Reservoir computing

Karol Bednarz

October 26, 2025

Contents

1 State of the art in reservoir computing

1.1 ESN - Echo State Network

1.1.1 Principles of reservoir computing

Based on the [1]: The state of reservoir dynamics can be expressed as:

$$h_{\mathrm{t}} = f\big((1-k)\cdot u_{\mathrm{t}}\cdot W_{\mathrm{in}} + k\cdot h_{\mathrm{t-1}}\cdot W_{\mathrm{h}} + y_{\mathrm{t-1}}\cdot W_{\mathrm{1b}} + b\big) \tag{1}$$

Where:

$h_{\mathrm{t-1}}$ – are the reservoir state respectively, from the previous time step,
u_{t} – is the observed data at time step t,
$y_{\mathrm{t-1}}$ – is the the predicted output state $t-1$,
$W_{\mathrm{in}} \in \mathbb{R}^{N_{\mathrm{u}}\times N_{\mathrm{h}}}$ – is the input weight matrix,
$W_{\mathrm{h}} \in \mathbb{R}^{N_{\mathrm{h}}\times N_{\mathrm{h}}}$ – is the internal weight matrix,
$W_{\mathrm{1b}} \in \mathbb{R}^{N_{\mathrm{h}}\times N_{\mathrm{y}}}$ – is the output fedback weight matrix,
$b \in \mathbb{R}^{N_{\mathrm{h}}}$ – is the bias vector.
f – is the activation function, typically tanh or sigmoid,
k – is the leaking rate, typically $k \in [0.1, 0.3]$.

With the computed reservoir dynamics, the output can be then obtained by:

$$y_{\mathrm{t}} = h_{\mathrm{t}} \cdot W_{\mathrm{out}} \tag{2}$$

Where:

$W_{\mathrm{out}} \in \mathbb{R}^{N_{\mathrm{h}} \times N_{\mathrm{y}}}$ – is the output weight matrix.

1.1.2 Echo state property

Any system that changes in a nonlinear way can work as the reservoir. However, starting a nonlinear system with random connection strengths creates problems. The reservoir is a system that feeds its outputs back into itself. This can make it unstable if the connection strengths aren't set up correctly. For example, if the internal connections are too strong, the system might get stuck giving the same output regardless of what input it receives. The random connection strengths must be chosen so the system doesn't grow out of control. For the system to work well, it must follow something called the "echo state property." This rule ensures that the reservoir's behavior eventually depends on the input signal rather than just its starting conditions. To meet this requirement, the internal connection matrix W_{h} is first set up using random values between -1 and 1. This matrix is then adjusted one time according to the echo state property rule:

$$W'_{\mathrm{h}} = \alpha \odot W_{\mathrm{h}} \tag{3}$$

$$W^{\dagger}_{\mathrm{h}} = \frac{\rho W_{\mathrm{h}}}{|\lambda_{\max}(W_{\mathrm{h}})|} \tag{4}$$

Where:

$\rho \in (0, 1)$ – is the spectral radius, typically $\rho \in [0.9, 1]$
$\lambda_{\max}(W_{\mathrm{h}})$ – is the largest eigenvalue of W_{h}.
$\alpha \in (0, 1)$ – is the sparsity coefficient, typically $\alpha \in [0.1, 0.3]$.

The spectral radius is a parameter that determines the amount of nonlinear interaction of input components through time.

Due to the recursive nature of the reservoir layer, such dynamics reflect trajectories of the past historical inputthe short-term memory (known as the fading memory). As another critical property for computing the RC principle, short-term memory can be quantitative measured by the coefficient of memory capacity

$$MC = \sum_{\mathrm{k}=1}^{\infty} MC_{\mathrm{k}} = \sum_{\mathrm{k}=1}^{\infty} d^2(u_{\mathrm{t-k}}, y_{\mathrm{t}}) = \sum_{\mathrm{k}=1}^{\infty} \frac{\mathrm{cov}^2(u_{\mathrm{t-k}}, y_{\mathrm{t}})}{\sigma^2(u_{\mathrm{t}})\sigma^2(y_{\mathrm{t}})} \tag{5}$$

Where:

$d^2(u_{\mathrm{t-k}}, y_{\mathrm{t}})$ – is the square of the correlation coefficient between the output y_{t} and the input $u_{\mathrm{t-k}}$ with a delay of k time steps,

According to the Lyapunov stability analysis, a large memory capacity is needed to compute the RC principle, which can be achieved at the asymptotically stable region.

1.1.3 Learning algorithm

The training of the reservoir computing model involves adjusting only the output weights W_{out}. The input weights W_{in}, internal weights W_{h}, and feedback weights $W_{1\text{b}}$ are typically initialized randomly and remain fixed during training. The training process can be summarized in the following steps:

$$Y = H \cdot W_{\text{out}} \tag{6}$$

Where:

$Y \in \mathbb{R}^{T \times N_{\text{y}}}$ – is the matrix of target outputs for all time steps,
$H \in \mathbb{R}^{T \times N_{\text{h}}}$ – is the matrix of reservoir states for all time steps,
T – is the total number of time steps.

In general, the W_{out} can be directly obtained by calculating the Moore-Penrose pseudoinverse of the reservoir states matrix H with respect to the target outputs matrix Y:

$$W_{\text{out}} = Y \cdot H^T \cdot (H \cdot H^T + \eta I)^{-1} \tag{7}$$

Where:

H^T – is the transpose of matrix H,
η – is the regularization parameter, typically $\eta \in [10^{-6}, 10^{-2}]$,
I – is the identity matrix of size $N_{\text{h}} \times N_{\text{h}}$.

