Reservoir computing

Karol Bednarz

January 5, 2026

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:

$$h_t = f\big((1-k) \cdot u_t \cdot W_{in} + k \cdot h_{t-1} \cdot W_h + y_{t-1} \cdot W_{1b} + b\big) \tag{1}$$

Where:

h_{t-1} – are the reservoir state respectively, from the previous time step,
u_t – is the observed data at time step t,
y_{t-1} – is the the predicted output state $t-1$,
$W_{in} \in \mathbb{R}^{N_u \times N_h}$ – is the input weight matrix,
$W_h \in \mathbb{R}^{N_h \times N_h}$ – is the internal weight matrix,
$W_{1b} \in \mathbb{R}^{N_h \times N_y}$ – is the output fedback weight matrix,
$b \in \mathbb{R}^{N_h}$ – 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:

$$y_t = h_t \cdot W_{\text{out}} \tag{2}$$

Where:

$W_{\text{out}} \in \mathbb{R}^{N_h \times N_y}$ – 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:

$$W_h' = \alpha \odot W_h \tag{3}$$
$$W_h^{\dagger} = \frac{\rho W_h}{|\lambda_{max}(W_h)|} \tag{4}$$

Where:

$\rho \in (0,1)$ – is the spectral radius, typically $\rho \in [0.9, 1]$
$\lambda_{max}(W_h)$ – 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 input—the 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

$$MC = \sum_{k=1}^{\infty} MC_k = \sum_{k=1}^{\infty} d^2(u_{t-k}, y_t) = \sum_{k=1}^{\infty} \frac{\text{cov}^2(u_{t-k}, y_t)}{\sigma^2(u_t)\sigma^2(y_t)} \tag{5}$$

Where:

$d^2(u_{t-k}, y_t)$ – is the square of the correlation coefficient between the output y_t and the input u_{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_{1b} are typically initialized randomly and remain fixed during training. The training process can be summarized in the following steps:

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

Where:

$Y \in \mathbb{R}^{T \times N_y}$ – is the matrix of target outputs for all time steps,
$H \in \mathbb{R}^{T \times N_h}$ – 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:

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

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 $N_h \times N_h$.

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

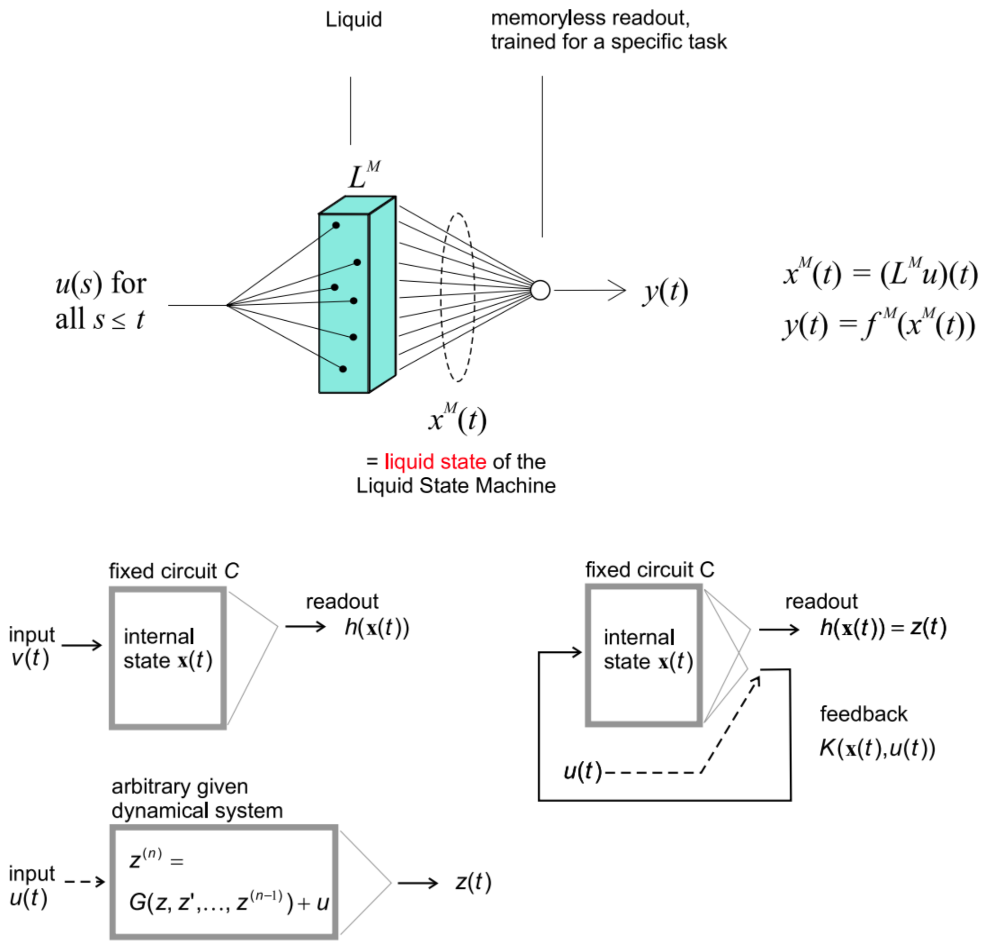
Based on [5, 9, 8].

