Novel Memristive Reservoir Computing with Evolvable Topology for Time Series Prediction

Xinming Shi[1][0000−0002−2053−6924], Leandro L. Minku[2][0000−0002−2639−0671], and Xin Yao[2,3][0000−0001−8837−4442]

[1] School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, UK
[2] School of Computer Science, University of Birmingham, Birmingham, UK
[3] School of Data Science, Lingnan University, Hong Kong SAR
x.shi@qub.ac.uk, l.l.minku@bham.ac.uk, xinyao@ln.edu.hk

Abstract. This study introduces a novel reservoir computing framework featuring an evolvable topology, optimized for minimal clustering degree and path length, which are key characteristics identified as beneficial for reservoir performance. We implement this framework in memristive circuits, enabling dynamic on-chip adaptation and evolution of the topology. We evaluate the efficacy of our memristive reservoir in a wave generation task and two time series prediction tasks. Experimental results demonstrate that our approach not only outperforms existing state-of-the-art methods in predictive performance but also reduces the required circuit area compared to other hardware-based reservoir implementations. This enhancement in both efficiency and performance illustrates the potential of our approach for advancing neuromorphic computing applications.

Keywords: Evolvable hardware · Neuromorphic computing · Reservior computing · Memristors · Time series.

1 Introduction

Reservoir Computing (RC) [13] is a branch of recurrent neural networks designed to address the computational and training limitations of traditional RNNs. It utilizes a dynamic system called the reservoir for nonlinear processing of input signals, projecting them into a higher-dimensional space. This nonlinear transformation allows temporal features to be mapped onto reservoir states, which are then processed by a trained linear readout layer. The reservoir's key characteristics include dependency on historical data to capture temporal relationships and a fading memory property, ensuring state dependency on recent past inputs [3]. RC has been implemented in various forms, including electronic, photonic, and atomic switch-based systems [35, 28, 31, 26].

In the context of software RC, efforts have been directed towards enhancing RC applications and refining existing methodologies [15]. Meanwhile, the hardware realm, especially with memristor-based RC, has attracted a lot of interest

due to the inherent benefits of memristors such as their non-volatility, energy efficiency, and high integration density [38, 19, 25]. These attributes make memristors suitable for implementing RC systems, offering a fresh perspective on reservoir design, especially considering their nonlinear dynamics and memory capabilities.

Various RC algorithms have been realized in hardware, prominently featuring Echo State Networks (ESN) [14] and Liquid State Machines (LSM) [22], which are grounded in nonlinear function neurons and spiking neurons, respectively. Specifically, FPGA-based implementations of ESN models, such as the 64-neuron model by Yi et al. [35] and the model for chaotic time series forecasting by Alomar et al. [1], have demonstrated their efficacy in data recognition and classification tasks. Likewise, LSM models have been successfully embedded in FPGAs and VLSIs, with notable implementations including a 135-neuron FPGA model for pattern recognition, achieving a high accuracy of 96.4% on the TI46 speech corpus [32], and larger LSM models with 200 [24] and 324 neurons [37] for speech recognition.

Beyond neuron-based RC models, dynamic system-based RC models, particularly photonic reservoirs, have garnered interest for their high-speed processing capabilities [2]. However, photonic reservoirs often require costly peripherals, such as digitizers and waveform generators [4]. As a cost-effective alternative, electronic reservoirs utilizing traditional Complementary Metal Oxide Semiconductor (CMOS) components, capacitors, and operational amplifiers have been developed [2–4]. Memristive reservoirs present a compact and energy-efficient alternative to CMOS systems, leveraging the nano-scale, low-power characteristics of memristors [30, 31]. However, the static architectures of current memristor-based reservoir computing (RC) systems limit their adaptability to varying computational demands and operational changes [29, 16]. This underscores the need for innovative, dynamically reconfigurable memristive RC frameworks that can efficiently handle diverse tasks and adapt to changing complexities, enhancing their versatility and functionality.

Despite advances, the limitations of random reservoir generation have led to the development of task-specific designs that often yield better outcomes [18]. This shift has fueled interest in evolutionary reservoirs, where studies focus on optimizing configurations like node count, connectivity, and weight parameters to enhance training effectiveness [5, 18, 6]. Evolutionary algorithms, including genetic algorithms [36], Evolino [23], evolutionary strategy [33], and Particle Swarm Optimization (PSO) [6], have proven effective, optimizing global parameters and modifying network topology to improve reservoir performance [8].