1.1.4 Model development

The echo state network (ESN) and the liquid state machine (LSM) are the two representations of RC. While ESN and LSM are topographically equivalent, the former adopts the actual numerical data from the input to compute the network dynamics, and the latter adopts the spiking signal to represent the spatiotemporal pattern. **The LSM can be also seen as a SNN, where the reservoir has numerous leaky integrate-and-fire (LIF) neurons interconnected with the same recursive nature as in ESN**. As the spiking event is involved, the training of LSM generally relies on the spike-timing-dependent plasticity (STDP) and short-term plasticity (STP), among other training methods for spiking networks.

1.2 Time-delayed reservoir computing

Based on [5, 9, 8].

The model of reservoir can be written in the following general form:

$$\mathbf{r}(n) = F\big(\gamma \mathbf{W}_{\text{in}} \mathbf{x}(n) + \beta \mathbf{W} \mathbf{r}(n-1)\big), \tag{8}$$
$$\mathbf{y}(n) = \mathbf{W}_{\text{out}} \mathbf{r}(n). \tag{9}$$

Where:

$\mathbf{r}(n) \in \mathbb{R}^N$ – is the reservoir state at time step n,
$\mathbf{x}(n) \in \mathbb{R}^d$ – is the input vector at time step n,
$\mathbf{y}(n) \in \mathbb{R}^p$ – is the output vector at time step n,
$\mathbf{W}_{\text{in}} \in \mathbb{R}^{N \times d}$ – is the input weight matrix, usually drawn from a random distribution $[-1, 1]$,
$\mathbf{W} \in \mathbb{R}^{N \times N}$ – is the internal weight matrix, usually drawn from a random distribution $[-1, 1]$,
$\mathbf{W}_{\text{out}} \in \mathbb{R}^{p \times N}$ – is the output weight matrix,

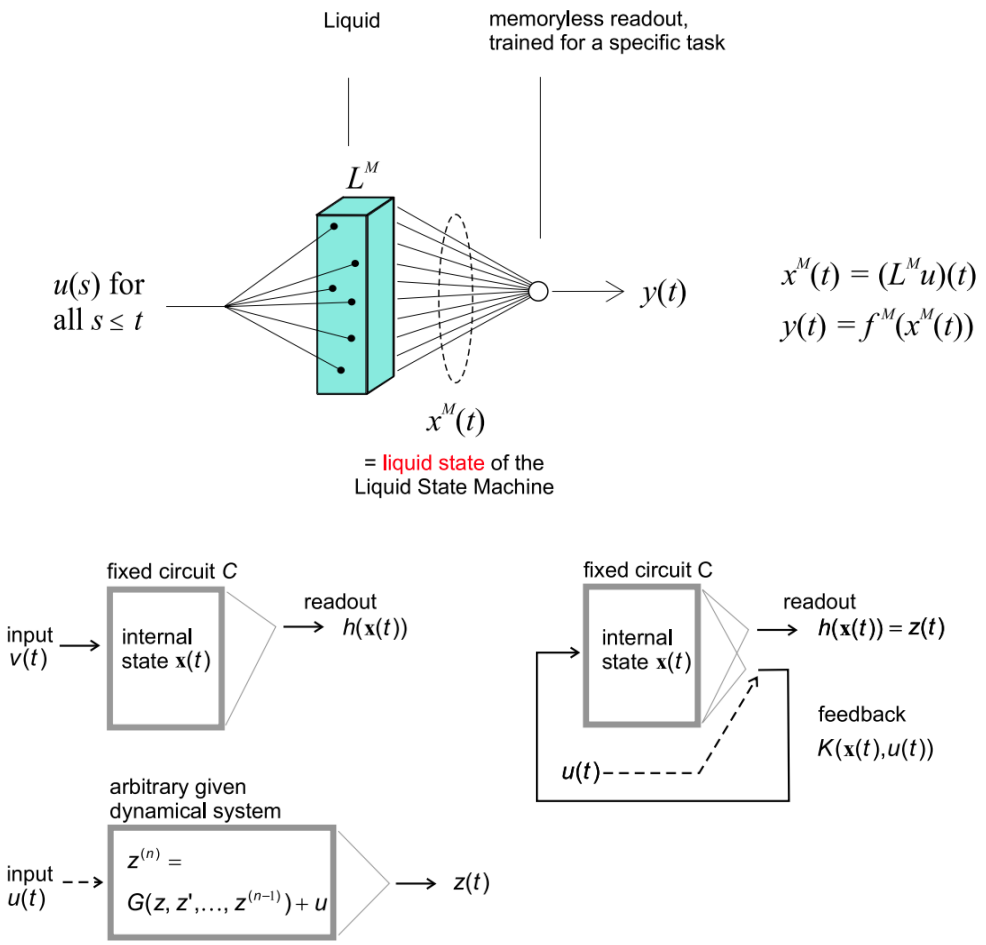

Figure 1: Liquid State Machine architecture

$\gamma, \beta \in \mathbb{R}^+$ – are the input and feedback scaling parameters,
$F : \mathbb{R}^N \to \mathbb{R}^N$ – is the nonlinear activation function, typically tanh or sigmoid.