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

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

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,



Liquid
memoryless readout,
trained for a specific task
L^M
$u(s)$ for all $s \le t$
$y(t)$
$x^M(t) = (L^M u)(t)$
$y(t) = f^M(x^M(t))$
$x^M(t)$
= liquid state of the Liquid State Machine
fixed circuit C
input $v(t)$
internal state $\mathbf{x}(t)$
readout $h(\mathbf{x}(t))$
fixed circuit C
internal state $\mathbf{x}(t)$
readout $h(\mathbf{x}(t)) = z(t)$
feedback $K(\mathbf{x}(t), u(t))$
$u(t)$
arbitrary given dynamical system
input $u(t)$
$z^{(n)} = G(z, z', \ldots, z^{(n-1)}) + u$
$z(t)$

Figure 1: Liquid State Machine architecture

$\mathbf{W}_{in} \in \mathbb{R}^{N \times d}$	– 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 $[-1, 1]$,
$\mathbf{W}_{out} \in \mathbb{R}^{p \times N}$	– is the output weight matrix,
$\gamma, \beta \in \mathbb{R}^{+}$	– 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].

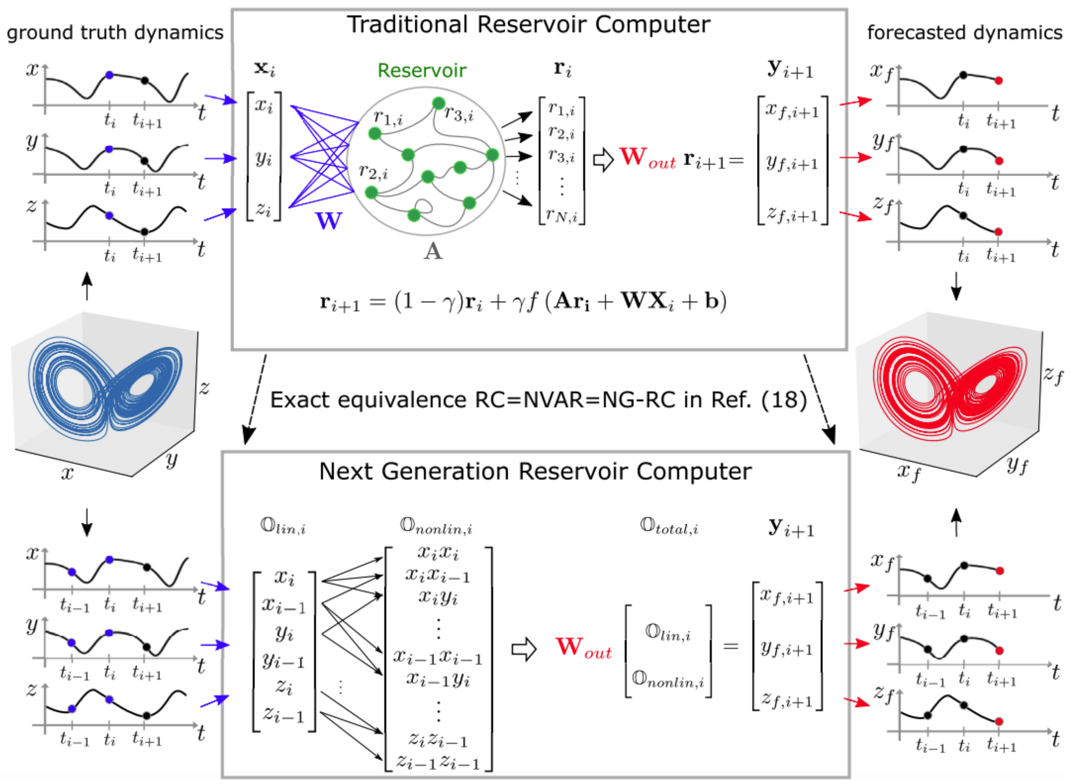

ground truth dynamics
Traditional Reservoir Computer
forecasted dynamics
Reservoir
$\mathbf{W}$
$\mathbf{A}$
$\mathbf{W}_{out}\,\mathbf{r}_{i+1}=$
$\mathbf{r}_{i+1} = (1-\gamma)\mathbf{r}_i + \gamma f\left(\mathbf{A}\mathbf{r_i} + \mathbf{W}\mathbf{X}_i + \mathbf{b}\right)$
Exact equivalence RC=NVAR=NG-RC in Ref. (18)
Next Generation Reservoir Computer
$\mathbb{O}_{lin,i}$
$\mathbb{O}_{nonlin,i}$
$\mathbb{O}_{total,i}$
$\mathbf{y}_{i+1}$
$\mathbf{W}_{out}$

