Evolving Memristive Reservoir

Xinming Shi, Leandro L. Minku *IEEE Senior Member*, and Xin Yao *IEEE Fellow*

***Abstract*—In light of the dynamic plasticity, nano-size and energy efficiency of memristors, memristive reservoirs have attracted increasing attention in diverse fields of research recently. However, limited by deterministic hardware implementation, hardware reservoir adaptation is hard to realize. Existing evolutionary algorithms for evolving reservoirs are not designed for hardware implementation. They often ignore the circuit scalability and feasibility of memristive reservoirs. In this work, based on the reconfigurable memristive units (RMUs), we first propose an evolvable memristive reservoir circuit that is capable of adaptive evolution for varying tasks, where the configuration signals of memristor are evolved directly avoiding the device variance of memristors. Second, considering the feasibility and scalability of memristive circuits, we propose a scalable algorithm for evolving the proposed reconfigurable memristive reservoir circuit, where the reservoir circuit will not only be valid according to the circuit laws, but also has the sparse topology, alleviating the scalability issue and ensuring the circuit feasibility during the evolution. Finally, we apply our proposed scalable algorithm to evolve the reconfigurable memristive reservoir circuits for a wave generation task, six prediction tasks, and one classification task. Through experiments, the feasibility and superiority of our proposed evolvable memristive reservoir circuit are demonstrated.**

***Index Terms*—memristor, reservoir computing, evolution algorithm, evolvable hardware, neural networks**

I. Introduction

RESERVOIR computing (RC) is a unified computational framework, originally derived from recurrent neural networks. It was originally proposed to provide possible solutions for the shortcomings of conventional recurrent neural network (RNN), like computationally expensive parameter update. Due to its modelling accuracy, modelling capacity, biological plausibility, as well extensibility and parsimony, RC methods have quickly become popular [1], [2], and constitute one of the core paradigms of RNN modelling.

The RC operation principle is based on a nonlinear dynamical system called reservoir. Specifically, a reservoir is a dynamic system that can perform nonlinear transformations of the input signals, and project them to a high-dimensional space (represented as the reservoir states). To reflect time correlations of input signals in the output signals, the reservoir needs to generate history-dependent dynamics. According to the work of Tanaka et al. [3], research on RC falls into two general categories, which are software and physical implementation, respectively. As for software implementation, mainstream research explores the practicability of applying RC in new areas or improving existing results by RC. Some researchers have modeled the dynamic behaviors of reservoir proposing simulated neural networks, such as ESN [4] and LSM [5]. They have been used to solve many computational problems, such as temporal pattern recognition, prediction, and generation tasks [6], [7]. The physical implementation of RC has also attracted increasing attention in diverse fields of research due to its fast speed of data processing and low learning cost [3]. Depending on different types of physical devices, electronic RC [8], photonic RC [9], and atomic switch RC [7] have been widely studied, where electronic RC have attracted great attention. A straightforward method of realizing electronic RC is to implement RNNs using neural network hardware or neuromorphic computing techniques. Researchers have employed different electronic devices to realize the electronic implementation of ESN and LSM following this straightforward method, such as FPGA [10] and MOSFET crossbar array [8]. Another method of realizing electronic RC is to employ other dynamical systems instead of RNNs. Therefore, some researchers mainly focus on exploring different dynamical systems that are equipped with the features of reservoirs, namely high dimensionality [11], nonlinearity [11] and short-term memory [12], to serve as reservoirs directly.

Memristor is a new-type nonlinear electronic component first established by Chua [13]. Since memristors are resistance-changeable, non-volatile, power-efficient and high-density integration-friendly [14], [15], memristor-based RC has attracted a large number of researchers. Some researchers followed the straightforward method of implementing RNN-based reservoirs by using both neuron and synapse circuits, so that memristor-based ESNs [16] and LSMs [17] could be realized. Some memristors can exhibit non-linear dynamics and short-term memory, which are in accordance with the key features of reservoirs. Therefore, some researchers made attempts to design memristive reservoirs based on these memristors without neuron circuits [18], [19], [20]. For example, Kulkarni et. al [18] and Gouhei Tanak et. al [20] have applied memristors only to construct the memristive networks for RC applied to pattern recognition.

However, as mentioned in previous studies [21], the construction of a random fixed reservoir has been regarded as not “good” enough for varying given tasks. A number of optimisation techniques have been explored to optimise the software reservoirs [22], but none, if any, have been used in hardware or memristive reservoirs. Compared with the software RC, designing or optimizing the memristive reservoirs is challenging, since the memristive circuit needs not only to operate as a reservoir but also to work legally and efficiently under the circuit laws. Considering the high development cost of ASICs (Application Specific Integrated Circuit), reconfigurable feature is also desirable for the implementation and optimization of memristive reservoir, which means that the

X. Shi and X. Yao are with Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech), Shenzhen, China, and School of Computer Science, University of Birmingham, UK. (e-mail: xxs972@cs.bham.ac.uk and xiny@sustech.edu.cn)

L. L. Minku is with School of Computer Science, University of Birmingham, UK. (e-mail:l.l.minku@bham.ac.uk)

memristive reservoir could be reconfigured on-chip for various tasks, showing the benefits to the research and developing cost of different reservoir circuits. However, the large search space by the vast number of connection combinations and scalable weights makes it difficult to design a suitable structure manually for a given task.