Memristive reservoir computing provides an efficient method for adaptive temporal data processing. Challenges in system design include scalability and practical constraints. Previous efforts have primarily centered on software-based or static solutions, with minimal focus on hardware-based, evolvable alternatives. This paper introduces a novel on-chip framework utilizing memristors' nonlinear properties to dynamically evolve and optimize reservoir topology. This results in

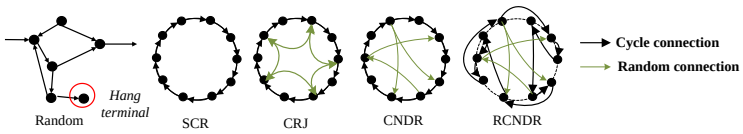

Fig. 1. Different Reservoir Topologies.

a reduced clustering degree and shorter average path lengths, enhancing compactness. The key contributions of our work are outlined as follows:

1. We propose a novel reservoir approach with an evolvable topology that can achieve small clustering degree and small average path length, which are desirable properties for reservoirs according to previous studies.
2. We implement our proposed reservoir approach into the memristive circuits, enabling the on-chip evolution and dynamic adaptation of the reservoir topology.
3. We evaluate the performance of the memristive reservoir circuit with an evolvable topology on one wave generation task and two time series prediction tasks. Our proposed approach outperforms the state-of-the-art methods in terms of the predictive performance.
4. We compare our proposed memristive reservoir featuring an evolvable topology with other hardware implementations of reservoirs. Our approach not only achieves superior predictive performance but also requires a smaller circuit area, distinguishing it from other implementations.

The rest of this paper is structured as follows. Section 2 proposes novel reservoir computing system with an evolvable topology. Section 3 introduces our proposed evolvable memristive reservoir circuits . Section 4 describes the experiments for verifying our proposed novel reservoir computing system with an evolvable topology. The conclusions of our work are presented in Section 5.

2 Novel Reservoir Computing System with Evolvable Topology

In Section 2.1, we propose novel reservoir computing system with an evolvable topology, including Cycle Random Reservoir (CNR) and the Random Cycle with Random Direct Connections Reservoir (RCNDR). In Section 2.2 we further compare different topologies of reservoir computing.

2.1 Cycle Random Reservoir and Random Cycle with Random Direct Connections Reservoir

Based on the premise that reservoirs should ideally have a small clustering degree for sparse connectivity and a small average path length to avoid overly cluttered dynamic information flow through the nodes, as suggested by Jaeger et. al [15], we propose the Cycle Random Reservoir (CNR) and the Random Cycle with Random Direct Connections Reservoir (RCNDR) topologies.

Cycle Random Reservoir (CNR) The CNR topology is motivated by the need for a structured yet dynamic approach to information processing in reservoirs. It combines a regular cycle, which ensures a predictable and stable foundation for signal propagation, with additional random connections that introduce variability and increase the dynamical richness of the reservoir. This setup maintains a low clustering degree, as most nodes only connect to their immediate neighbors in the cycle, with random connections preventing the formation of densely interconnected clusters. Furthermore, the CNR topology ensures a small average path length due to the cycle, but the random connections allow for longer individual paths when needed. This design effectively balances the representation of different dynamical timescales, allowing the reservoir to process a wide range of patterns and signals with varying temporal characteristics.

In the CNR, the adjacency matrix W consists of two parts: the cyclic component W_{cycle} and the random component W_{random}. The CNR adjacency matrix can be formulated as:

$$W_{\text{cycle}}(i,j) = \begin{cases} 1 & \text{if } j = (i \mod N) + 1, \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$W_{\text{random}}(i,j) = \begin{cases} 1 & \text{with probability } p, \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The overall adjacency matrix W for CNR is then given by:

$$W = W_{\text{cycle}} + W_{\text{random}}. \tag{3}$$

Random Cycle with Random Direct Connections Reservoir (RCNDR) Building upon the CNR, the RCNDR topology further refines the reservoir's capacity to handle complex temporal patterns by introducing non-random direct connections. These direct connections act as shortcuts within the circular structure, reducing the average path length between distant nodes. This feature is particularly beneficial for capturing long-term dependencies in time series data or for tasks requiring the integration of information over extended periods. The direct connections are chosen strategically rather than randomly to optimize the reservoir's performance for specific tasks, allowing the RCNDR to maintain sparse connectivity while enhancing computational power and flexibility.

For the RCNDR, the adjacency matrix W also includes an additional component for the non-random direct connections W_{direct}:

$$W_{\text{direct}}(i,j) = \begin{cases} 1 & \text{if a direct connection} \\ & \text{exists between neurons } i \text{ and } j, \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Thus, the overall adjacency matrix W for RCNDR is given by:

$$W = W_{\text{cycle}} + W_{\text{random}} + W_{\text{direct}}. \tag{5}$$

Table 1. Comparison of different reservoir topologies based on their degrees of local clustering and average path length.

Topology	**Random** [13]	**CRJ**[21]	**SCR** [20]	**CNDR**	**RCNDR**
Degree of local clustering	Small	High	Small	High	Small
Average path length	Small	Small	High	Small	Small
Circuit vality	NO	Yes	Yes	Yes	Yes

Each element of W_{random} and W_{direct} may be scaled or drawn from a distribution to encode specific dynamics into the reservoir.