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:

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

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

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

Where:

$\mathbf{O}_{total,i+1} \in \mathbb{R}^{M\times 1}$ – is the feature vector at time step $i+1$,
$\mathbf{W}_{out} \in \mathbb{R}^{p\times M}$ – 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 $\mathbf{O}_{total}$ and we match $\mathbf{Y}$ to the desired output $\mathbf{Y}_d$ in a least-square sense using Tikhonov regularization so that $\mathbf{W}_{out}$ is given by

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

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 $\mathbf{O}_{total}$ 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 $\mathbf{r} \odot \mathbf{r} = [r_1^2, r_2^2, \ldots, r_N^2]^T$

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

Where:

$\oplus$ – is the concatenation operator.

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

The linear features $\mathbf{O}_{linear,i}$ 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 $\mathbf{X}_i = [x_{1,i}, x_{2,i}, \ldots, x_{d,i}]^T$ is a d-dimensional vector, $\mathbf{O}_{linear,i}$ has $d \cdot k$ components, and is given by

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

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 $\mathbf{O}_{nonlinear}$ of the feature vector is a nonlinear function of $\mathbf{O}_{linear}$. 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 $\mathbf{O}_{linear} \otimes \mathbf{O}_{linear}$, which is a symmetric matrix with $(dk)^2$ elements. A quadratic nonlinear feature vector $\mathbf{O}^{(2)}_{nonlinear}$, for example, is composed of the $(dk)(dk+1)/2$ unique monomials of $\mathbf{O}_{linear} \otimes \mathbf{O}_{linear}$, 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

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

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

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

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

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

2 Memristors in reservoir computing

2.1 Reservoir computing using dynamic memristors for temporal information processing

Based on [3, 6].