Evolutionary approaches have the advantage that they are often able to create a variety of different candidates for the solution [23], which may have good potential for the design and optimization of the memristive reservoir. However, even though several evolutionary algorithms have been designed for optimizing the simulated networks of reservoirs [24], there are very limited algorithms designed for optimizing hardware or memristive reservoirs directly. Several key issues involved in the hardware implementation of reservoirs are ignored in those algorithms, such as the feasibility of evolved design and scalability issue of evolved design. For example, Dale et al. [25] proposed an evolvable carbon nanotube reservoir by optimizing electrodes based on a basic genetic algorithm, where the scalability problem of evolving their physical reservoir has not been considered. Chatzidimitriou et al. [24] proposed an adaptive evolution and learning algorithm for optimizing a simulated network of reservoir (ESN), but it cannot be applied to evolve the hardware or memristive reservoir directly, since the feasibility of circuit has not been considered in their algorithm.

In this paper, we propose the first scalable algorithm for evolving reconfigurable memristive reservoir circuits on chip. On-chip methods could overcome the poor characteristics of pre-developed practical devices, thereby increasing performance of implemented systems. Our contributions are as follows:

- We design the first reconfigurable memristive reservoir circuit that can be evolved on-chip. This is achieved by exploiting different nonlinear behaviors of memristor currents.
- We propose a scalable adaptation algorithm for on-chip memristive reservoir evolution, providing an effective method of designing memristive reservoirs automatically. Specifically:
  Different from existing approaches, which evolve memristor states, the configuration signals are evolved directly to control the memristor currents, which can be done on chip. This prevents the situation where the actual memristance is different from the desired one due to device variances, which negatively affects existing approaches [26]. Different from the algorithms designed for evolving simulated networks of reservoirs, the feasibility and scalability of the memristive circuit are taken into the consideration in our proposed algorithm.
- We show that our proposed memristive reservoir was able to solve one generation and six prediction tasks, obtaining superior results compared to state-of-the-art approaches in terms of regression and circuit performance.

The rest of this paper is structured as follows. Section II introduces the related work. Section III proposes the evolvable memristive circuit design. Section IV proposes the adaptive evolution algorithm for evolving the memristive reservoir circuit. Section V describes the experiments for verifying our proposed evolvable memristive reservoir. The conclusions of our work are presented in Section VI.

II. Related Work

A. Hardware implementation of reservoir computing

Different types of RC algorithms have been implemented by hardware. ESN and LSM are two types of RC algorithms which are based on nonlinear function neuron and spiking neuron, respectively. Both ESN and LSM models have been fully developed in FPGAs for data recognition and classification. Yi et al. [10] proposed an FPGA-based ESN model with 64 neurons. An ESN model was also implemented in a FPGA for chaotic-time series forecasting [27]. An LSM model has been implemented with 135 neurons in a FPGA for pattern recognition to evaluate the presented FPGA neuromorphic processors. It achieved a recognition accuracy of 96.4% [17] on a speech recognition benchmark, the TI46 speech corpus.

The dynamic system-based RC model has also been implemented in hardware [28] in addition to those neuron-based RC models [10]. The photonic reservoir has recently gotten a lot of attention. However, signal processing using the photonic reservoir may necessitate the purchase of expensive peripheral devices such as a digitizer and waveform generator [29]. Electronic reservoirs are also being investigated for the development of low-cost machine learning devices [3]. Currently some of the electronic reservoirs are built on traditional Complementary Metal Oxide Semiconductor (CMOS) devices combined with other components such as capacitor and operational amplifiers [28], [30], [29]. Memristive reservoirs outperform CMOS reservoirs in terms of circuit area and power consumption due to the nano size and energy efficiency of memristors [19], [3]. There has been research [20], [18] using memristors to implement reservoir computing with greater energy efficiency and low training cost. However, they are based on the specified circuit, whose topology cannot be evolved adaptively to different tasks and cannot be changed dynamically during the circuit execution.

B. Evolution of reservoir computing

Some researchers asserted that just simply creating a reservoir at random is unsatisfactory. It seems obvious that, when addressing a specific modeling task, a specific reservoir design that is adapted to the task will lead to better results than a naive random creation [21]. Therefore, researching evolutionary reservoirs has been regarded as a natural idea. Recently, there have been several studies optimizing reservoirs to achieve better performance on a given application [23]. As mentioned in [31] and [21], most of the reservoir computing optimization and improvements are in the dynamic reservoir itself. In [32], both the number of nodes, reservoir connectivity, and weights have been optimized, with connectivity and input scaling playing a great role in the studied approach, and the quantity (number of connections) and quality (values) of weights having a big impact on the training process.

By adopting appropriate evolutionary algorithms, it is thus easy to perform better than average by choosing the right reservoir [23]. This has been done with genetic algorithms [33], Evolino [22], evolutionary strategy [34], and Particle Swarm Optimization (PSO) [32]. Generally, there have been three classes of evolutionary algorithms to optimize RC [35]. The first is to optimize the global parameters of the reservoir, such as the spectral radius, and the scaling of weights. The second is to optimize the topology/architecture of the network directly, such as weight connections. The third class can be done in a hybrid way of the other groups. However, there is no research focusing on the optimization of hardware or memristive reservoirs [25].

III. Memristive Reservoir Circuit Design

A. Memristor model

In software RC, a reservoir can perform nonlinear transformations of the input signals, and project them to a high-dimensional space by its short-term memory effect. Therefore, in order to implement the circuit counterpart of RC, the reservoir circuit that can exhibit short-term memory should be constructed first. According to [3], some of the memristive devices or systems are capable of exhibiting nonlinear dynamic behavior in a short-term memory manner. In this work, we apply the memristor model proposed by [36], which is equipped with short-term memory (forgetting effect), to construct the reservoir part of the circuit.