2.2 Reservoir Topology Comparison

According to work [15], the local clustering coefficient is a measure of the degree of interconnection between nodes in a reservoir. A low local clustering coefficient means that the nodes in the reservoir are sparser, which may be beneficial to avoid the information flow being too cluttered in the reservoir. The average path length is a measure of the average distance between any two nodes in the reservoir. A small average path length means that information propagation in the reservoir is faster and more efficient, since only a few steps are required to reach any two nodes. However, a large average path length means that information propagation in the reservoir is slower and less efficient, as many steps are required to reach any two nodes. In addition, the average path length is also related to the dynamic time scale in the reservoir, that is, how long the nodes in the reservoir can maintain memory. A small average path length, while having longer individual paths, enables the reservoir to represent multiple dynamic time scales, thereby improving the reservoir's memory capabilities.

Fig. 1 visualizes different reservoir topologies of reservoir. Table 1 presents comparison of reservoir topologies, which reveals the network characteristics. The random topology exhibits both a low local clustering coefficient and a small average path length, forming an efficient yet sparse network that excels in information distribution. However, due to its random topology, there will be the hang terminals in their circuit implementations, which lacks of the circuit validity.

3 Evolvable Memristive Reservoir Circuit

In this work, we apply a memristor model that is similar with the one applied in work [29], which has non-volatile memory that regulates the flow of electrical current in a circuit and can remember the amount of charge that has previously flowed through it. This memristive behavior is determined by the memristance of the device, which changes according to the history of applied voltage and current.

Table 2 outlines the essential parameters defining the SPICE model of a memristor, a component known for its ability to retain a state of resistance based on the history of voltage and current passed through it. The low and high resistance states, denoted by R_{on} and R_{off}, respectively, establish the operational bounds

Table 2. Parameters of the memristor model and their values used in the experiments.

Parameter	Description	Value
R_{on}	Low resistance state	100Ω
R_{off}	High resistance state	$10k\Omega$
x_{ini}	Initial memristance state	'ra' (variable)
u_v	Ion mobility parameter	1×10^{-14}/stime
p	Power coefficient in window function	1
D	Thickness of the memristive material	10 nm

of the memristor, allowing it to switch between conductive and non-conductive modes. The initial state x_{ini} sets the starting resistance level, pivotal for the device's memory aspect. The ion mobility parameter uv is critical for dictating the rate at which the resistance state can change, reflective of the physical properties of the memristive material. The power coefficient p influences the nonlinearity of the memristance change, while the thickness D of the memristive material directly affects the device's scaling and resistive properties. Together, these parameters are integral to modeling the dynamic behavior of memristors in computational circuits, capturing their unique capability to integrate memory and processing in a single element.

3.1 Memristive Reservoir Model

We apply the schematic of a memristive network introduced in Tanaka et al. [29] as our basic memristive RC frameowrk. The network receives an input series x, which undergoes a masking operation that serves dual purposes. Primarily, this masking distributes the singular time series data across the entire neural array, essentially performing a dimensional expansion of the input. Secondly, it normalizes the input data by ensuring that time series with a non-zero mean are transformed to have a zero mean, a feature that proves beneficial in simplifying the ridge regression model by obviating the need for an intercept. In the context of physical reservoir computing, as introduced in prior studies [7, 29], the role of the input mask layer is realized through pre-processing and is not subjected to training. This approach is retained in our proposed model where the input pre-processing remains fixed throughout the computation process.

The input to the reservoir $\mathbf{x}(t) \in \mathbb{R}^{1\times n}$ is accompanied by a teaching signal $\mathbf{y}(t) \in \mathbb{R}^{1\times n}$ for training. The reservoir state $\mathbf{h}(t) \in \mathbb{R}^{1\times N}$ is used to generate the output $\hat{\mathbf{y}} \in \mathbb{R}^{1\times n}$ via the output matrix $W_{\text{out}} \in \mathbb{R}^{N\times n}$, calculated as $\hat{\mathbf{y}} := \mathbf{h}(t) \cdot W_{\text{out}}$.

The goal of training is to adjust W_{out} to minimize the error between $\hat{\mathbf{y}}$ and $\mathbf{y}(t)$, with an L^2 regularization to mitigate overfitting, expressed as [20]:

$$W_{\text{out}} := \underset{W\in\mathbb{R}^{N\times n}}{\arg\min} \left(\sum_{t=1}^{T^*} ||\hat{\mathbf{y}}(t) - \mathbf{y}(t)||^2 + \lambda ||W||^2 \right), \tag{6}$$

where $\lambda \geq 0$ is a regularization hyperparameter. Using ridge regression, W_{out} is derived from:

$$W_{\text{out}} = (H^{\mathsf{T}} H + \lambda \mathbb{I}_N)^{-1} H^{\mathsf{T}} \mathbf{y}. \tag{7}$$

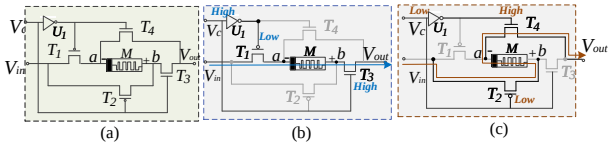

Fig. 2. Circuit Schematic of Programmable Memristive Unit (PMU) and its equivalent circuits.