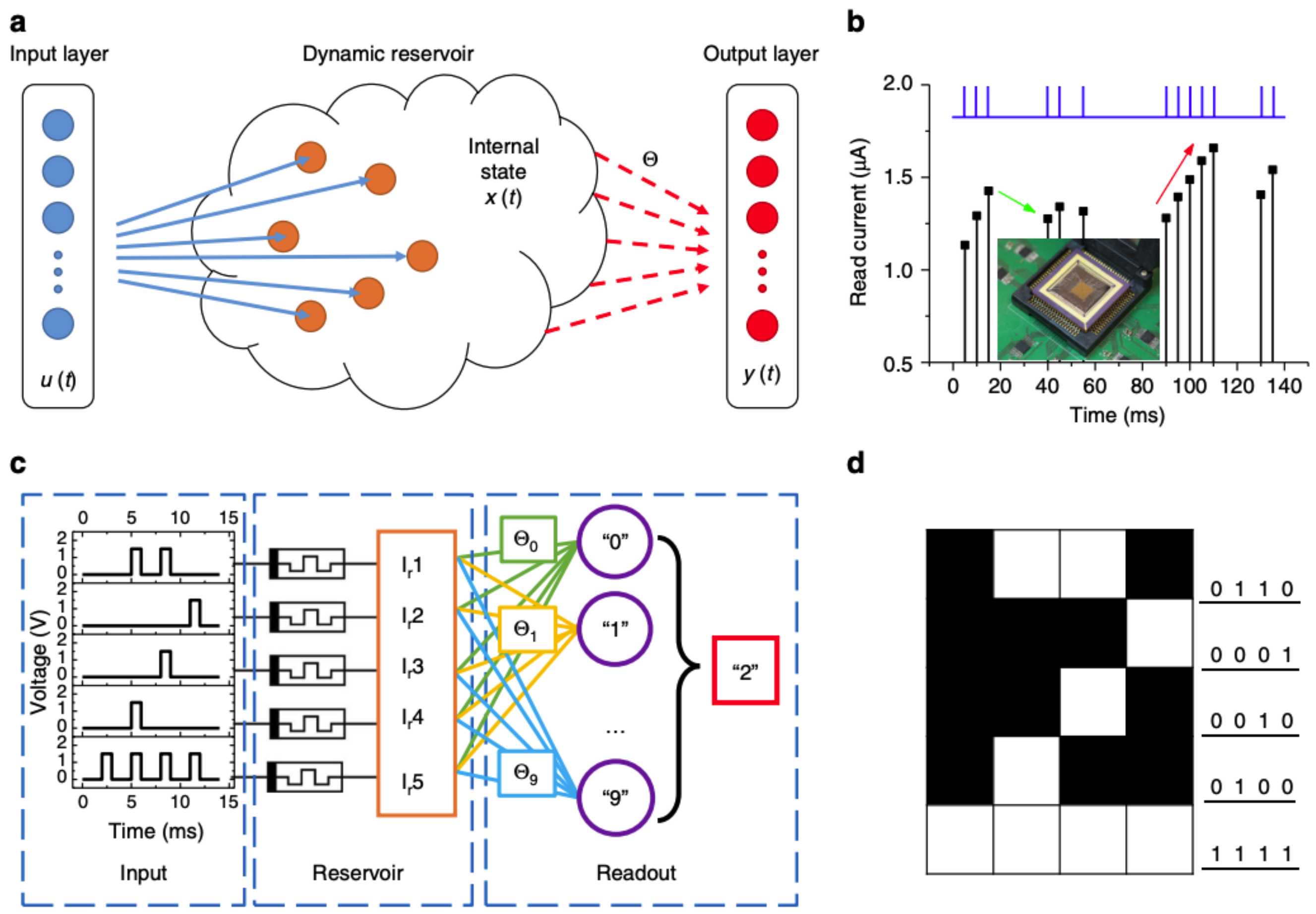


Figure 3: Reservoir computing system based on a memristor array. (a) Schematic of an RC system, showing the reservoir with internal dynamics and a readout function. Only the weight matrix Θ connecting the reservoir state x(t) and the output y(t) needs to be trained. (b) Response of a typical WOx memristor 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.

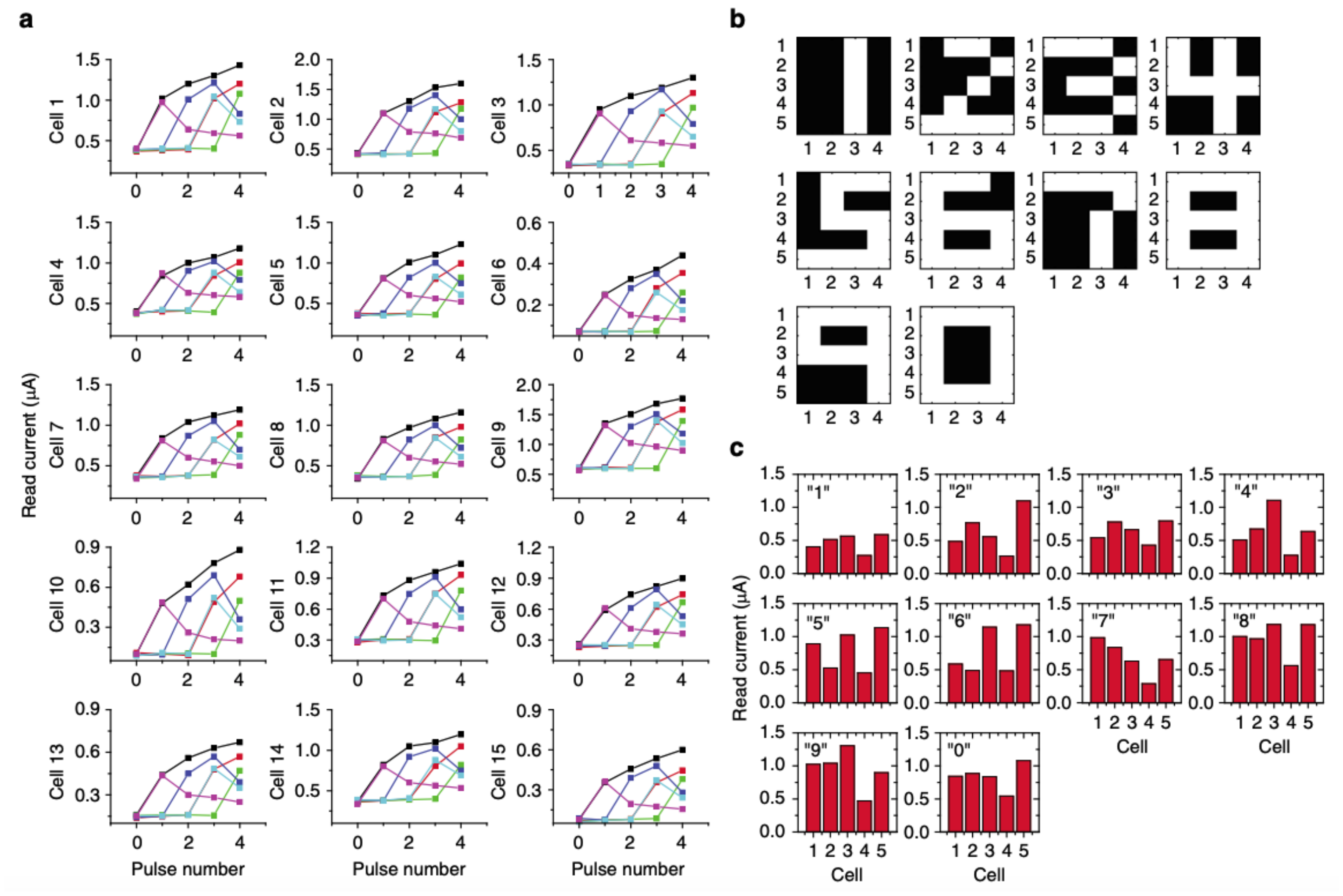


Figure 4: Reservoir states used to differentiate different temporal inputs. (a) The response of 15 memristors from the array to 6 different pulse streams (black: [1 1 1 1], purple: [1 0 0 0], blue: [0 1 1 0], red: [0 0 1 1], cyan: [0 0 1 0], green: [0 0 0 1]), showing similar response from all devices, 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.

