32,33]. However, it is well known that such systems are effectively very high dimensional. So our central result of successfully implementing reservoir computing using just a simple, low-dimensional system, is noteworthy and surprising in its effectiveness.

In this study, we have undertaken two classes of tasks, one processing temporal signals and the other nontemporal inputs. One of the directions our work suggests is the use of the transient dynamics of nonchaotic nonlinear systems as the "reservoir" in single-node reservoir computing, as it offers both stability and complexity. The temporal patterns embedded in the transient dynamics of a nonlinear system can thus provide a rich set of transformations for the readout layer. We present results from numerical simulations, with the parameters of the dynamical system utilized as a reservoir chosen in such a way that it can be easily realized in laboratory experiments.

Importantly, this work can also be extended to deal with noisy real-world data sets containing impurities. Further, physical implementations of the idea can be potentially extended to much smaller, faster, and power-efficient systems, for instance, dynamical systems realized with integrated circuit chips. So these ideas can lead to the foundation of powerful machine-learning enabled chips.

In conclusion then, we have demonstrated that a single low-dimensional nonlinear dynamical system has remarkable potential for information processing, and can serve as a "reservoir" for reservoir computing. These results also open up the possibility of other simple dynamical systems for single-node reservoir computing, and a wide variety of natural systems can be considered as potential candidates for the reservoir. Thus this work provides a significant step forward towards the broad goal of exploiting intrinsic dynamics of natural systems for intelligent computation.

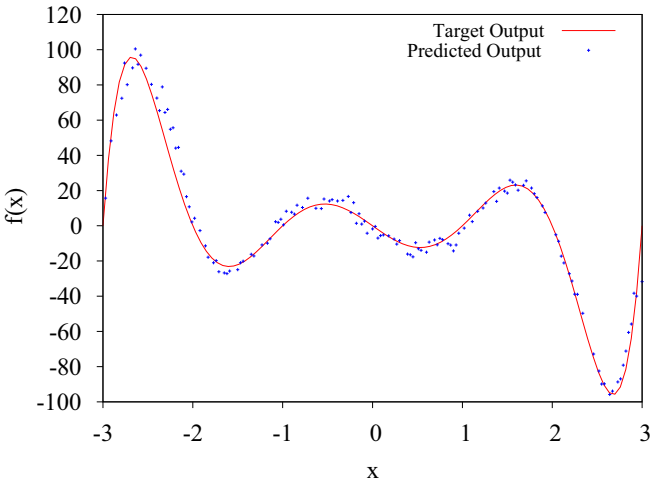

FIG. 5. Accuracy of task I, using the time series of a laboratory realization of a pendulum as a reservoir, demonstrating the ability of a simple experimental system to execute computational tasks. Here the frequency encoding scheme is used.

ACKNOWLEDGMENTS

M.D.S. acknowledge financial support from SERB, Department of Science and Technology (DST), India (Grant No. CRG/2021/003301). M.D.S. and S.M. are also supported by the Department of Science and Technology (DST), India, under the Indo-Russian Joint Research Programme (Grant No. INT/RUS/RSF/P-18). S.S. acknowledges support from the J.C. Bose National Fellowship (Grant No. JBR/2020/000004).

APPENDIX A: TRAINING OF THE RESERVOIR

1. Details of the procedure

Step 1. Produce the training data set as input $u(t) = [u(t_1), u(t_2), \ldots, u(t_L)]$ and the corresponding output $v(t) = [v(t_1), v(t_2), \ldots, v(t_L)]$.