In the equation, H represents the feature matrix used to compute output weights W_{out} in ridge regression. This approach uses the unique dynamics of memristors, which convert the input signal into a high-dimensional space. The outputs, processed by W_{out}, utilize the memristor currents, effectively merging memristive properties with linear readout for complex computations.

3.2 Memristor-based Reservoir Circuit

In this work, we employ the Programmable Memristive Unit (PMU) designed by Yang et al. [34] to develop an adaptable memristive network for Reservoir Computing (RC). The PMU's structure, which allows synchronized memristance adjustments, is ideal for creating a dynamic memristive reservoir. As depicted in Fig. 2, the PMU consists of four transistors—two PMOS (T_1 and T_2) and two NMOS (T_3 and T_4), coupled with a single memristor. The input (V_{in}) and control (V_c) signals, the latter switching between 'logic 0' (0V) and 'logic 1' (5V), regulate the memristor's current flow. With no input ($V_{in} = 0$), the memristor's state is preserved. Conversely, an active V_{in} combined with a high V_c enables current to flow from terminal a to b via T_1 and T_3, increasing memristance. A low V_c reverses this flow, reducing memristance as current moves from b to a.

By varying V_c's pulse widths and voltage states, we can tailor current flow and memristance levels. This capability allows us to connect multiple PMUs into complex networks with varied topologies, forming evolvable memristive reservoirs. These reservoirs are dynamically tunable for specific tasks, enhancing the flexibility and efficiency of the RC framework and opening new possibilities for advanced computational architectures.

3.3 Evolution of Memristive Reservoir Circuit

Circuit Representation In Section 3.1, we outline the reservoir computing system structure, comprising an input layer, a reservoir, and an output layer. The reservoir topology uses a binary adjacency matrix W_{bool}, with reservoir and output weights stored in W_{res} and W_{out}. Our algorithm evolves W_{bool} and W_{res}, while W_{out} is optimized offline using ridge regression (Equation (7)).

Our memristor-based model uses conductance changes in memristors to represent node states, adjusted through control voltage V_c pulse widths (Figure 2). Instead of directly evolving weights, we modify V_c pulse widths to control memristor currents and dynamically adjust system behavior.

Algorithm 1 Pseudo Code for Reservoir Evolution

Initialize parameters: crossover and mutation rates P_c, P_m, generation g
Define tournament size num_Tour, and reservoir size range num_ini_node to num_max_node
Initialize population $\boldsymbol{p}$ with genomes defining W_{bool}, W_{res}
for each generation g **do**
  for each genome in $\boldsymbol{p}$ **do**
    $circuit\_netlist \leftarrow phenotype(genome)$
    $res\_out \leftarrow$ `NGSPICE`$(circuit\_netlist)$
    if $res\_out \neq 0$ **then**
      Calculate W_{out} via ridge regression
      $\hat{\mathbf{y}} \leftarrow res\_out \times W_{out}$
      $f \leftarrow 100 - A\sqrt{\langle \|\hat{\mathbf{y}} - \mathbf{y}\|^2 \rangle}$
    end if
  end for
  $champion \leftarrow$ clone of genome with the best fitness
  $\boldsymbol{p}[0] \leftarrow champion$
  for each genome in $\boldsymbol{p}[1:N)$ **do**
    Remove a fraction ε of the smallest positive weights
    Every e generations, refresh the connection
    Else, adjust connections randomly
    Perform selection, crossover, and mutation operations
  end for
  Update all genomes based on fitness, crossover and mutation
end for
return optimized W_{bool}, W_{res}, W_{out}.

Table 3 shows the genome of our memristive circuit, with the evolutionary algorithm adjusting both topology (W_{bool}, '0' for no connection, '1' for a connection) and pulse widths (W_{res}), where relevant entries correspond to active connections limited to pulse widths between $[0, 0.5]$. Figure 3 illustrates the initialization process for configurations. The architecture includes W_{bool} and W_{res} as $N \times N$ matrices, with W_{out} as an $N \times M$ matrix connecting reservoir nodes to output neurons.

Circuit Evolution Algorithm To implement our model in hardware, we need to avoid hang terminals of nodes in the circuit, which are parts of the circuit that are disconnected from the rest. These can occur if the reservoir is initialized randomly. We prefer reservoirs with low clustering degree (sparse reservoirs) [15], which can facilitate the information flow among the reservoir nodes. Thus, we apply our proposed CNDR and RCNDR to generate the initial reservoirs.

Algorithm 1 outlines our evolutionary process. It begins with population initialization ('num_Pop'), followed by translating each genome into a circuit netlist. Circuit performance is evaluated via NGSPICE simulations, and fitness scores are recorded. The best genome is preserved as the champion gene. Adaptive scalability adjustments involve selectively removing weaker connections based on a

Table 3. Circuit representation of proposed memristor-based RC

Gene	Matrix Meaning	Size	Determination Method
$W_{\text{bool}}^{i,j}$	Reservoir topology	$N \times N$	Evolution
$W_{\text{res}}^{i,j}$	Configuration signal	$N \times N$	Evolution
$W_{\text{out}}^{i,j}$	Output weights	$N \times M$	Offline training

N indicates the reservoir size, and M represents the output dimension.