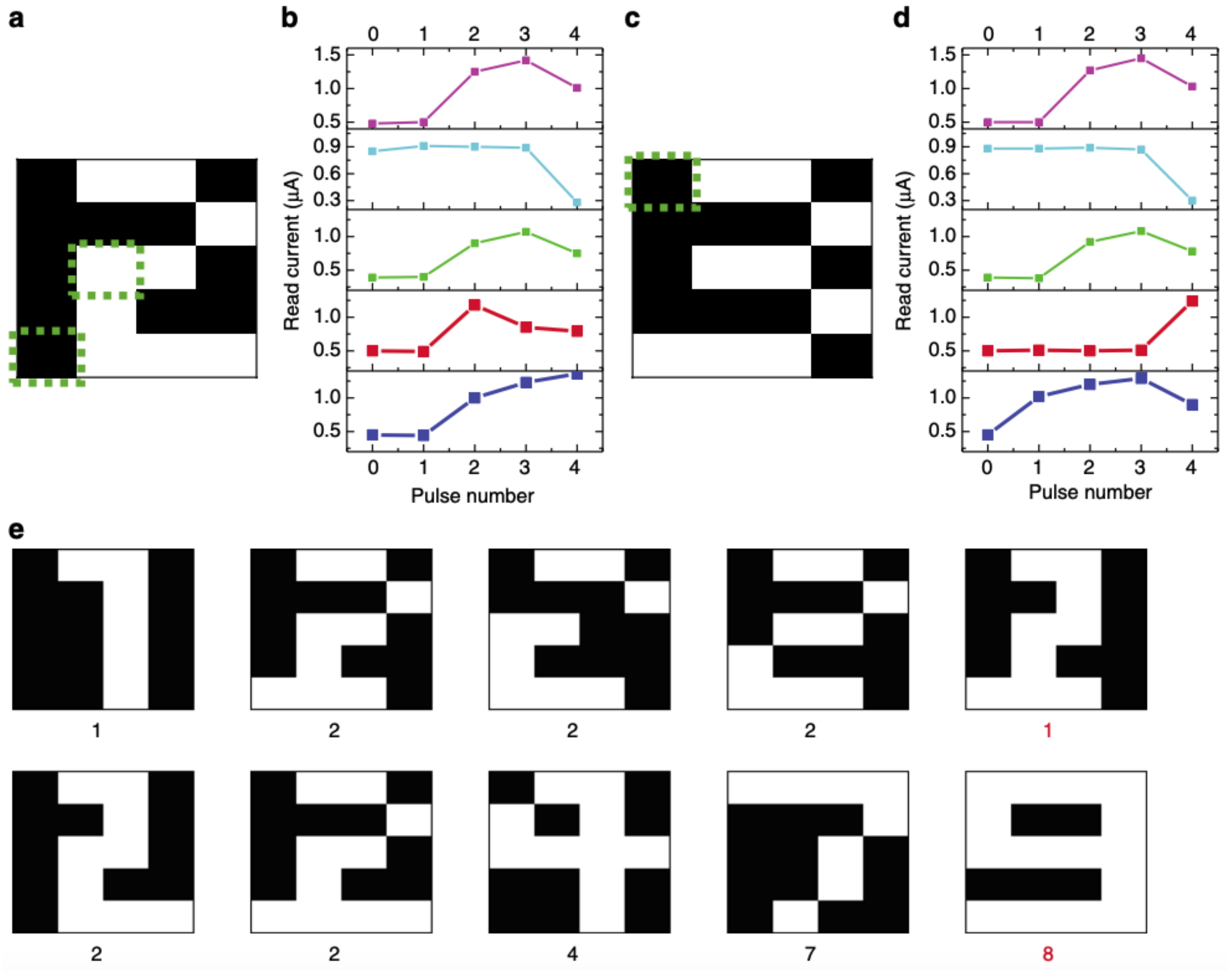


Figure 5: Recognition of noisy images. (a, c) Distorted images of digits 2 and 3 are generated by adding noise to the original data at locations marked by the dashed squares. (b, d) Corresponding reservoir states for these two inputs, showing differences in the two digits can be clearly captured by the 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