For example, in task I, $u(t)$ is a set of x values randomly distributed over the domain $[-3 : 3]$ and $v(t)$ is corresponding $f(x)$ in the same order. For task II, $u(t)$ is the x state variable data of a chaotic Lorenz system in a prescribed time interval

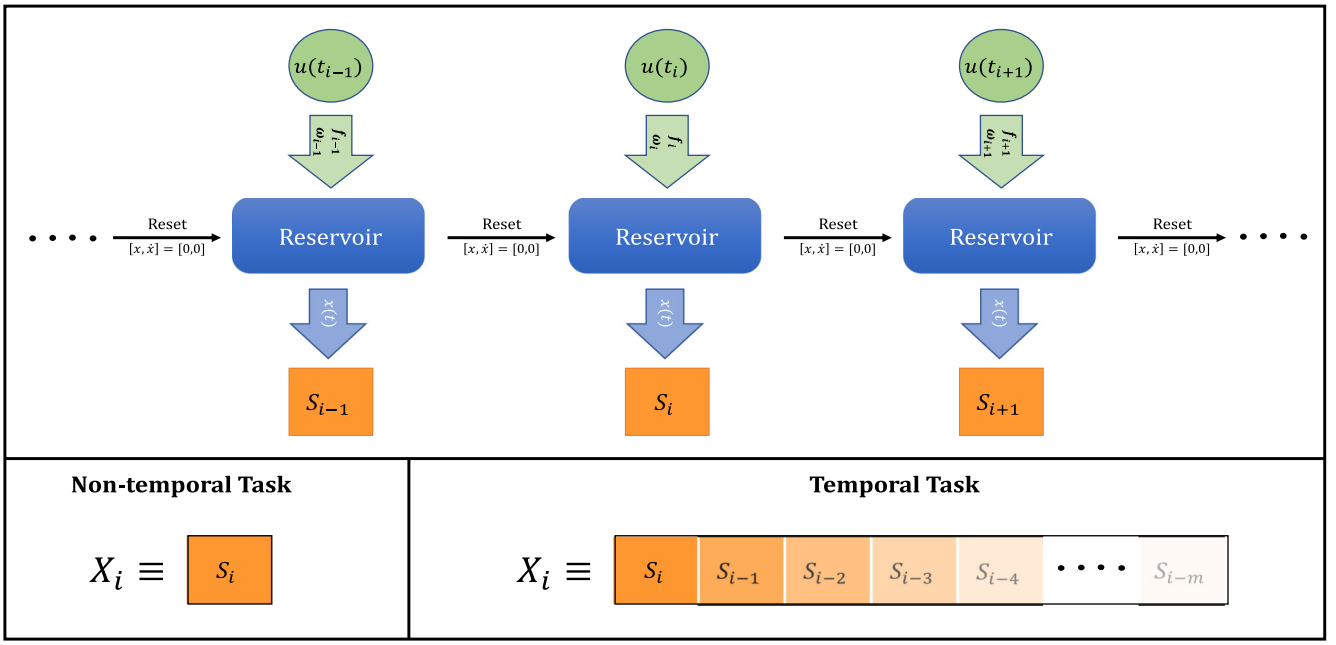

FIG. 6. Top: The schematic for the training procedure. $u(t_i)$, which refers to one particular point of input data set $u(t)$, is fed to the reservoir by setting the driving force amplitude (f) and frequency (ω) accordingly (refer to step 3). Bottom: The process to form reservoir state vector X_i using reservoir dynamics S_i, in which the fading color intensity represents decreasing weights for the temporal task case (see step 4).

(for instance, in a time interval of length 0.1) and $v(t)$ is the corresponding z state variable data.

Step 2. Scale the input $u(t)$ in a selected range of the encoding parameter (f or ω). For amplitude encoding, $u(t)$ is scaled to $[f_{\min}, f_{\max}]$ range and ω is held constant for all $u(t_i)$'s. So the updated input to the reservoir becomes $[(f_1, \omega), (f_2, \omega), \ldots, (f_L, \omega)]$ in the form of magnitude and frequency of the driving force. Similarly for frequency encoding, $u(t)$ is scaled to the $[\omega_{\min}, \omega_{\max}]$ range and f is held constant for all $u(t_i)$'s. Hence the input to the reservoir becomes $[(f, \omega_1), (f, \omega_2), \ldots, (f, \omega_L)]$.

For example consider task I performed with the amplitude encoding scheme; the range of f is taken to be $[1:2]$. So the input x in the range $[-3:3]$ needs to be scaled into $[1:2]$. Hence, the $x \to f$ scaling is defined as $f = 1 + (x+3)/6$.