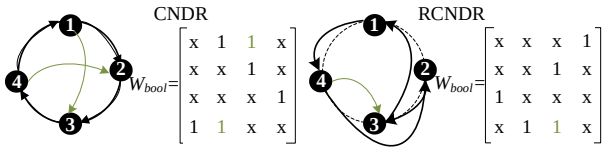

Fig. 3. Initialization example of CNDR and RCNDR.

threshold ε, depicted in Figure 4. Every 20 generations, removed connections are reinstated; otherwise, an equivalent number of new connections are added. The process also includes a crossover of selected parent genomes and the introduction of random mutations. After iterating through all generations, the algorithm outputs the optimized boolean and reservoir weight matrices (W_{bool} and W_{res}), and the output weight matrix (W_{out}), yielding the evolved solution for the reservoir computing system.

Figure 5 depicts six mutation operations: *Weight Mutation*, *Add Node*, *Delete Node*, *Internal Connection Mutation*, and *Input and GND Mutation*. In *Weight Mutation*, for values in W_{res} where W_{bool} is nonzero, there is a probability P_m of mutating each to a new value, uniformly selected from the allowable range. In the *Add Node* operation, the W_{bool} matrix is expanded to include a new node in a random position into the ring structure. This ensures that each new node connects to the subsequent one, with the final node linking back to the first. The *Delete Node* operation randomly removes a node, following a procedure similar to *Add Node*. For *Internal Connection Mutation*, a new connection between two previously unconnected nodes within the reservoir is randomly established if possible. The potential new connections (zero elements in the W_{bool} matrix, excluding self-connections) are counted, and one is randomly selected to set its corresponding W_{bool} value to 1, thereby creating the connection. In the *Input and GND Mutation*, the grounding of an input node is altered. For that, a random index from the *in_gnd* array, which stores indices of nodes that are either grounded or connected to an input, is selected. A new node, not already in the *in_gnd* array, is chosen to be grounded, replacing the currently grounded node at the selected index.

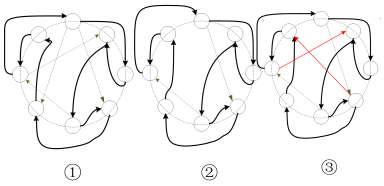

Fig. 4. Adaptive scalability adjustment to reservoir connection. (1) Identifying the connection with lowest weights. (2) Removing those connections. (3) Adding random connection with the same number of removed one.

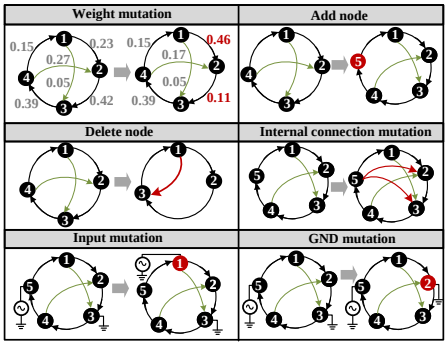

Fig. 5. Six types of mutation to the memristive reservoir circuit. In each sub-figure, the left part represents the individual before mutation, and right part represents the individual after mutation, where the red parts indicate the mutation parts, more detail could be found in Section 3.3.

4 Experimental Studies

The experimental studies are introduced in this section, where Section 4.1 describes the tasks and Section 4.2 presents the experimental results, including ablation experiments, comparisons with SOTA models and other hardware implementation of reservoirs, respectively. The number of hidden neurons of RNN, ESN and LSTM is set as 100, which is the same as our proposed approach.

4.1 Task Description

In order to verify the feasibility of proposed approach, three tasks are executed, including one wave generation task and two prediction tasks.

Wave Generation Task Memristors have been utilized as individual components or in network configurations for wave generation, as highlighted by Sillin et al. [27]. Furthermore, wave generation tasks are often employed to assess the practicality of hardware-based reservoir computing, a concept explored by Tanaka et al. [30]. In our experiments, the wave generation setup, depicted in

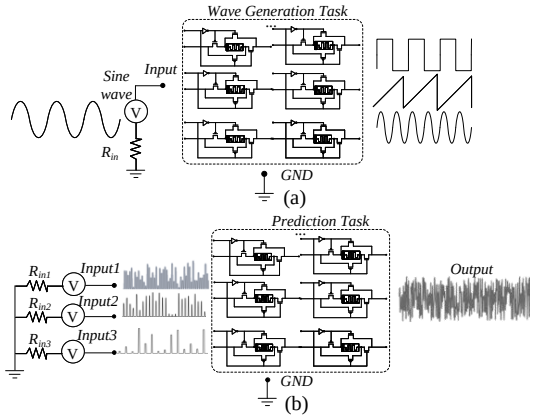

Fig. 6. The embryo circuits created for wave generation (a) and prediction task (b).