The memristor model proposed by Chen et al. [36] has the forgetting effect, which will be applied to implement the short-term memory effect of the reservoir in this work. Its mathematical model is shown as follows:

$$i = (1-x)\alpha[1-e^{-\beta v}] + x\gamma sinh(\delta v), \tag{1}$$

$$\dot{x} = (\lambda[e^{\eta_1 v} - e^{\eta_2 v}] - \frac{x-\theta}{\tau})f(x), \tag{2}$$

$$\dot{\varepsilon} = \sigma(e^{\eta_1 v} - e^{\eta_2 v})f(x), \tag{3}$$

$$\dot{\tau} = \theta(e^{\eta_1 v} - e^{\eta_2 v}), \tag{4}$$

$$f(x) = \frac{(sign(v)+1)(sign(1-x)+1) + (sign(-v)+1)(sign(x)+1)}{4}, \tag{5}$$

where i is the current passing through memristor and v is the voltage applied across the memirstor; x is the Ohmic-like conducting channel, which is equivalent to the conductance and has been normalized to $[0, 1]$, and $x = 0$ indicates fully schottky-dominated conduction while $x = 1$ indicates fully tunneling-dominated conduction; α is the barrier height for Schottky barrier; β denotes the depletion width in the Schottky barrier region; γ is the barrier height for tunneling; η_1 and η_2 are the interface effect with positive voltage and negative voltage, they are all positive-valued fitting parameters determined by material properties and in dependent of x; ε is the retention of the Ohmic-like conducting channel, which can vary within the range [0, 1]; λ is a positive constant of controlling the changing rate of x; τ is the diffusion time; δ and θ are the corresponding parameters for ε and τ; $f(x)$ is the window function for better elaborating the memristor dynamics. The values of the parameters used in this work are given in Table I of the supplementary material. They were adapted from the parameters of the bipolar model [36] by tuning through circuit simulation tests to ensure the memristors can generate fading dynamics within our simulation time window (0.5s) one by one.

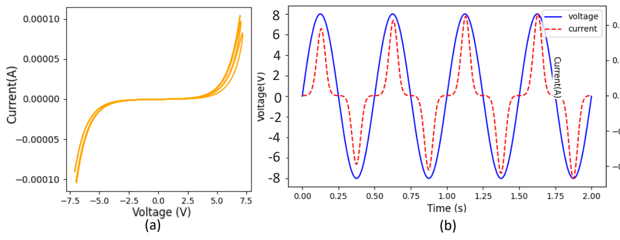


Fig. 1. (a) I–V hysteresis curves of the forgetting memristor model [36] (b) The circuit simulation result with applied sine voltage.

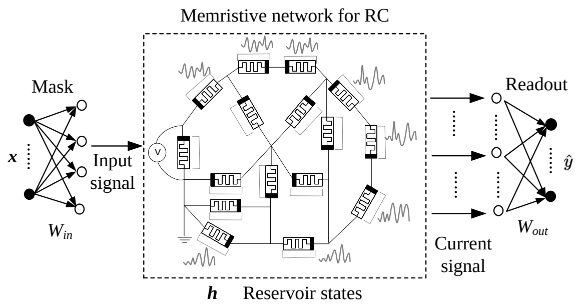


Fig. 2. Schematic illustration of memristive network for RC with random topology, redrawn from work [20]

According to Equation (1), there is a nonlinear relationship between the applied voltage and the current flowing across the memristor. We also performed circuit simulation tests to verify this nonlinear relationship. Fig. 1 (a) shows the I-V hysteresis curves of the forgetting memristor model [36], which exhibit high nonlinearity between the voltage and current. The applied voltage and the current flowing across the memristor are shown in temporal domain in Fig. 1, where the nonlinearity transformation is also explicit.

In this paper, the evolvable memristor-based reservoir computing is introduced, which is based on the reconfigurable memristive units proposed in our previous work [37]. Considering the synchronous weight adjustment of the proposed reconfigurable memristive unit, it is suitable to be applied to change and evaluate the reservoir topology and weights. In addition, the overall circuit architecture of evolvable memristor-based reservoir computing is also introduced in this section.

B. Memristive Network for RC

Fig. 2 shows the schematic illustration of the memristive network for RC with random topology [20]. After applying a masking operation to x, it is fed into the memristive network. This process has two effects. First, the input mask distributes the information contained in the same time series value into all neurons and it makes the dimensional multiplexing of the input. Second, the mask values with zero mean make the input time series x with non-zero mean become zero; such property is convenient for eliminating the intercept in ridge regression. Regarding the physical reservoir computing, the function of the input mask layer was realized through the form of pre-processing [26], [20]. As for our proposed approach, the same as the previous work [26], [20], it is also implemented by the form of input pre-processing and will not be trained.

The input signal $\mathbf{x}(t) \in \mathbb{R}^{1\times n}, t \in \{1,...,T^*\}$ of the reservoir comes together with a corresponding teaching signal $\mathbf{y}(t) \in \mathbb{R}^{1\times n}, t \in \{1,...,T^*\}$ for training purposes. Since we use a linear readout layer in our proposed reservoir model, for each input signal and reservoir layer $\mathbf{h}(t) \in \mathbb{R}^{1\times N}, t \in \{1,...,T^*\}$, an n-dimensional output $\hat{\mathbf{y}} \in \mathbb{R}^{1\times n}, t \in \{1,...,T^*\}$ can be obtained by using an output parameter matrix $W_{out} \in \mathbb{R}^{N\times n}$ and by setting $\hat{\mathbf{y}} := \mathbf{h}(t)\cdot W_{out}, t \in \{1,...,T^*\}$.