In general, the time-delayed reservoir computing (TDRC) can be implemented using a single nonlinear node with delayed feedback loop instead of a large network of interconnected nodes. The delay dynamics create a high-dimensional state space that can be exploited for computation.

1.3 Liquid State Machine – Reservoir with spiking neurons

Based on [7, 2].

1.4 Next generation reservoir computing

Based on [4].

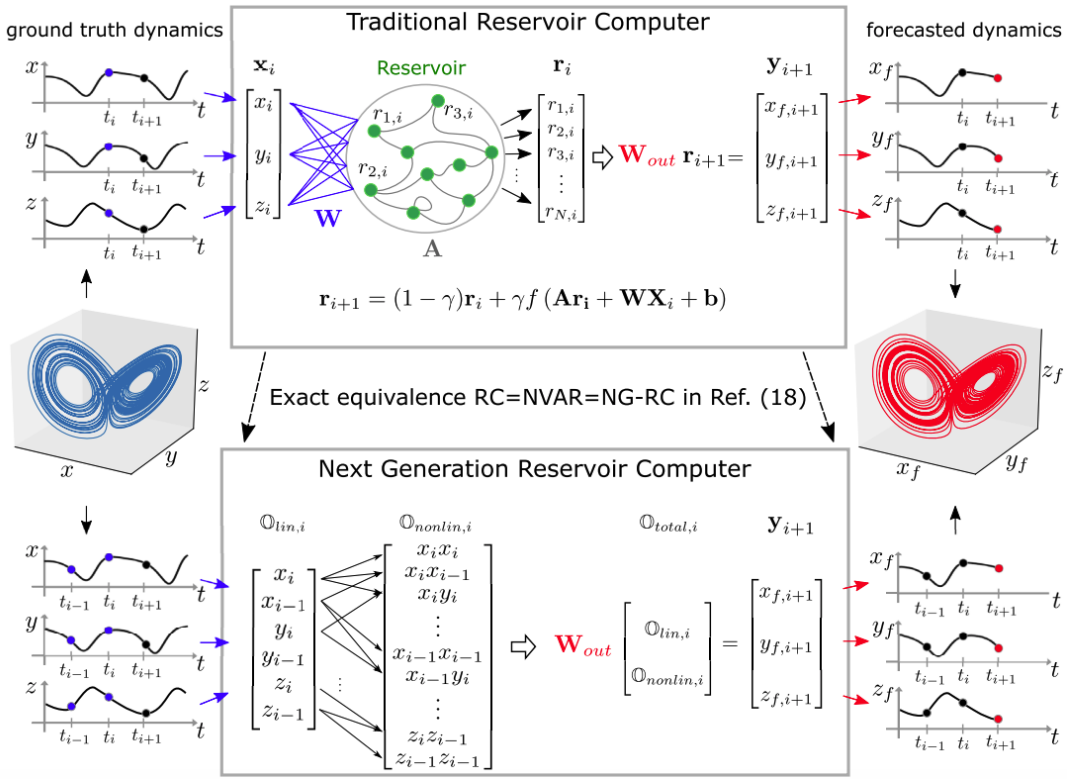

Figure 2: A traditional RC is implicit in an NG-RC. (top) A traditional RC processes time-series data associated with a strange attractor (blue, middle left) using an artificial recurrent neural network. The forecasted strange attractor (red, middle right) is a linear weight of the reservoir states. (bottom) The NGRC performs a forecast using a linear weight of time-delay states (two times shown here) of the time series data and nonlinear functionals of this data (quadratic functional shown here).

The reservoir is a dynamic system whose dynamics can be represented by:

$$\mathbf{r}_{\mathrm{i}+1} = (1-\gamma)\mathbf{r}_{\mathrm{i}} + \gamma f\big(\mathbf{A}r_{\mathrm{i}} + \mathbf{W}_{\mathrm{in}}\mathbf{X}_{\mathrm{i}} + b\big) \tag{10}$$

Where:
$r_{\mathrm{i}} = [r_{\mathrm{i}}^1, r_{\mathrm{i}}^2, \ldots, r_{\mathrm{i}}^N]^T \in \mathbb{R}^{N\times 1}$ is the reservoir state at time step i,
The output layer expresses the RC output $\mathbf{Y}_{\mathrm{i}+1}$ as a linear transformation of a feature vector $\mathbf{O}_{\mathrm{total,i}+1}$, constructed from the reservoir state $\mathbf{r}_{\mathrm{i}+1}$, through the relation:

$$\mathbf{Y}_{\mathrm{i}+1} = \mathbf{W}_{\mathrm{out}}\mathbf{O}_{\mathrm{total,i}+1} \tag{11}$$

Where:

$\mathbf{O}_{\mathrm{total,i}+1} \in \mathbb{R}^{M\times 1}$ – is the feature vector at time step $i+1$,
$\mathbf{W}_{\mathrm{out}} \in \mathbb{R}^{p\times M}$ – is the output weight matrix,
M – is the size of the feature vector.