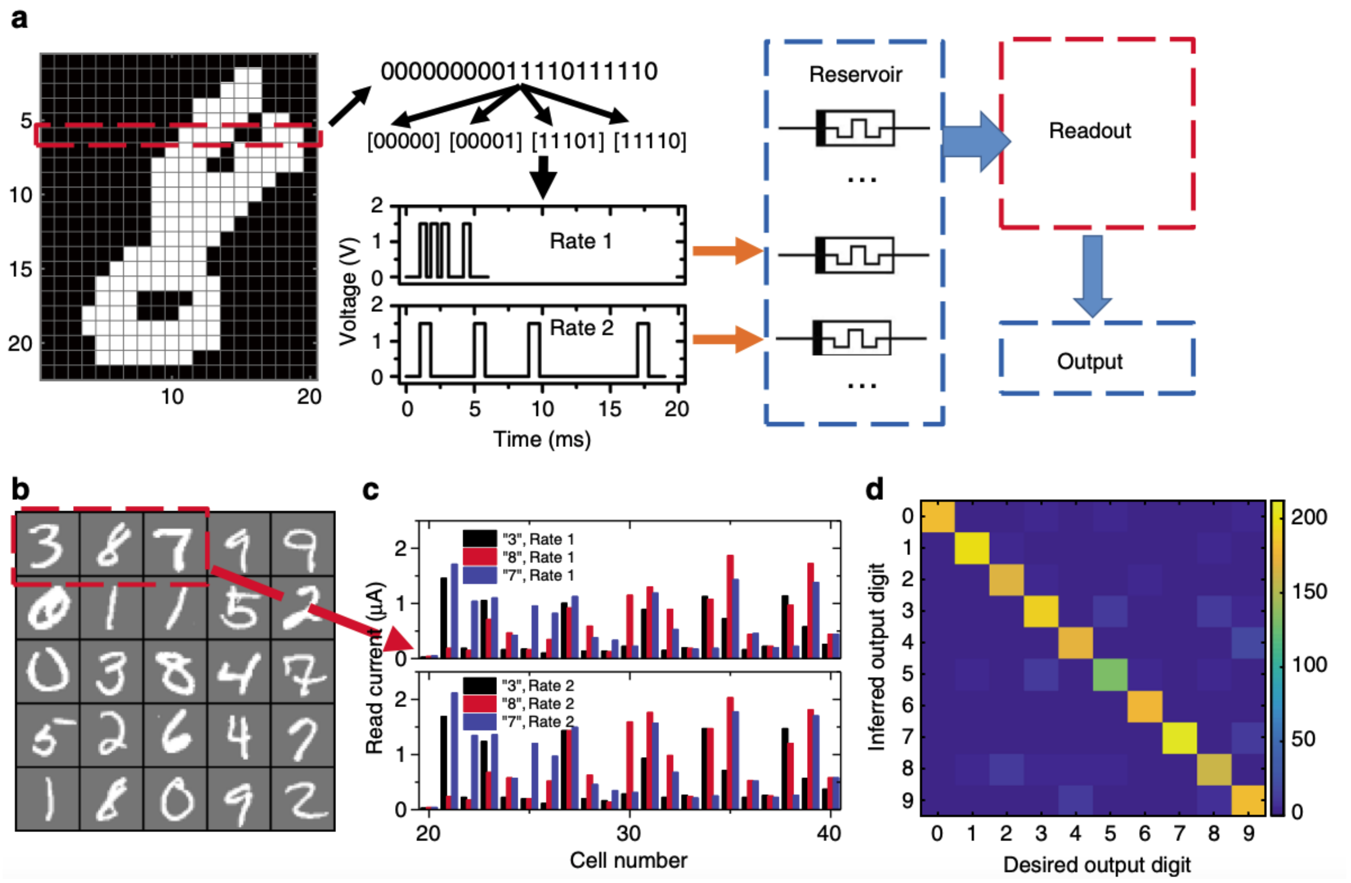


Figure 6: Handwritten digit recognition using a memristor-based RC system. (a) The process flow. The original digit image is first converted into pulse streams and fed to the memristor-based reservoir at different rates. The recognition result is generated after feeding the reservoir state to a trained 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.

4 Simulation results

The experimental investigation commenced with comprehensive computational simulations employing the memristor MMS model.

For the evaluation of the proposed reservoir computing architecture, the standardized MNIST (Modified National Institute of Standards and Technology) dataset was utilized, comprising grayscale images of handwritten digits with dimensions of 28×28 pixels. This benchmark dataset represents a fundamental test case for pattern recognition systems and enables direct comparison with existing neuromorphic computing paradigms.

4.1 Row-Wise Input Encoding Methodology

The initial phase of the simulation protocol involved a row-wise decomposition strategy, described in [3] for processing the input images. Specifically, each 28×28 pixel image was segmented into 28 discrete rows, with each row being processed sequentially through the reservoir architecture. This decomposition approach facilitated the implementation of a reservoir configuration comprising 28 memristive elements, where each memristor corresponded to a distinct spatial position within the input row vector.