Figure 6 (a), involves inputting a 1 kHz, 5V sine wave into the reservoir. The desired outputs from the reservoir computing system include a square wave, a triangular wave, and a sine wave, all at 1 kHz with an amplitude of 1 mV, and an additional sine wave at 2 kHz with an amplitude of 1 mV.

Nonlinear Dynamic System Prediction Task We use the nonlinear autoregressive moving average (NARMA) systems of order 10 to test our method, which is the same as the one used in [26] We train on the first 1000 points of a NARMA sequence and test on the next 1000 points. We discard the first 200 points of both sequences as the washout period.

Nonlinear Audio Prediction Task In the field of audio prediction, the objective is to predict future audio samples based on a specified history horizon. Such techniques are vital in scenarios like audio restoration, where segments of audio are missing, or when dealing with impulsive noise. Researchers, including Greaves et al., have observed that long-term memory is significantly pronounced in music [10]. The dataset used in this study, sourced from Holzmann et al., comprises a brief recording from a Jazz quartet [12]. The dataset is divided into a training set, which includes the first 2000 data points, and a test set, comprising the subsequent 2000 points. The initial 200 data points are utilized as a washout period to stabilize the system before actual training begins.

4.2 Experiment Results

Ablation Experiment In this section, we test the impact of various operations within our proposed algorithm. Table 4 provides a comparison of our method when excluding *Crossover*, *Mutation* operation, and its different mutation operators within CNDR and RCNDR configurations across three tasks, using RMSE

Table 4. Results (RMSE) of ablation experiment

Tasks	All Equipped	No Crossover	No All Mutation	No Add	No Delete	No Step	No Input/GND	No Weight
CNDR								
Wave Generation	0.0164	0.1672	0.2883	0.1125	0.0384	0.1344	0.0217	0.0929
Nonlinear Dynamic System	0.0553	0.0671	0.1932	0.0564	0.0681	0.0692	0.0598	0.0701
Nonlinear Audio Prediction	0.1104	0.1238	0.2156	0.1144	0.1237	0.2092	0.1139	0.1324
RCNDR								
Wave Generation	0.0127	0.1321	0.2539	0.1044	0.0284	0.0965	0.0209	0.0913
Nonlinear Dynamic System	0.0369	0.0653	0.1436	0.0983	0.0548	0.0591	0.0471	0.0683
Nonlinear Audio Prediction	0.1028	0.1139	0.2045	0.1023	0.1123	0.1987	0.1102	0.1301

as the metric. A lower RMSE indicates better performance, while an increase in RMSE points to a decline in performance. The *All Equipped* condition consistently recorded the lowest RMSE values, signifying optimal performance with all features intact. The removal of specific features, particularly *No All Mutation*, led to significant increases in RMSE, highlighting the critical role of mutation operations in our proposed approach. Similarly, the removal of *Crossover (No Crossover)* and *Internal Connection Mutation(No Step)* resulted in substantial RMSE increases, underlining their importance in achieving effective results. RCNDR configuration often showed lower RMSE than CNDR, suggesting its greater efficacy or robustness for the tasks tested. This study underscores the vital contributions of mutation, crossover, and specific operational steps in maintaining RMSE and improving performance in complex computational tasks. Overall, the absence of crossover or mutation operations results in a decrease in the RSME of our algorithm.

Comparisons with SOTA Models With the objective of evaluating the predictive performance of the proposed approach, we conduct comparisons with several existing software baseline models and memristive reservoir circuits. The software basline models include RNN [9], ESN [13], and LSTM [11], and these memristive reservoir circuits have different topologies, including random, cycle, cycle with jump (CRJ), CNDR and RCNDR evolved by our proposed approach. Their regression performance comparisons are shown in Table 5. The results demonstrate that traditional software baselines perform well in tasks with temporal dependencies, such as LSTM in nonlinear audio tasks. Our proposed method CNDR and RCNDR display competitive, and in some cases superior, performance in waveform generation and nonlinear system tasks, highlighting the potential and efficiency of memristor technology in practical applications.

Comparisons with Other Types of Hardware Implementations of Reservoir Table 6 presents the comparisons of various hardware-based reservoirs. It is observed that both ESN and LSM models require n neurons to build the reservoir, with each neuron necessitating multiple Mosfets to replicate nonlinear interactions between the input and output signals. Traditional FPGA and PCB implementations, such as those by Yi et al. and Kume et al., rely heavily on a large number of MosFets and capacitors, leading to complex, power-intensive, and bulky circuits. As reported by Yi et al. [35], constructing a single neuron requires 37 Mosfets and 1 capacitor. Their setup involves over 60 neurons for the

Table 5. Results (RMSE) of SOTA model comparisons

Models	Wav. Gen.	Non. Sys.	Non. Aud.
Software baselines			
RNN [9]	0.0158	0.0448	**0.0277**
ESN [13]	0.0351	0.0773	0.1321
LSTM [11]	0.0129	0.0415	0.0393
Memristive reservoir circuits			
Random	-	-	-
Cycle [26]	0.4772	0.1943	0.1569
CRJ [26]	0.3551	0.1387	0.1498
Ours-CNDR	0.0164	0.0553	0.1104
Ours-RCNDR	**0.0127**	**0.0369**	0.1028