The RC is trained using supervised training via regularized least-squares regression. Here, the training data points generate a block of data contained in $\mathbf{O}_{\mathrm{total}}$ and we match $\mathbf{Y}$ to the desired output $\mathbf{Y}_{\mathrm{d}}$ in a least-square sense using Tikhonov regularization so that $\mathbf{W}_{\mathrm{out}}$ is given by

$$\mathbf{W}_{\mathrm{out}} = \mathbf{Y}_{\mathrm{d}}\mathbf{O}_{\mathrm{total}}^T(\mathbf{O}_{\mathrm{total}}\mathbf{O}_{\mathrm{total}}^T + \alpha\mathbf{I})^{-1} \tag{12}$$

A different approach to RC is to move the nonlinearity from the reservoir to the output layer. In this case, the reservoir nodes are chosen to have a linear activation function $f(\mathbf{r}) = \mathbf{r}$, while the feature vector $\mathbf{O}_{\text{total}}$ becomes nonlinear. A simple example of such RC is to extend the standard linear feature vector to include the squared values of all nodes, which are obtained through the Hadamard product $\mathbf{r} \odot \mathbf{r} = [r_1^2, r_2^2, \ldots, r_{\text{N}}^2]^T$

$$\mathbf{O}_{\text{total}} = \mathbf{r} \oplus (\mathbf{r} \odot \mathbf{r}) = [r_1, r_2, \ldots, r_{\text{N}}, r_1^2, r_2^2, \ldots, r_{\text{N}}^2]^T \tag{13}$$

Where:

$\oplus$ – is the concatenation operator.

Here, $\mathbf{O}_{\text{total}} = c \cdot \mathbf{O}_{\text{linear}} \oplus \mathbf{O}_{\text{nonlinear}}$, where c is a constant and $\mathbf{O}_{\text{nonlinear}}$ is a nonlinear part of the feature vector. Like a traditional RC, the output is obtained using these features.

The linear features $\mathbf{O}_{\text{linear,i}}$ at time step i is composed of observations of the input vector $\mathbf{X}$ at the current and at $k-1$ previous times steps spaced by s, where $(s-1)$ is the number of skipped steps between consecutive observations. If $\mathbf{X}_{\text{i}} = [x_{1,\text{i}}, x_{2,\text{i}}, \ldots, x_{\text{d,i}}]^T$ is a d-dimensional vector, $\mathbf{O}_{\text{linear,i}}$ has $d \cdot k$ components, and is given by

$$\mathbf{O}_{\text{linear,i}} = \mathbf{X}_{\text{i}} \oplus \mathbf{X}_{\text{i-s}} \oplus \mathbf{X}_{\text{i-2s}} \oplus \ldots \oplus \mathbf{X}_{\text{i-(k-1)s}} \tag{14}$$

Based on the general theory of universal approximators, k should be taken to be infinitely large. However, it is found in practice that the Volterra series converges rapidly, and hence truncating k to small values does not incur large error. This can also be motivated by considering numerical integration methods of ordinary differential equations where only a few subintervals (steps) in a multistep integrator are needed to obtain high accuracy. We do not subdivide the step size here, but this analogy motivates why small values of k might give good performance in the forecasting tasks considered below.

An important aspect of the NG-RC is that its warm-up period only contains sk time steps, which are needed to create the feature vector for the first point to be processed.

The nonlinear part $\mathbf{O}_{\text{nonlinear}}$ of the feature vector is a nonlinear function of $\mathbf{O}_{\text{linear}}$. While there is great flexibility in choosing the nonlinear functionals, we find that polynomials provide good prediction ability. Polynomial functionals are the basis of a Volterra representation for dynamical systems and hence they are a natural starting point. We find that low-order polynomials are enough to obtain high performance. All monomials of the quadratic polynomial, for example, are captured by the outer product $\mathbf{O}_{\text{linear}} \otimes \mathbf{O}_{\text{linear}}$, which is a symmetric matrix with $(dk)^2$ elements. A quadratic nonlinear feature vector $\mathbf{O}^{(2)}_{\text{nonlinear}}$, for example, is composed of the $(dk)(dk+1)/2$ unique monomials of $\mathbf{O}_{\text{linear}} \otimes \mathbf{O}_{\text{linear}}$, which are given by the upper triangular elements of the outer product tensor. We define $\lceil \otimes \rceil$ as the operator that collects the unique monomials in a vector. Using this notation, a p-order polynomial feature vector is given by

$$\mathbf{O}^{(p)}_{\text{nonlinear}} = \underbrace{\mathbf{O}_{\text{linear}} \lceil \otimes \rceil \mathbf{O}_{\text{linear}} \lceil \otimes \rceil \ldots \lceil \otimes \rceil \mathbf{O}_{\text{linear}}}_{\text{p times}} \tag{15}$$

The total feature vector used for the Lorenz63 forecasting task is given by

$$\mathbf{O}_{\text{total}} = c \cdot \mathbf{O}_{\text{linear}} \oplus \mathbf{O}_{\text{linear}} \oplus \mathbf{O}^{(2)}_{\text{nonlinear}} \tag{16}$$

To allow the NG-RC to focus on the subtle details of this process, we use a simple Euler-like integration step as a lowest-order approximation to a forecasting step by modifying Eq. 2 so that the NG-RC learns the difference between the current and future step. To this end, Eq. 2 is replaced by

$$\mathbf{X}_{\text{i+1}} = \mathbf{X}_{\text{i}} + \mathbf{W}_{\text{out}} \mathbf{O}_{\text{total,i}} \tag{17}$$

2 Memristors in reservoir computing

2.1 Reservoir computing using dynamic memristors for temporal information processing

Based on [3, 6].