Step 3. Run the reservoir starting with initial state $[x, \dot{x}] = [0, 0]$ with driving force parameters $[f_i, \omega_i]$ corresponding to input $u(t_i)$. The reservoir state $x(t)$ is recorded at a fixed sampling rate $\kappa\Omega$ [$\kappa = 1, 2, 3, \ldots$], an integer multiple of the sampling cycle frequency Ω; i.e., for each sampling cycle we record κ values of $x(t)$. Thus if we store the data for N cycles for each input, states to be stored can be written as $S = [x(0), x(\tau), x(2\tau), \ldots, x(\kappa N\tau)]$, where $\tau = \frac{2\pi}{\kappa\Omega}$ is the sampling interval.

For the two schemes of input encoding the value of Ω is different. In the amplitude encoding scheme we take $\Omega = \omega$, the frequency of driving force, and for the frequency encoding scheme we consider $\Omega = \omega_0$, the natural frequency of the oscillator. In general, Ω can be treated as another hyperparameter for both schemes.

For each point $u(t_i)$, $i = 1, 2, 3, \ldots$, of the input signal, one reservoir state S_i is noted. That is, S_i is the reservoir's transient dynamics corresponding to the parameter $[f_i, \omega_i]$ determined by a single input point $u(t_i)$. Step 3 is repeated for $i = 1, 2, 3, \ldots, L$ (see Fig. 6 for schematic).

Step 4. A reservoir state vector X_i corresponding to a single input point $u(t_i)$ is produced from stored dynamics S_i's. For nontemporal tasks, $X_i = [S_i]^T$. But for temporal tasks, $X_i = [w_0 S_{i-m}, w_1 S_{i-m-1}, \ldots, w_{m-1} S_{i-1}, w_m S_i]^T$, where w_j, $j = 0, 1, 2, \ldots, m$, are the weights of previous input states, following a linear distribution in the range [0,1]. Here m is the finite memory. In our numerical simulations, we have considered $m = 100$.

So, for temporal tasks, an initial m number of input points are used only to generate the dynamics required to achieve the memory effect to form reservoir state X. This implies the very first reservoir state X_1 corresponds to input data point $u(t_{m+1})$.

Step 5. All reservoir state vectors X_i are stacked to form state vector matrix $\Re$. Thus, for a complete input signal, one has the reservoir state vector matrix $\Re = [X_1, X_2, X_3, \ldots, X_L]$, L being the length of the input signal. The matrix $\Re$ has the dimension $\kappa N \times L$ for nontemporal tasks and $m\kappa N \times L$ for temporal tasks, where κ and N are hyperparameters that can be optimized for best results.

The output data set $v(t)$ is used for regression to find the set of output connection W. Now if the corresponding output for $u(t)$ is $v(t)$, the linear transformation between the output signal and the reservoir state vector matrix can be written as $v = W\Re$, where W is the $(1 \times \kappa N)$-dimensional connection matrix. This matrix can be evaluated using a training data set by regression method as $W = v\Re^{-1}$. For this purpose, we have specifically employed the Moore-Penrose pseudoinverse.

2. Linear memory

For sequential information processing, the memory of previous inputs is essential to be taken care of. That is the

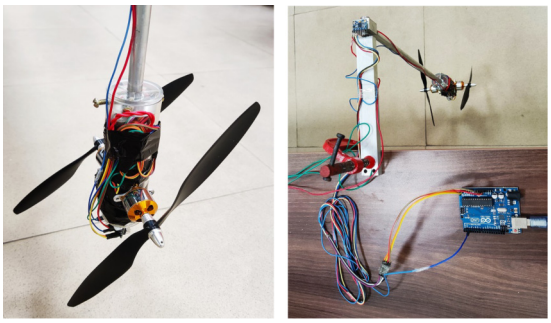

FIG. 7. Experimental setup.

reason why reservoir computing has been so successful in performing sequential tasks using its echo state properties. Information regarding any particular input is echoed into the reservoir dynamics corresponding to subsequent inputs. But for our scheme, to utilize the transient dynamics, we need to reset the reservoir after each input point multiplexing. Thus reservoir dynamics corresponding to any input becomes completely independent of each other. So to achieve the memory effect for sequential task processing, we manually stack the previous input dynamics with any particular input dynamics with gradually lesser weights than the previous ones. The weights for our case are considered to be a uniform linear distribution in the range $[0:1]$ with previous dynamics having lesser weights. The linear distribution works better than a few other nonlinear distributions like exponential decaying weights, because the memory stored in a dynamical system is degraded with increase in nonlinearity [37,38].