The encoding of pixel information into the temporal domain was accomplished through a voltage-based pulse amplitude modulation scheme. Each grayscale pixel value, normalized within the range [0, 1], was mapped to a corresponding voltage amplitude in the range [0, $\mathrm{V_{max}}$] volts, where $\mathrm{V_{max}}$ represents the maximum applied voltage determined by the operational constraints of the memristive devices. This linear mapping function can be expressed as:

$$V_{\text{pixel}}(i,j) = V_{\max} \cdot \frac{I(i,j)}{255} \tag{18}$$

where $I(i,j)$ denotes the grayscale intensity value at pixel position (i,j), and $V_{\text{pixel}}(i,j)$ represents the corresponding encoded voltage amplitude.

4.1.1 Results

For the provided row-wise input encoding methodology, the memristor-based reservoir computing system achieved a classification accuracy of 60.37% on the MNIST dataset. This performance metric underscores the low efficacy of the proposed architecture in leveraging the intrinsic dynamics of memristive devices for temporal information processing tasks.

4.2 Patch-Based Feature Extraction and Memristor Reservoir Processing

Building upon the row-wise processing paradigm, a more sophisticated approach was developed employing a patch-based feature extraction methodology combined with memristor reservoir computing. This technique leverages local spatial correlations within the input images while maintaining computational efficiency through systematic spatial subsampling.

4.2.1 Spatial Patch Extraction Algorithm

The foundation of this approach rests upon a sliding window mechanism for extracting overlapping or non-overlapping rectangular patches from the input images. The patch extraction function implements a systematic traversal of the image space according to the following algorithmic procedure:

Given an input image $\mathbf{I} \in \mathbb{R}^{H \times W}$ where H and W denote the height and width respectively, the extraction process generates a set of patches $\mathcal{P} = \{P_1, P_2, ..., P_N\}$ where each patch $P_i \in \mathbb{R}^{k_h \times k_w}$

represents a local spatial region of dimensions $k_h \times k_w$ (the kernel size). The extraction follows a raster scan pattern with a specified stride parameter s that controls the spatial sampling density.

The mathematical formulation of the patch extraction can be expressed as:

$$P_{m,n} = \mathbf{I}[is : (is + k_h - 1), js : (js + k_w - 1)] \tag{19}$$

where the indices (m, n) correspond to the patch coordinates in the output grid, and:

$$i = 1 + m \cdot s, \quad m \in \{0, 1, ..., \lfloor \frac{H - k_h}{s} \rfloor\} \tag{20}$$

$$j = 1 + n \cdot s, \quad n \in \{0, 1, ..., \lfloor \frac{W - k_w}{s} \rfloor\} \tag{21}$$

Each extracted patch is subsequently vectorized through a flattening operation, transforming the 2D patch into a 1D feature vector of length $k_h \cdot k_w$:

$$\vec{p}_i = \text{vec}(P_i) \in \mathbb{R}^{k_h \cdot k_w} \tag{22}$$

4.2.2 Implementation Parameters for MNIST Processing

For the specific case of MNIST digit classification, the following hyperparameters were empirically determined to optimize the balance between feature resolution and computational efficiency:

- **Kernel dimensions**: $(k_h, k_w) = (5, 5)$, yielding 25-dimensional feature vectors per patch
- **Stride**: $s = 3$ pixels, providing moderate spatial overlap between adjacent patches
- **Image dimensions**: $(H, W) = (28, 28)$ pixels (standard MNIST format)

These parameters result in a systematic decomposition of each image into a grid of patches with dimensions:

$$n^h_{\text{patches}} = \lfloor \frac{28 - 5}{3} \rfloor + 1 = 8 \tag{23}$$

$$n^w_{\text{patches}} = \lfloor \frac{28 - 5}{3} \rfloor + 1 = 8 \tag{24}$$

$$n^{\text{total}}_{\text{patches}} = n^h_{\text{patches}} \times n^w_{\text{patches}} = 64 \tag{25}$$

This configuration effectively transforms each 784-dimensional input image (28×28 pixels) into 64 localized feature patches, each containing 25 pixels of spatial information.

4.2.3 Temporal Encoding via Carrier Wave Modulation

The proposed methodology involves the encoding of patch information through amplitude-modulated carrier waves. For each patch vector $\vec{p}_i$, a time-varying voltage signal is generated using the following carrier wave generation scheme:

$$V_{\text{carrier}}(t) = A(t) \cdot \Pi \left(\frac{t \bmod T_{\text{period}}}{D \cdot T_{\text{period}}} \right) \tag{26}$$

where:

- $A(t)$ represents the amplitude modulation function derived from the patch pixel values

- $\Pi(\cdot)$ denotes the rectangular pulse function
- $T_{\text{period}} = 10^{-5}$ seconds (10 microseconds) is the carrier period
- $D = 0.8$ is the duty cycle, resulting in 8 microseconds on-time and 2 microseconds off-time
- The sampling frequency $f_s = 10^7$ Hz (10 MHz) ensures adequate temporal resolution

The amplitude modulation encodes each pixel value within the patch as a discrete voltage level, creating a multi-level pulse train that preserves the spatial information in the temporal domain. This encoding is particularly well-suited for memristor dynamics, as it exploits the devices' inherent integration capabilities.