Table 6. Comparison of different hardware reservoirs

Method	Implementation	#Memristors	#MosFets	#Capacitors
FPGA-based ESN [35]	FPGA	0	37	1
Mosfet Crossbar-based ESN [17]	PCB	0	16	0
CMOS-based LSM [22]	IC	0	24	3
Our Method	**IC**	**1**	**4**	**0**

Components are listed per neuron. n indicates the number of neurons in the reservoir.

reservoir, amounting to a total of 2368 Mosfets and 64 capacitors. In contrast, our method, which utilizes memristors and manipulates current signals, employs merely 4 Mosfets and 1 memristor to achieve similar nonlinear dynamics. This significantly reduces the complexity of the circuit, enhancing scalability. Leveraging the diverse nonlinear behaviors offered by RMUs allows our approach to efficiently realize reservoir computing with only 30 nodes, thus presenting a more efficient and logical alternative compared to prior works [35, 17, 22].

5 Conclusion

Our study demonstrates the feasibility and benefits of using an evolvable topology in memristive reservoir computing systems. The proposed framework not only boosts computational performance, achieving higher predictive performance across tasks compared to current methods, but also reduces circuit area, enhancing practicality in resource-limited settings. These results underscore the potential of evolvable topologies to improve the efficiency and adaptability of neuromorphic computing, leading to more compact and effective hardware solutions for real-world applications.

In future work, we will extend our evaluations to a broader range of tasks to deepen our comparisons with methods like LSTM and improve robustness. We'll provide detailed assessments of metrics like RMSE, accuracy, and parameter counts. Additionally, we'll explore the trade-offs associated with our model's complexity and performance, particularly in scalability and robustness for larger networks. We also plan to include additional established state-of-the-art methods in our comparisons to enhance the validation of our findings.

References

1. Alomar, M.L., Canals, V., Perez-Mora, N., Martínez-Moll, V., Rosselló, J.L.: Fpga-based stochastic echo state networks for time-series forecasting. Comput. Intell. Neurosci. **2016** (2016)
2. Amil, P., Cabeza, C., Marti, A.C.: Exact discrete-time implementation of the mackey–glass delayed model. IEEE Trans. Circuits Syst. II, Exp. Brief **62**(7), 681–685 (2015)
3. Appeltant, L., Soriano, M.C., Van der Sande, G., Danckaert, J., Massar, S., Dambre, J., Schrauwen, B., Mirasso, C.R., Fischer, I.: Information processing using a single dynamical node as complex system. Nature Commu. **2**(1), 1–6 (2011)
4. Bai, K., Yi, Y.: Dfr: An energy-efficient analog delay feedback reservoir computing system for brain-inspired computing. ACM J. Emerg. Technol. Comput. Syst. **14**(4), 1–22 (2018)
5. Bala, A., Ismail, I., Ibrahim, R., Sait, S.M.: Applications of metaheuristics in reservoir computing techniques: a review. IEEE Access **6**, 58012–58029 (2018)
6. Chouikhi, N., Ammar, B., Rokbani, N., Alimi, A.M.: Pso-based analysis of echo state network parameters for time series forecasting. Appl. Soft Comput. **55**, 211–225 (2017)
7. Duport, F., Akrout, A., Smerieri, A., Haelterman, M., Massar, S.: Analog input layer for optical reservoir computers. arXiv preprint arXiv:1406.3238 (2014)
8. Ferreira, A.A., Ludermir, T.B., De Aquino, R.R.: An approach to reservoir computing design and training. Expert Syst. Appl. **40**(10), 4172–4182 (2013)
9. Giles, C.L., Kuhn, G.M., Williams, R.J.: Dynamic recurrent neural networks: Theory and applications. IEEE Trans. Neural Netw. **5**(2), 153–156 (1994)
10. Greaves-Tunnell, A., Harchaoui, Z.: A statistical investigation of long memory in language and music. In: Proc. 36th Int. Conf.Mach. Learn. pp. 2394–2403 (Long Beach, California, USA, 2019)
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
12. Holzmann, G.: Reservoir computing: a powerful black-box framework for nonlinear audio processing. In: Proc. 12th Int. Conf. Digit. Audio Eff. (DAFx) (Como, Italy, 2009)
13. Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. German Nat. Res. Cntr. Inf. Technol., GMD Report **148**(34), 13 (2001)
14. Jaeger, H.: Echo state network. Scholarpedia **2**(9), 2330 (2007)
15. Jaeger, H., Haas, H.: Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science **304**(5667), 78–80 (2004)
16. Kulkarni, M.S., Teuscher, C.: Memristor-based reservoir computing. In: Proc. IEEE/ACM Int. Symp. Nano. Archit. (NANOARCH). pp. 226–232. IEEE (2012)
17. Kume, Y., Bian, S., Sato, T.: A tuning-free hardware reservoir based on mosfet crossbar array for practical echo state network implementation. In: Proc. Asia South-Pacific Design Automat. Conf. (ASP-DAC). pp. 458–463. IEEE (2020)
18. Lukoševičius, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Computer Sci. Rev. **3**(3), 127–149 (2009)
19. Moon, J., Ma, W., Shin, J.H., Cai, F., Du, C., Lee, S.H., Lu, W.D.: Temporal data classification and forecasting using a memristor-based reservoir computing system. Nature Electronics **2**(10), 480–487 (2019)