APPENDIX B: EXPERIMENTAL REALIZATION OF THE SINGLE PENDULUM RESERVOIR

1. Components

The components used for this experiment are listed and described below (see Fig. 7).

(1) A hollow aluminium rod of length 50 cm and cross-section diameter of 1 cm is used, with one end attached to a rigid platform to hang from by a pivot. A bob is attached to the other end.

(2) A cylindrical bob of length 6 cm and cross-section diameter 4.5 cm holds two opposite facing propellers aligned in the plane of oscillation. This also contains the control unit of propellers inside with some added weight. The total weight of the bob is 0.5 kg approximately.

(3) Two A2212/13T (1000 kV) brushless DC (BLDC) motors are attached to the bob with two 10-in. (1045) propellers each.

(4) Two 30-A electronic speed controllers (ESCs), kept inside the bob, are used for controlling the speed of two motors by a microcontroller.

(5) A microcontroller (Arduino Nano) attached with the bob is used to receive wireless data and pass it to the ESCs. This is actually the part of the circuit responsible for generating the driving force function.

(6) An HC 05 Bluetooth module enables the possibility to receive control input wirelessly without affecting the natural dynamics.

(7) An MPU-6050 GY-521 gyro sensor is attached to the pendulum near the pivot to collect the angular deflection data of the pendulum with time. This sensor is interfaced with another microcontroller via fine flexible wires.

(8) An Arduino Uno is used to receive and decode the gyro sensor data and to communicate with the computer for calculation.

(9) An external DC power supply is used to power the whole setup.

2. Experimental steps

(a) Step I. We start by setting up the system with all the components and required circuit connections.

(b) Step II. First, we need to calibrate two brushless DC motors. The controllers (ESCs) take a high-frequency square wave signal input and according to its pulse width the speed is decided. So, by the microcontroller we need to generate a pulse width modulated (PWM) signal specifying the width in the range [0,180]. Basically, 0 corresponds to no rotation and 180 to full speed. Now, the speed decides the amount of force exerted (F) by the propellers. We need to find out a relation between $f = F/m$ and the input pulse width for the pulse width modulation.

This is done using a simple setup. Say, for pulse width p a propeller generates a force F and due to that the pendulum rests at an angular displacement θ. In this case, $F = mg\sin(\theta)$ or $f = g\sin(\theta)$. Hence, for any p we can find the value of f. Repeating this process with both the motors a sufficient number of times with the setup, and fitting the data to straight lines, we can find calibration curves for any value of f for the two motors.

(c) Step III. We need to program the microcontrollers according to the requirement of operation. The Arduino Nano, attached to the bob, should receive the wireless signal of the ESC inputs and the ω value to generate the driving force function. Similarly, the Arduino Uno interfacing the gyro sensor should be programed to sample data at rate defined by τ.

(d) Step IV. The setup is run with required inputs and the gyro sensor data from the Arduino Uno are stored. Reservoir states are generated for both training and testing data inputs.

(e) Step V. Training reservoir states are used for regression with their corresponding output.

(f) Step VI. Using the optimal output weight evaluated by regression, test reservoir states are used to find the output, and the predicted output is compared with the target output.