The training consists of finding the output parameter matrix W_{out} that minimizes the distance between the outputs and the teaching signals, with L^2 norm regularization. This amounts to solving the following optimization problem:

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

where $\lambda||W||^2$ refers to the regularisation term to prevent overfitting by limiting the norm of the solution, and $\lambda \geq 0$ is a hyperparameter that controls its intensity. In order to solve this problem, ridge regression has been applied, whose solution is given by:

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

As mentioned in Section III-A, a memristor could be generally described by an algebraic equation and a differential equation, which are as follows:

$$I = G(w,V)V, \tag{8}$$

$$\frac{dw}{dt} = f(w,V), \tag{9}$$

where the function f determines how the internal state behaves depending on the input voltage. Therefore, a network of memristors could be used as a reservoir that maps the input signal into the high-dimensional feature space. The current signal of memristors will be used as the output signal of the memristive RC. Then, the output signal of memristive reservoir can be processed by the readout layer multiplying with W_{out}, so that we can get the actual output $\hat{\mathbf{y}}$.

C. Construction of Memristive Reservoir based on Reconfigurable Memristive Unit (RMU)

We propose to construct the memristive network for RC in a reconfigurable manner based on the reconfigurable memristive unit (RMU) from [37] to construct the memristive network. Since the memristance of this RMU is capable of being tuned synchronously, it is suitable to construct the memristive reservoir. The RMU is composed of four transistors and one memristor, as shown in Fig. 3. In Fig. 3, V_{in} and V_c represent

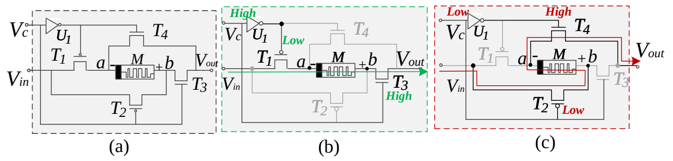


Fig. 3. (a) Circuit schematic of reconfigurbale memristor-based unit; (b) Equivalent circuit when $V_c = 1$ (High voltage level); (c) Equivalent circuit when $V_c = 0$ (Low voltage level).

the input signal and control signal, respectively. Control signal V_c has two states, which are *logic 0* and *logic 1* represented as $0V$ and $5V$ voltage, respectively. T_1 and T_2 are PMOS transistors, while T_3 and T_4 are NMOS transistors. The input signal V_{in} will be fed into the unit from the source s terminal of T_1, and the control signal V_c will be fed into the unit by the inverter U_1. The gate g terminals of T_1 and T_4 are connected to the signal V_c, while the gate g terminals of T_2 and T_3 are connected to the control signal V_c. One terminal of memristor is connected to T_1 and T_4, and another one is connected to T_2 and T_3.

When the input voltage V_{in} is 0, there will be no current or voltage flowing through the memristor, so that the state of the memristor will not change. When the input voltage V_{in} is not 0, there will be two working situations according to two states of control voltage V_c. Fig. 3 shows the circuit schematic of reconfigurbale memristor-based unit and the equivalent circuits with different levels of V_c. Terminals a and b of the memristor M are the bottom (-) and top (+), respectively. When V_c stays in a high voltage level, the transistors T_1 and T_3 turn on, T_2 and T_4 turn off, so that the current flows from a to b. This can be regarded as applying the negative voltage from the top terminal, incurring an increasing memristance. Fig. 3(b) shows the equivalent circuit. When V_c stays in a low voltage level, the transistors T_2 and T_4 turn on, T_1 and T_3 turn off, so that the current flows from b to a. This can be regarded as applying the positive voltage from the top terminal (+), incurring a decreasing memristance. Fig. 3(c) shows the equivalent circuit. By changing the voltage states and the pulse width of control voltage V_c, the current will flow in different directions and the memristance will vary to different values. Therefore, we propose to connect the RMUs and feed them with different voltage states and pulse width of control voltage V_c so that circuits with different topologies and memristances can be implemented. This can then be used as configurable memristive reservoirs.

Fig. 4 gives an example to illustrate how the memristor-based reconfigurable unit works. The upper figure shows the input signal V_{in}, where we use the sine wave as the example. The bottom two plots provide two situations of control signal V_c and its corresponding current flowing across the memristor M. For the first situation, the frequency of V_c is larger compared with that of the second situation. During the first 0.1s, there is no input signal ($V_{in} = 0$), so that the currents on both of the two cases are also 0. When V_c stays in *logic 1*, the T_1 and T_3 will be turned on, while T_2 and T_4 will be turned off, so that the current of M will flow from point a to b, which is the negative direction for the memristor. When V_c

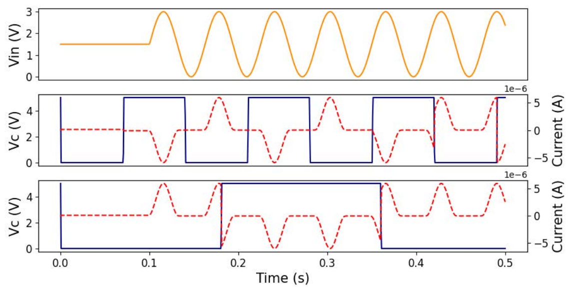


Fig. 4. Simulation results of reconfigurbale memristor-based unit. The yellow line denotes the V_{in}, the red dash line denotes the current flowing across the memristor and the blue solid line denotes the configuration voltage V_c.

stays in *logic 0*, the T_2 and T_4 will be turned on, while T_1 and T_3 will be turned off, so that the current of M will flow from point b to a, which is the positive direction for the memristor.