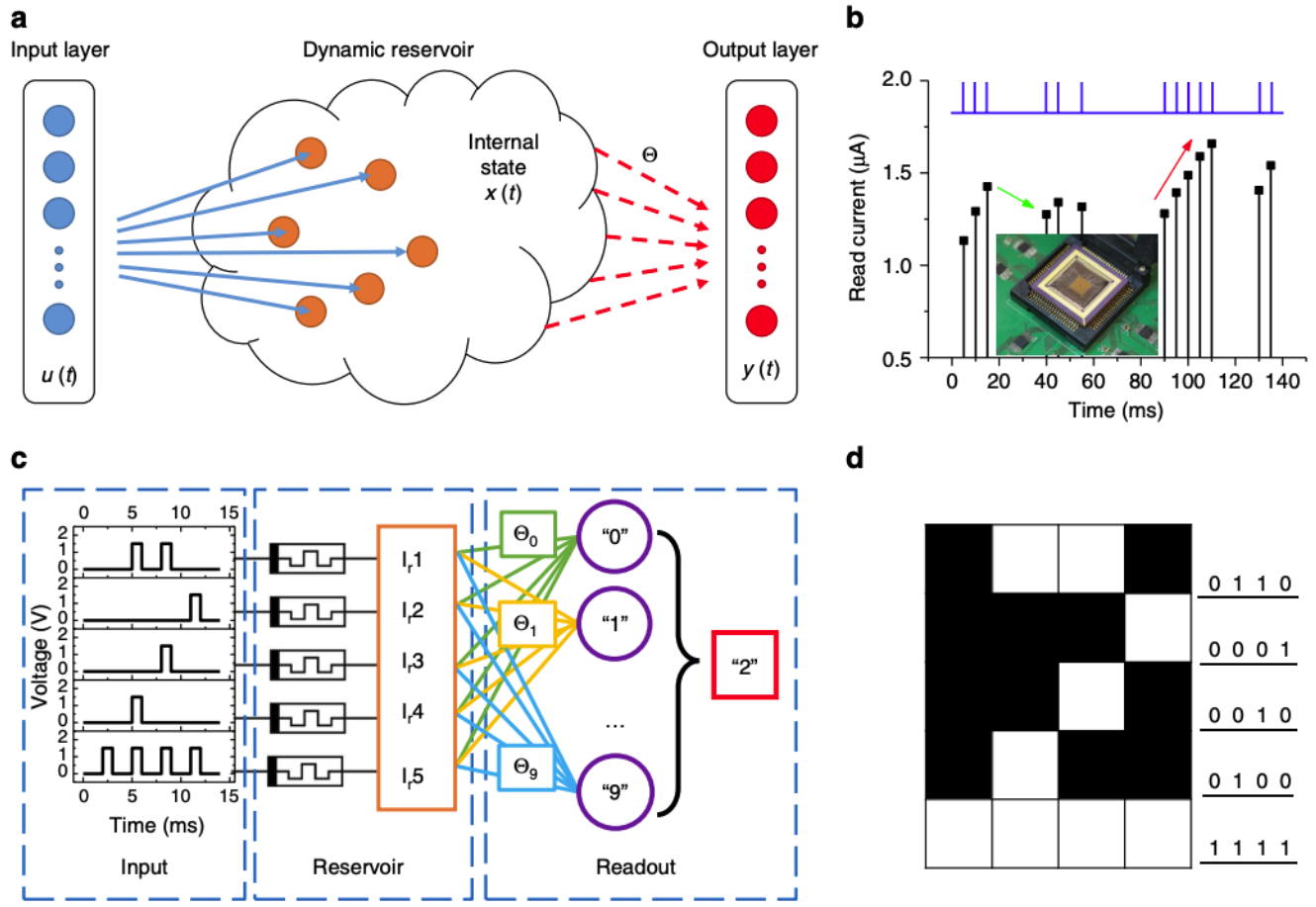

Figure 3: Reservoir computing system based on a memristor array. (a) Schematic of an RC system, showing the reservoir with internal dynamics and a readout function. Only the weight matrix Θ connecting the reservoir state x(t) and the output y(t) needs to be trained. (b) Response of a typical WOx memristor to a pulse stream with different time intervals between pulses. Inset: image of the memristor array wired-bonded to a chip carrier and mounted on a test board. (c) Schematic of the RC system with pulse streams as the inputs, the memristor reservoir and a readout network. For the simple digit recognition task of 5×4 images, the reservoir consists of 5 memristors. (d) An example of digit 2 used in the simple digit analysis.