3. Drop in performance with the amplitude encoding scheme

Figure 5 of the article shows the results from this experiment using the frequency encoding scheme. However, there is a significant effect of noise on the performance when the amplitude encoding scheme is used. The factors that affect the performance under the amplitude encoding scheme are rationalized as follows. To use the amplitude encoding scheme one needs to multiplex the input with the amplitude of force. In the experimental setup, we can control only the pulse width of the ESC input, and two transformations need to be implemented. First, the pulse width information is converted into speed, and second, according to varying speeds, different magnitudes of the reaction force generated by the propellers are exerted

on the system. So there are many potential factors affecting the control of input force, such as the electronic or thermal noise affecting the ESCs and air density, ambient temperature, the environment's aerodynamics, and so on. So there is no direct control over the forcing amplitude, i.e., the value of f. On the other hand, the frequency encoding scheme simply encodes input information using the frequency of the force ω, on which there is a direct control. Since the frequency can be controlled with a precision of $\sim(\mu s)^{-1}$ by the microcontroller, the frequency encoding scheme yields better results.

[1] R. Shaw, Strange attractors, chaotic behavior, and information flow, Z. Naturforsch. A **36**, 80 (1981).

[2] S. Sinha and W. L. Ditto, Dynamics Based Computation, Phys. Rev. Lett. **81**, 2156 (1998).

[3] T. Munakata, S. Sinha, and W. Ditto, Chaos computing: Implementation of fundamental logical and arithmetic operations and memory by chaotic elements, IEEE Trans. Circuit Syst. **49**, 1629 (2002).

[4] J. P. Crutchfield, W. L. Ditto, and S. Sinha, Introduction to focus issue: Intrinsic and designed computation: Information processing in dynamical systems–beyond the digital hegemony, Chaos: Interdiscip. J. Non. Sci. **20**, 037101 (2010).

[5] K. Mainzer, *Thinking in Complexity: The Computational Dynamics of Matter, Mind, and Mankind* (Springer, Berlin, Heidelberg, 2007).

[6] T. Toffoli, Nothing makes sense in computing except in the light of evolution, Int. J. Unconv. Comput. **1**, 3 (2004).

[7] A. Choudhary, J. F. Lindner, E. G. Holliday, S. T. Miller, S. Sinha, and W. L. Ditto, Physics-enhanced neural networks learn order and chaos, Phys. Rev. E **101**, 062207 (2020).

[8] S. T. Miller, J. F. Lindner, A. Choudhary, S. Sinha, and W. L. Ditto, The scaling of physics-informed machine learning with data and dimensions, Chaos, Solitons Fractals: X **5**, 100046 (2020).

[9] A. Choudhary, J. Lindner, E. Holliday, S. Miller, S. Sinha, and W. Ditto, Forecasting Hamiltonian dynamics without canonical coordinates, Nonlinear Dyn. **103**, 1553 (2021).

[10] C.-D. Han, B. Glaz, M. Haile, and Y.-C. Lai, Adaptable Hamiltonian neural networks, Phys. Rev. Research **3**, 023156 (2021).

[11] J. Meiyazhagan, S. Sudharsan, and M. Senthilvelan, Model-free prediction of emergence of extreme events in a parametrically driven nonlinear dynamical system by deep learning, Eur. Phys. J. B **94**, 156 (2021).

[12] D. Beniaguev, I. Segev, and M. London, Single cortical neurons as deep artificial neural networks, Neuron **109**, 2727 (2021).

[13] H. Jaeger, The echo state approach to analysing and training recurrent neural networks-with an erratum note, Bonn, Germany: German National Research Center for Information Technology GMD Technical Report **148** (2001), p. 13.

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

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

[16] M. Lukoševičius, H. Jaeger, and B. Schrauwen, Reservoir computing trends, Künstliche Intelligenz **26**, 365 (2012).

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

[18] K. Nakajima, Physical reservoir computing—an introductory perspective, Jpn. J. Appl. Phys. **59**, 060501 (2020).

[19] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach, Phys. Rev. Lett. **120**, 024102 (2018).

[20] Y. Zhong, J. Tang, X. Li, B. Gao, H. Qian, and H. Wu, Dynamic memristor-based reservoir computing for high-efficiency temporal signal processing, Nat. Commun. **12**, 1 (2021).

[21] M. Rafayelyan, J. Dong, Y. Tan, F. Krzakala, and S. Gigan, Large-Scale Optical Reservoir Computing for Spatiotemporal Chaotic Systems Prediction, Phys. Rev. X **10**, 041037 (2020).