The left part of Fig. 5 shows an example circuit composed of 9 RMUs. The right part of Fig. 5 shows different states of control voltage V_c and the corresponding equivalent circuits. The example circuit is composed of 9 RMUs, and the voltage states of V_c of the different RMUs control the current directions of their corresponding memristors. The table shows 4 examples of possible circuit topologies, where the voltage states with blue highlight indicate that the corresponding RMUs have been selected to construct circuits, and the voltage states with the darker highlight indicate their present voltage states (0 or 1). Taking the first situation as an example, the units RMU_{1_1}, RMU_{2_1}, RMU_{2_2} and RMU_{3_3} are selected, where the voltage states of V_{c1_1}, V_{c2_1}, V_{c2_2} and V_{c3_3} are 0, 1, 1, and 1, respectively, and then an equivalent circuit composed of 4 memristors is obtained. In the same way, more circuits could be constructed.

In summary, using the RMU, current states with varying dynamic behaviours could be generated to create memristive reservoirs by innovatively controlling the voltage Vc rather than tuning their memristance. This operation can overcome the device variance of memristors, thereby increasing performance of implemented systems. Reservoir states with sufficient dynamic behaviours could be implemented using the various nonlinear and fading states of generated currents. Furthermore, the circuit evolution platform could be built to support the evolution of memristive reservoir circuits by connecting the RMUs and tuning their applied control voltages Vc in an evolvable manner. This means that the memristive reservoir could be evolved on-chip for the first time. This can prevent device variance of memristors, such as device-to-device variance, potentially leading to more accurate reservoirs. The details of the evolutionary algorithm design will be given in Section IV.

IV. Evolutionary Design of Memristive Circuits

In this work, a scalable evolutionary algorithm for optimizing memristor-based reservoir computing is proposed, which can evolve the memristor-based reservoir in an adaptively sparse manner. The genome (Section IV-A), initialization method (Section IV-B), evolution algorithm design (Section IV-C), as well the crossover (Section IV-C1) and mutation operations (Section IV-C2) are introduced in this section.

TABLE I
Genome of proposed memristor-based RC

Genes	Matrix Meaning	Size	How to determine
$W_{bool}^{i,j}$	Reservoir topology	$N \times N$	Evolution
$W_{res}^{i,j}$	Configuration signal	$N \times N$	Evolution
$W_{out}^{i,j}$	Output weights	$N \times M$	Offline training

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

A. Chromosome Representation

As mentioned in the previous section III-B, the classical paradigm of reservoir computing is composed of an input layer, a reservoir and an output layer. Therefore, we apply a binary adjacency matrix W_{bool} to represent the topology of reservoir, and matrices W_{res} and W_{out} to represent the weight values of the reservoir and output layer, respectively. W_{bool} and W_{res} are evolved by our proposed algorithm, whereas W_{out} is set by offline training through ridge regression using Equation (7).

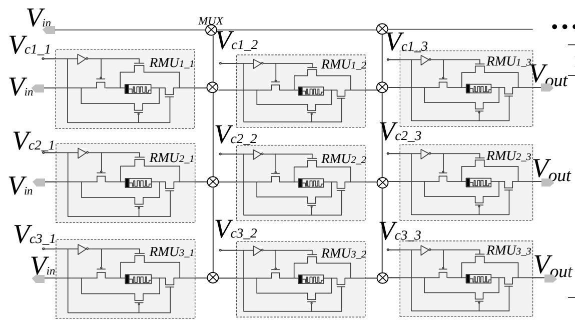
In our proposed memristor-based reservoir computing, the states of reservoir nodes are represented by the states of the current flowing through the memristors. As shown in Fig. 3, the conductance of memristor can be tuned by the control voltage V_c with different pulse widths, during which the internal current state of memristor will also keep changing under the applied voltage. Therefore, instead of evolving the weight values of the reservoir directly, the weights between nodes are evolved by a matrix of the pulse width of control voltage V_c, by which the currents flowing across the memristors can be controlled to present different dynamic behaviours.

Table I shows the genome of the proposed memristive reservoir circuit. The topology and configuration signal matrix of memristive reservoir will be evolved during the evolution procedure. The reservoir topology is represented by an adjacent matrix W_{bool}, of which value 0 represents that there will be no connection between the corresponding nodes, while value 1 represents that there will be a connection between the two corresponding nodes. The diagonal of W_{bool} has value 0 since there is no self-connection of nodes in our work. Besides the topology of the reservoir, the configuration signals between RMUs are also evolved, and are represented by a $N \times N$ matrix W_{res}. It should be noted that only if the corresponding value in W_{bool} is 1, the weight of the connection will be valid in matrix W_{res}. Assuming the number of reservoir nodes and output neurons are N and M, the sizes of W_{bool}, W_{res} and W_{out} are $N \times N$, $N \times N$ and $N \times M$, respectively.

As for the matrix W_{res}, its values are limited in $[0, 0.5]$, which is the pulse width of the voltage applied to them.