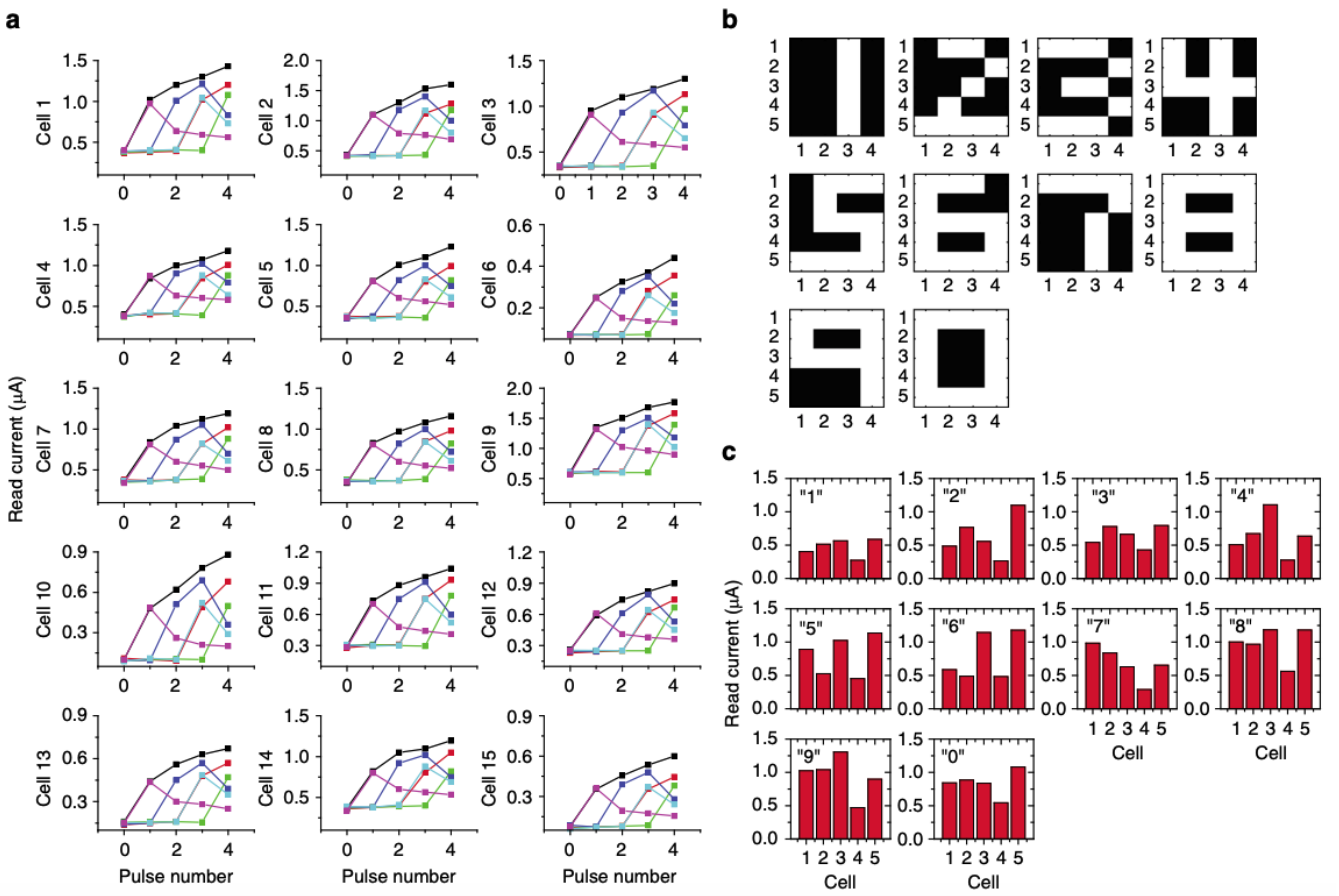

Figure 4: Reservoir states used to differentiate different temporal inputs. (a) The response of 15 memristors from the array to 6 different pulse streams (black: [1 1 1 1], purple: [1 0 0 0], blue: [0 1 1 0], red: [0 0 1 1], cyan: [0 0 1 0], green: [0 0 0 1]), showing similar response from all devices, as well as device variations. (b) Images of the 10 digits used in this test. (c) Experimentally measured reservoir states after the memristors are subjected to the 10 inputs. The reservoir state is reflected as the read currents of the 5 memristors forming the reservoir.