20. Rodan, A., Tino, P.: Minimum complexity echo state network. IEEE Trans. Neural Netw. **22**(1), 131–144 (2010)
21. Rodan, A., Tiňo, P.: Simple deterministically constructed cycle reservoirs with regular jumps. Neural Comput. **24**(7), 1822–1852 (2012)
22. Roy, S., Banerjee, A., Basu, A.: Liquid state machine with dendritically enhanced readout for low-power, neuromorphic vlsi implementations. IEEE Trans. Biomed. Circuits Syst. **8**(5), 681–695 (2014)
23. Schmidhuber, J., Wierstra, D., Gagliolo, M., Gomez, F.: Training recurrent networks by evolino. Neural Comput. **19**(3), 757–779 (2007)
24. Schrauwen, B., D'Haene, M., Verstraeten, D., Van Campenhout, J.: Compact hardware liquid state machines on fpga for real-time speech recognition. Neural Netw. **21**(2-3), 511–523 (2008)
25. Shi, X., Minku, L.L., Yao, X.: Adaptive memory-enhanced time delay reservoir and its memristive implementation. IEEE Trans. Comput. **71**(11), 2766–2777 (2022)
26. Shi, X., Minku, L.L., Yao, X.: Evolving memristive reservoir. IEEE Trans. Neural Netw. Learn. Syst. pp. 1–15 (2023). https://doi.org/10.1109/TNNLS.2023.3270224
27. Sillin, H.O., Aguilera, R., Shieh, H.H., Avizienis, A.V., Aono, M., Stieg, A.Z., Gimzewski, J.K.: A theoretical and experimental study of neuromorphic atomic switch networks for reservoir computing. Nanotechnology **24**(38), 384004 (2013)
28. Takeda, S., Nakano, D., Yamane, T., Tanaka, G., Nakane, R., Hirose, A., Nakagawa, S.: Photonic reservoir computing based on laser dynamics with external feedback. In: Proc. Int. Conf. Neur. Inf. Process. pp. 222–230. Springer (2016)
29. Tanaka, G., Nakane, R.: Simulation platform for pattern recognition based on reservoir computing with memristor networks. Sci. Rep. **12**(1), 1–13 (2022)
30. Tanaka, G., Nakane, R., Yamane, T., Takeda, S., Nakano, D., Nakagawa, S., Hirose, A.: Waveform classification by memristive reservoir computing. In: Proc. 24th Int. Conf. Neur. Inf. Process. pp. 457–465. Springer (2017)
31. Tanaka, G., Yamane, T., Héroux, J.B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., Hirose, A.: Recent advances in physical reservoir computing: A review. Neural Netw. **115**, 100–123 (2019)
32. Wang, Q., Li, Y., Li, P.: Liquid state machine based pattern recognition on fpga with firing-activity dependent power gating and approximate computing. In: Proc. IEEE Int. Symp. Circuits Syst. (ISCAS). pp. 361–364 (Montréal, QC, Canada, 2016)
33. Wang, X., Jin, Y., Hao, K.: Evolving local plasticity rules for synergistic learning in echo state networks. IEEE Trans. Neural Netw. Learn. Syst. **31**(4), 1363–1374 (2019)
34. Yang, L., Zeng, Z., Shi, X.: A memristor-based neural network circuit with synchronous weight adjustment. Neurocomputing **363**, 114–124 (2019)
35. Yi, Y., Liao, Y., Wang, B., Fu, X., Shen, F., Hou, H., Liu, L.: Fpga based spike-time dependent encoder and reservoir design in neuromorphic computing processors. Microprocess. Microsyst. **46**, 175–183 (2016)
36. van der Zant, T., Bečanović, V., Ishii, K., Kobialka, H.U., Ploeger, P.: Finding good echo state networks to control an underwater robot using evolutionary computations. IFAC Proceedings Volumes **37**(8), 215–220 (2004)
37. Zhang, Y., Li, P., Jin, Y., Choe, Y.: A digital liquid state machine with biologically inspired learning and its application to speech recognition. IEEE Trans. Neural Netw. Learn. Syst. **26**(11), 2635–2649 (2015)
38. Zhong, Y., Tang, J., Li, X., Gao, B., Qian, H., Wu, H.: Dynamic memristor-based reservoir computing for high-efficiency temporal signal processing. Nature Commu. **12**(1), 1–9 (2021)