B. Sparse Initialization

Regarding hardware implementation, there will be problems of hang terminals of nodes in the circuit if the reservoir follows a completely random initialization. Hang terminals indicate



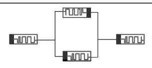
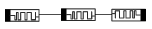
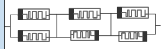
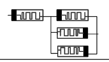
NO.	$V_{c1\text{-}1}$	$V_{c1\text{-}2}$	$V_{c1\text{-}3}$	$V_{c2\text{-}1}$	$V_{c2\text{-}2}$	$V_{c2\text{-}3}$	$V_{c3\text{-}1}$	$V_{c3\text{-}2}$	$V_{c3\text{-}3}$	Equivalent Circuits
1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	
2	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	
3	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	
4	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	

Fig. 5. Circuit example composed by 9 RMUs and the corresponding equivalent circuits under the different states of control voltage V_c

the part of the circuit with open connection with the whole circuit. Therefore, we do not initialize the reservoir entirely at random. It has been argued that reservoirs should ideally have small clustering degree (sparse reservoirs) [6], so that the dynamic information flow through the reservoir nodes is not too cluttered. Therefore, we adopt cycle reservoirs with regular jumps (CRJ) [38] to initialize the candidate reservoirs. According to previous work [38], CRJ leads to a fixed simple regular topology with sparse connections. The reservoir nodes are connected in a unidirectional cycle with bidirectional shortcuts (jumps). So the nonzero elements of W_{res} are:

- The lower sub diagonal $W_{res}^{i,j}$, where $i = j + 1$.
- The upper-right corner $W_{res}^{1,N}$.
- The jump entries. Consider the jump size $1 < l < \lfloor N/2 \rfloor$, if (N mod l)= 0, there will be N/l regular jumps. If (N mod l)$\neq$ 0, then there will be $\lfloor N/l \rfloor$ regular jumps.

By this sparse initialization, there will be a slightly higher degree of local clustering while achieving a much smaller average path length. Moreover, in term of the circuit validity, there will be no hang terminals since the memristors are connected in series or parallel in the reservoir circuits. Based on this sparse initialization, we can prevent excessive connections in the circuits, ensuring a scalable topology during the evolution.

C. Evolutionary Algorithm

The pseudo code of reservoir circuit evolution is described in Algorithm 1. The first step is to initialise the population. Then, for all num_Pop candidate reservoirs, they are first transferred into corresponding circuit netlists according to Section IV-A, and given their fitness evaluation. The champion gene will be kept in the population. This procedure is repeated for a maximum number of generations num_Gen. Regarding the fitness evaluation, it contains two stages. First, the circuit netlist of the training is simulated by using NGSPICE. The currents of the memristor-based reconfigurable units in a reservoir are recorded as res_out. Second, the weights of the output layer W_{out} are calculated using ridge regression. Then, the actual output $\hat{\mathbf{y}}$ is calculated by $res\_out \times W_{out}$ and the fitness is calculated by Equation (10):

$$fitness = 100 - A\sqrt{\left\langle \|\hat{\mathbf{y}}(t) - \mathbf{y}(t)\|^2 \right\rangle}, \tag{10}$$

where 100 is an arbitrary value to convert the problem into a maximization problem; A is a scaling factor to control how large the differences between the scaled error (right term) and 100 (left term) are; $\mathbf{y}(t)$is the desired output (target); $\hat{\mathbf{y}}(t)$ is the readout output; $\|.\|$ denotes the Euclidean norm; and $\langle . \rangle$ denotes the empirical mean calculated over a set of examples.

After the fitness evaluation, the best gene will be kept to execute the *Adapt* operation. This operation is designed so that the reservoir connections can be evolved gradually and the reservoir is kept sparsely connected [39]. $IsAdapt$ is a Boolean value pre-defined parameter, where value 0 represents that the genomes will not go through adaptive sparse adjustment while value 1 represents that the genomes will go through adaptive sparse adjustment. Specifically, a fraction of the weights, the ones closest to zero, are removed. And then, new weights are added randomly in the same amount as the ones previously removed, following the Pseudo code shown in Algorithm 2. Next, two parents are selected and they go through crossover and mutation with probability P_c and P_m, respectively. Their details are introduced in Sections IV-C1 and IV-C2.

1) Crossover: Algorithm 3 displays the pseudocode of the crossover operation. As for the crossover of reservoir's genome, the parent could be regarded as the product of W_{res} and W_{bool} of one individual. For the crossover operation of parents with different sizes, the size of offspring's reservoir should be determined firstly. There are two alternatives of determining the size of offspring's reservoir, which are to follow the parent with larger size and to choose the size of the parent with better fitness, respectively. The crossover probability is P_c, which is to decide whether to take the crossover operation. Besides the determination of offspring's size, the weights value of offspring's reservoir are determined by the following rule [24]:

$$w_{ij} = \begin{cases} \frac{w_{ij}^1 + w_{ij}^2}{2} & \text{if } w_{ij}^1, w_{ij}^2 \neq 0 \text{ and } random < 0.5, \\ w_{ij}^1 & \text{if } w_{ij}^1, w_{ij}^2 \neq 0 \text{ and } 0.5 \leq random < 0.75, \\ w_{ij}^2 & \text{if } w_{ij}^1, w_{ij}^2 \neq 0 \text{ and } 0.75 \leq random < 1.0, \\ w_{ij}^1 & \text{if } w_{ij}^2, = 0 \text{ and } w_{ij}^1 \neq 0, \\ w_{ij}^2 & \text{if } w_{ij}^1, = 0 \text{ and } w_{ij}^2 \neq 0. \end{cases} \tag{11}$$

Fig. 6 shows an example that explains how the parents with different sizes execute the crossover operation. As shown in Fig. 6, the sizes of $parent_1$ and $parent_2$ are different, where