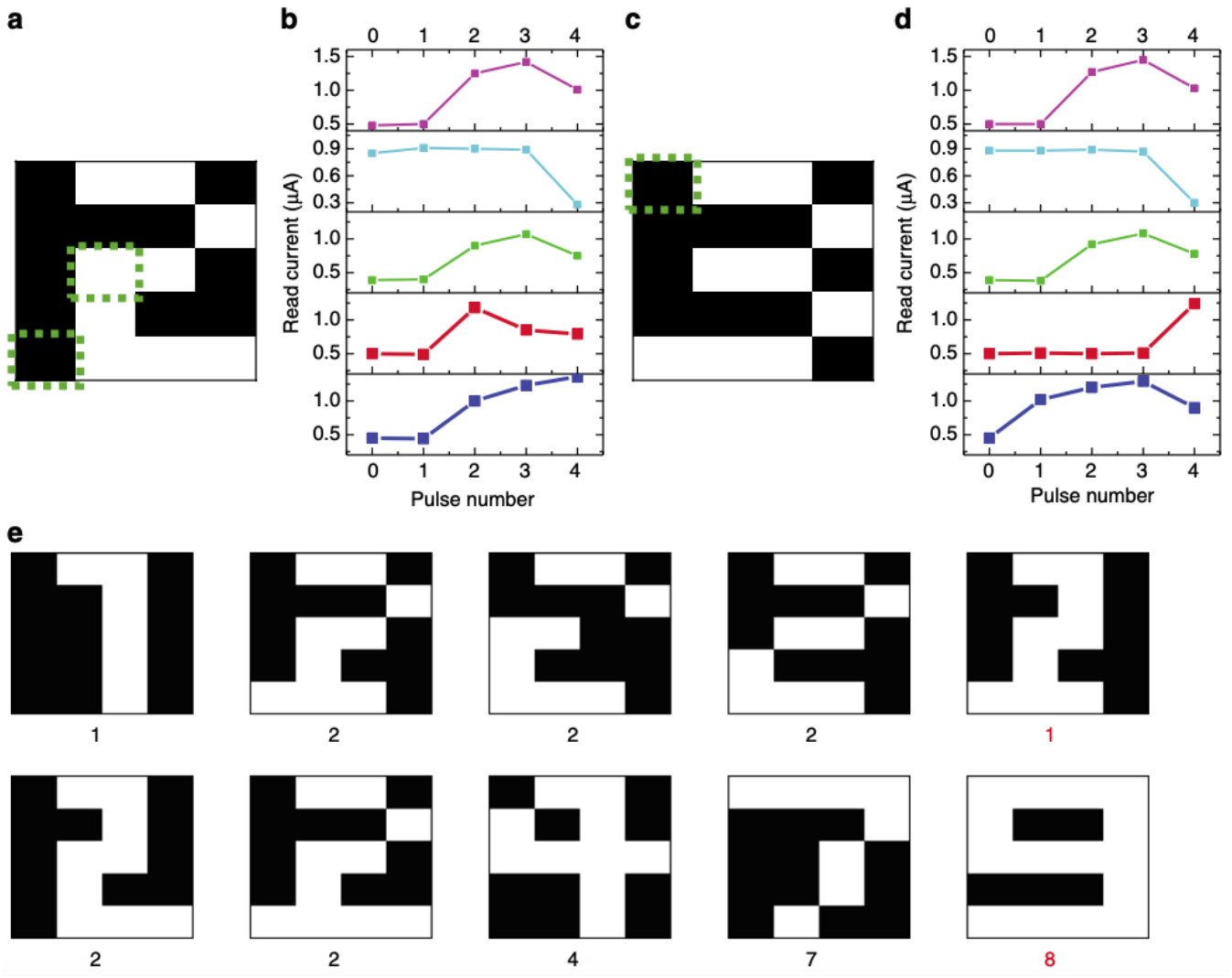

Figure 5: Recognition of noisy images. (a, c) Distorted images of digits 2 and 3 are generated by adding noise to the original data at locations marked by the dashed squares. (b, d) Corresponding reservoir states for these two inputs, showing differences in the two digits can be clearly captured by the memristors corresponding to the last two rows. (e) Recognition results of noisy digits. The RC system output, e.g., the predicted digit is shown below each case. The distorted images are generated by adding noise to the original training samples. The system can still successfully identify the majority of the distorted images without additional training, until too much noise is added as in the last two cases where the incorrect classifications are marked in red