[22] S. Ghosh, A. Senapati, A. Mishra, J. Chattopadhyay, S. K. Dana, C. Hens, and D. Ghosh, Reservoir computing on epidemic spreading: A case study on COVID-19 cases, Phys. Rev. E **104**, 014308 (2021).

[23] S. Saha, A. Mishra, S. Ghosh, S. K. Dana, and C. Hens, Predicting bursting in a complete graph of mixed population through reservoir computing, Phys. Rev. Research **2**, 033338 (2020).

[24] H. Zhang, H. Fan, L. Wang, and X. Wang, Learning Hamiltonian dynamics with reservoir computing, Phys. Rev. E **104**, 024205 (2021).

[25] P. R. Vlachas, J. Pathak, B. R. Hunt, T. P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos, Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics, Neural Networks **126**, 191 (2020).

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

[27] N. A. Silva, T. D. Ferreira, and A. Guerreiro, Reservoir computing with solitons, New J. Phys. **23**, 023013 (2021).

[28] T. L. Carroll, Do reservoir computers work best at the edge of chaos?, Chaos **30**, 121109 (2020).

[29] L. C. G. Govia, G. J. Ribeill, G. E. Rowlands, H. K. Krovi, and T. A. Ohki, Quantum reservoir computing with a single nonlinear oscillator, Phys. Rev. Research **3**, 013077 (2021).

[30] J. H. Jensen and G. Tufte, Reservoir computing with a chaotic circuit, in *Artificial Life Conference Proceedings 14* (MIT Press, Cambridge, MA, 2017), pp. 222–229.

[31] L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer, Information processing using a single dynamical node as complex system, Nat. Commun. **2**, 1 (2011).

[32] G. Dion, S. Mejaouri, and J. Sylvestre, Reservoir computing with a single delay-coupled non-linear mechanical oscillator, J. Appl. Phys. **124**, 152132 (2018).

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

[34] F. J. Romeiras and E. Ott, Strange nonchaotic attractors of the damped pendulum with quasiperiodic forcing, Phys. Rev. A **35**, 4404 (1987).

[35] A. Bondeson, E. Ott, and T. M. Antonsen, Quasiperiodically Forced Damped Pendula and Schrödinger Equations with Quasiperiodic Potentials: Implications of their Equivalence, Phys. Rev. Lett. **55**, 2103 (1985).

[36] M. Ding, C. Grebogi, and E. Ott, Evolution of attractors in quasiperiodically forced systems: From quasiperiodic to strange nonchaotic to chaotic, Phys. Rev. A **39**, 2593 (1989).

[37] D. Verstraeten, J. Dambre, X. Dutoit, and B. Schrauwen, Memory versus non-linearity in reservoirs, in *Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN)* (IEEE, Piscataway, NJ, 2010), pp. 1–8.

[38] M. Inubushi and K. Yoshimura, Reservoir computing beyond memory-nonlinearity trade-off, Sci. Rep. **7**, 1 (2017).

[39] R. Falahian, M. Mehdizadeh Dastjerdi, M. Molaie, S. Jafari, and S. Gharibzadeh, Artificial neural network-based modeling of brain response to flicker light, Nonlinear Dyn. **81**, 1951 (2015).

[40] J. C. A. Barata and M. S. Hussein, The Moore-Penrose pseudoinverse: A tutorial review of the theory, Braz. J. Phys. **42**, 146 (2012).

[41] E. N. Lorenz, Deterministic nonperiodic flow, J. Atmos. Sci. **20**, 130 (1963).

[42] C. Sparrow, *The Lorenz Equations: Bifurcations, Chaos, and Strange Attractors*, Vol. 41 (Springer Science & Business Media, 2012).

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

[44] J. Choi and P. Kim, Reservoir computing based on quenched chaos, Chaos Solitons Fractals **140**, 110131 (2020).

[45] S. Mandal and M. D. Shrimali, Achieving criticality for reservoir computing using environment-induced explosive death, Chaos **31**, 031101 (2021).