Algorithm 1 Pseudo code of reservoir evolution.

```
1: Set the probability of crossover and mutation $P_c$ and $P_m$, generation $g$, tournament size num_Tour, minimum and maximum size of reservoir num_ini_node and num_max_node
2: Population initialization $\boldsymbol{p}$'s genomes by setting their $W_{bool}$, $W_{res}$
3: for each generation $g$ do
4:     for each genome do
5:         $circuit\_netlist \leftarrow phenotype(genome)$
6:         $res\_out \leftarrow$ NGSPICE($circuit\ netlists$)
7:         if $res\_out \neq 0$ then
8:             Calculate $W_{out}$ using ridge regression.
9:             $\hat{\mathbf{y}} \leftarrow res\_out \times W_{out}$
10:            $f \leftarrow 100 - A\sqrt{\left\langle \|\hat{\mathbf{y}} - \mathbf{y}\|^2 \right\rangle}$
11:        end if
12:    end for
13:    $champion \leftarrow$ clone of genome with the best fitness
14:    $\boldsymbol{p}[0] \leftarrow champion$
15:    for each genome in $\boldsymbol{p}[1, N)$ do
16:        if $IsAdapt$ then
17:            $genome \leftarrow Adapt(genome, g)$
18:        end if
19:        $parent_1, parent_2 \leftarrow selection(all\ genomes)$
20:        $genome \leftarrow crossover(parent1, parent2, P_c)$
21:        $genome \leftarrow mutate(genome, P_m)$
22:    end for
23: end for
24: Return $W_{bool}$, $W_{res}$, $W_{out}$
```

$parent_1$ is a 3×3 weight matrix and $parent_2$ is a 5×5 weight matrix. Therefore, there will be three types of weight pairs on these two matrices, which are matching pair, disjoint pair and excess part. The yellow area shows the overlapped part of $parent_1$ and $parent_2$, while the gray area shows the excess part. The matching pairs (green solid line) in the overlapped part represent that two weights in the same position of two parents' matrix have the same value type, which means are all "zero elements" or all "nonzero elements". And the rest of the situation belongs to the disjoint pairs (red dash line). Based on the size of the reservoir, excess weights are either completely adopted or completely discarded. The weights with disjoint or matching connection are either averaged or selected from either parent based on Equation (11).

2) Mutation: In order to encourage diversity of reservoirs during the evolution, five types of mutation operators are applied:

- **Weight mutation**: For the values in W_{res} corresponding to the position where W_{bool} is not zero, there will be the probability P_m to mutate them to a new value taken uniformly at random within the allowable range.
- **Add node:** To add a node to the reservoir and initialize its corresponding W_{bool} matrix to 0.
- **Delete node:** To calculate the weight sum of W_res associated with each node, and delete the node with the smallest weight sum.

Algorithm 2 Pseudo code of adaptive sparse $Adapt(genome, g)$

```
1: set sparsity $\varepsilon$
2: if $IsAdapt$ then
3:     remove a fraction $\varepsilon$ of the smallest positive weights by setting their corresponding $W_{bool}^{i,j}$ to zero
4:     if $g$ mod $e == 0$ (at every $e$ generations) then
5:         return the sparse connection
6:     else
7:         add randomly new weights in the same amount as the ones removed previously
8:     end if
9: end if
```

Algorithm 3 Pseudo code of crossover $crossover()$

```
1: $parent_1, parent_2$ selection by tournament strategy
2: if $parent_1.size()! = parent_2.size()$ then
3:     if $random < P_c$ then
4:         if $parent_1.size() > parent_2.size()$ then
5:             $offspring = parent_1$
6:         else
7:             $offspring = parent_2$
8:         end if
9:     else
10:        if $parent_1.fitness > parent_2.fitness$ then
11:            $offspring = parent_1$
12:        else
13:            $offspring = parent_2$
14:        end if
15:    end if
16:    for $w_{ij}$ in $offspring$ do
17:        if $w_{ij}$ is in overlapped area then
18:            $w_{ij} \leftarrow$ wights_update()
19:        end if
20:    end for
21: end if
```

- **Jump mutation:** The jump step of CRJ structure will be mutated. The value range of the step is between 2 and N/2. According to the new jump step, W_{bool} will be updated, so that the CRJ structure could be rebuilt.
- **Input/GND node mutation:** The position of reservoir nodes connected to the input signal and GND will be mutated to increase circuit diversity picking a new position uniformly at random, since different terminals of the circuit connected to Input or GND could be a different circuit.

Besides encouraging diverse candidate reservoirs, these operators maintain circuit validity during the evolution. Regarding the reservoir diversity, either of nodes number, connection ways and their degree could be mutated by our proposed five mutation operations, which is beneficial to the diversity of memristive reservoir circuits. Regarding the circuit validity, the proposed add, delete or jump operators will generate or delete nodes based on the W_{bool} and W_{res}, so that there will be no hang terminals that incur the invalid circuits, like open

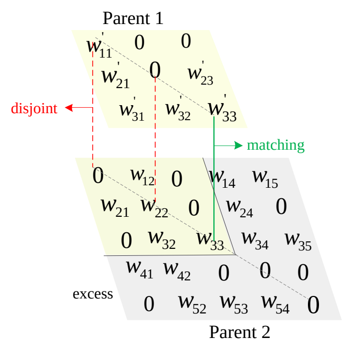

Parent 1
w'_{11} 0 0
w'_{21} 0 w'_{23}
w'_{31} w'_{32} w'_{33}
disjoint
matching
0 w_{12} 0 w_{14} w_{15}
w_{21} w_{22} 0 w_{24} 0
0 w_{32} w_{33} w_{34} w_{35}
w_{41} w_{42} 0 0 0
excess
0 w_{52} w_{53} w_{54} 0
Parent 2

Fig. 6. Crossover operation for the parents with different sizes.

circuits. The operator of Input/GND mutation will ensure that there will be always input and GND terminals in the evolved circuit. Therefore, our proposed mutation operations ensure that the mutated individuals remain feasible circuits during the evolution. Moreover, the generated circuits will all be taken through circuit simulation on NGSPICE, where circuits that fail the NGSPICE simulation are given a bad fitness value as the penalty. This may happen, for instance, if there is a circuit convergence problem or any other problems that can only be revealed through simulation.