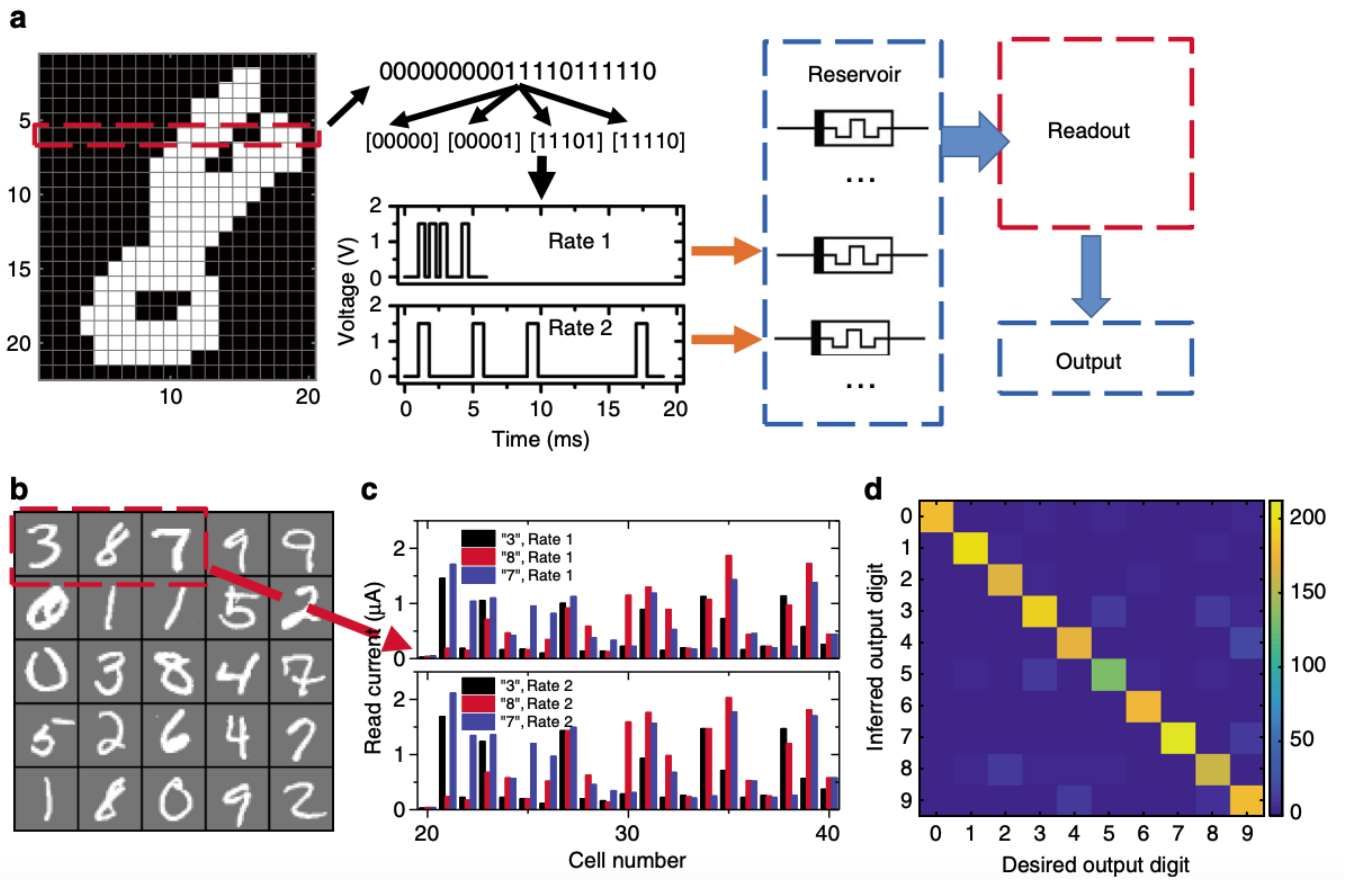

Figure 6: Handwritten digit recognition using a memristor-based RC system. (a) The process flow. The original digit image is first converted into pulse streams and fed to the memristor-based reservoir at different rates. The recognition result is generated after feeding the reservoir state to a trained readout function. (b) Some examples from the MNIST database. (c) Reservoir states corresponding to the three examples in (b) at two input rates (rate 1: timeframe width 0.8 ms, with pulse of 1.5 V, 0.5 ms; rate 2: timeframe width 4 ms, with pulses of 1.5 V, 0.8 ms), showing significant differences in the reservoir states. (d) False color confusion matrix showing the experimentally obtained classification results from the RC system vs. the desired outputs. The occurrence of the predicted output for each test case is represented by colors shown in the color scale. A recognition accuracy of 88.1% was obtained from the reservoir consisting of only 88 memristors.

3 Potential next steps in memristor-based reservoir computing

- Investigation of possible architectures for memristor-based reservoir computing systems, including hybrid approaches that combine memristors with other types of hardware.
- Development of training algorithms specifically designed for memristor-based reservoirs, taking into account the unique properties of memristors.
- Exploration of possibility of robust modelling of circuits with memristors within reservoir computing framework.

References

[1] Kang Jun Bai et al. "Design Strategies and Applications of Reservoir Computing: Recent Trends and Prospects [Feature]". In: *IEEE Circuits and Systems Magazine* 23.4 (2023), pp. 10–33. ISSN: 1558-0830. DOI: 10.1109/mcas.2023.3325496. URL: http://dx.doi.org/10.1109/MCAS.2023.3325496.

[2] Lucas Deckers et al. "Extended liquid state machines for speech recognition". In: *Frontiers in Neuroscience* Volume 16 - 2022 (2022). ISSN: 1662-453X. DOI: 10.3389/fnins.2022.1023470. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2022.1023470.

[3] Chao Du et al. "Reservoir computing using dynamic memristors for temporal information processing". In: *Nature Communications* 8.1 (Dec. 2017). ISSN: 2041-1723. DOI: 10.1038/s41467-017-02337-y. URL: http://dx.doi.org/10.1038/s41467-017-02337-y.

[4] Daniel J. Gauthier et al. "Next generation reservoir computing". In: *Nature Communications* 12.1 (Sept. 2021). ISSN: 2041-1723. DOI: 10.1038/s41467-021-25801-2. URL: http://dx.doi.org/10.1038/s41467-021-25801-2.

[5] Lyudmila Grigoryeva et al. "Time-Delay Reservoir Computers and High-Speed Information Processing Capacity". In: *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*. IEEE, Aug. 2016, pp. 492–495. DOI: 10.1109/cse-euc-dcabes.2016.230. URL: http://dx.doi.org/10.1109/CSE-EUC-DCABES.2016.230.

[6] Su In Hwang et al. "Multi-timescale temporal processing for ion-motion memristor-based physical reservoir computing". In: *Device* 3.9 (Sept. 2025), p. 100907. ISSN: 2666-9986. DOI: 10.1016/j.device.2025.100907. URL: http://dx.doi.org/10.1016/j.device.2025.100907.

[7] Wolfgang Maass. "Liquid State Machines: Motivation, Theory, and Applications". In: *Computability in Context*. IMPERIAL COLLEGE PRESS, Feb. 2011, pp. 275–296. ISBN: 9781848162778. DOI: 10.1142/9781848162778_0008. URL: http://dx.doi.org/10.1142/9781848162778_0008.

[8] S. Ortín et al. "A Unified Framework for Reservoir Computing and Extreme Learning Machines based on a Single Time-delayed Neuron". In: *Scientific Reports* 5.1 (Oct. 2015). ISSN: 2045-2322. DOI: 10.1038/srep14945. URL: http://dx.doi.org/10.1038/srep14945.

[9] Ulrich Parlitz. "Learning from the past: reservoir computing using delayed variables". In: *Frontiers in Applied Mathematics and Statistics* 10 (Mar. 2024). ISSN: 2297-4687. DOI: 10.3389/fams.2024.1221051. URL: http://dx.doi.org/10.3389/fams.2024.1221051.