4.2.4 Feature Vector Construction and Augmentation

The final feature representation for each image is constructed by aggregating the terminal states of all memristors after processing their respective patches:

$$\mathbf{x}_j = [x_1^{\text{final}}, x_2^{\text{final}}, ..., x_{64}^{\text{final}}, 1]^T \in \mathbb{R}^{65} \tag{27}$$

where $x_i^{\text{final}} = x_i(t_{\max})$ represents the final state of the memristor processing patch i, and the additional unity element serves as a bias term for subsequent linear classification layers.

This feature extraction process is applied systematically to the entire training dataset, resulting in a transformed feature matrix:

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N] \in \mathbb{R}^{65\times N} \tag{28}$$

where N denotes the number of training samples.

4.2.5 Computational Complexity and Parallelization Potential

The patch-based approach offers several computational advantages over full-image processing:

1. **Reduced dimensionality**: Each memristor processes only 25 pixels instead of 784, significantly reducing the temporal duration of input signals.
2. **Natural parallelism**: The 64 patches can be processed independently, enabling straightforward parallelization across multiple memristor units or computational cores.
3. **Memory efficiency**: The patch-wise processing requires storage of only $(n_{\text{patches}} + 1) \times N$ floating-point values for the feature matrix, compared to the original $784 \times N$ pixel values.

The total computational time for processing a single image can be estimated as:

$$T_{\text{total}} = n_{\text{patches}} \times T_{\text{patch}} + T_{\text{overhead}} \tag{29}$$

where T_{patch} represents the time required for a single patch-memristor interaction, and T_{overhead} accounts for data preparation and feature aggregation operations.

4.3 Patch Encoding for Memristor Crosbar Array Reservoir

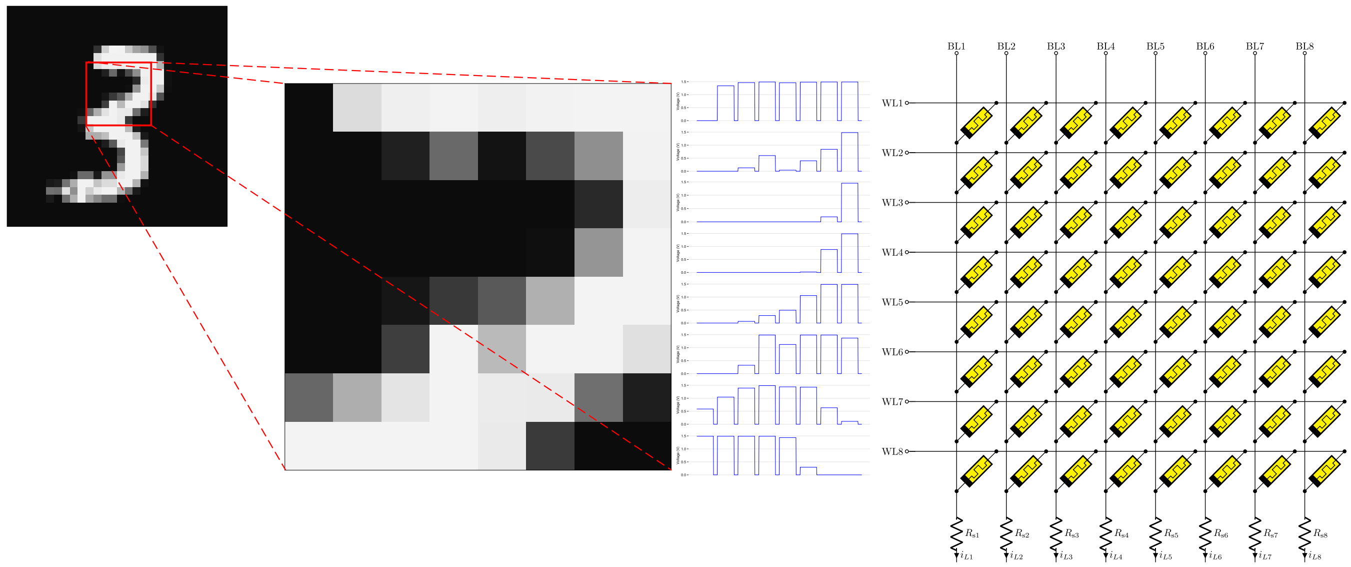


Figure 7: Patch-based feature extraction and memristor reservoir processing methodology for the memristor crossbar array reservoir computing system.

4.4 Results

For the provided patch-based feature extraction and memristor reservoir processing methodology, the memristor-based reservoir computing system achieved a classification accuracy of 80.25% on the MNIST dataset (91.33% for tetst datasets for the crossbar array) . This performance metric underscores higher efficacy of the proposed architecture in leveraging localized spatial features and the intrinsic dynamics of memristive devices for temporal information processing tasks.

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.