In summary, our approach is the first to enable the memristive reservoir circuits to be evolved on-chip. This is achieved through our proposed scalable adaptation algorithm incorporated into our evolutionary algorithm, based on the proposed circuit evolution platform. The reservoir circuits can be sparsely initialised and furnished with circuit validity using the CRJ-type initialization approach. In addition, circuit-specific genetic operators are designed, enhancing the circuit's diversity and validity. The function $Adapt()$ is important for not only maintaining the sparse topology but also to progressively developing useful connections.

V. Experimental Studies

A. Test Tasks

In order to verify the feasibility of proposed adaptive memristive RC, seven tasks are executed, including one wave generation task and six prediction tasks.

1) Wave Generation Task: Memristors have recently been used as single devices or in the form of networks for wave generation tasks [7]. In addition, wave generation task also has been commonly applied to verify the feasibility of hardware reservoir computing [19]. The wave generation task used in our experiments is illustrated in Fig. 7 (a), where the input voltage of the reservoir is the sine wave with 1K HZ and 5V amplitude, and the output target of the RC is square wave with 1K HZ and 1mV amplitude (orange line), triangular wave with 1K HZ and 1mV amplitude (red line), and sine wave with 2K HZ and 1mV amplitude (green line), respectively. The bottom plot of the Fig. 7 (a) shows an example of the current signals used with the RMUs to generate the output waves, where the input signal could be mapped into the high-dimensional feature space by the currents of RMUs. Therefore, there will be 900×3 points to be generated in the wave generation task.

2) Nonlinear Dynamic System Prediction Task: We considered the NARMA systems of order 10 [40] to verify our proposed method:

$$\begin{aligned} y\left(t+1\right) =& 0.3y\left(t\right)+0.05y\left(t\right)\sum_{i=0}^{9}y\left(t-i\right) \\ &+1.5s\left(t-9\right)s\left(t\right)+0.1, \end{aligned} \tag{12}$$

where $s\left(t\right)$ is a random input series ranged from $[0, 0.5]$ and $y\left(t\right)$ is the output of the system. NARMA tasks aim at measuring the ability of a neural network to model nonlinear and long-term memory systems. We selected the first 1000 of a NARMA sequence for training, and the remaining 1000 were used for testing. The first 200 values from the training and test sequences were used as the initial washout period.

3) Nonlinear Audio Prediction Task: In audio prediction, one tries to forecast future samples out of a given history horizon. Such methods are necessary for instance in audio restoration, whenever a sequence of consecutive samples is missing, or when impulsive noise appears. Researchers found that long memory appears to be strongly represented in music [41]. This dataset is from [42], which is a short recording of a Jazz quartet. The training set consists of the first 2000 points and the test set consists of the next 2000 points. The first 200 values were used as the initial washout period.

4) ARFIMA Series Prediction Task: We generated the ARFIMA series using the following model with long memory effect:

$$(1-0.7B+0.4B^2)(1-B)^{0.4}Y_t=(1-0.2B)\varepsilon_t \tag{13}$$

where B is the backshift operator, Y_t is the time series at the time t, and ε_t indicates the error, and the previous forecast is adjusted in the direction of the error. The training and testing data length are set as 2000 and 2000, respectively. The first 200 values from the training and test sequences were used as the initial washout period.

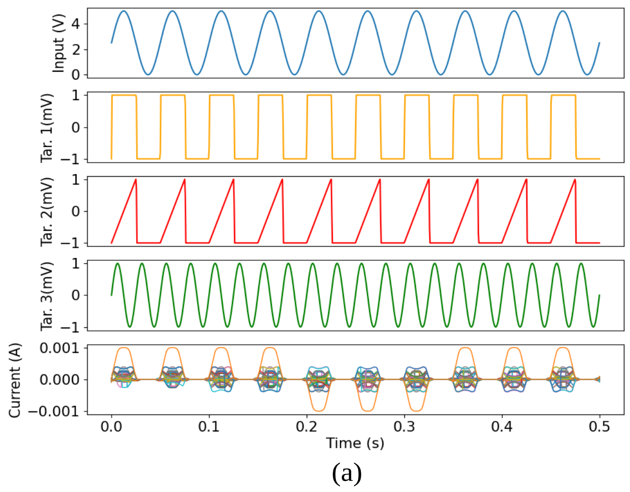
5) Tree Ring Prediction Task: This dataset contains 4351 tree ring measures of a pine tree from an Indian Garden, Nevada Gt Basin obtained from R package tsdl [1], where 1000 items are used for training, and 1000 for testing. And the first 200 values from them were used as the initial washout period.

6) Dow Jones Industrial Average (DJI) Prediction Task: The raw dataset contains DJI daily closing prices from 2000 to 2019 obtained from Yahoo Finance, where 1000 items are used for training and 1000 for testing. And the first 200 values from them were used as the initial washout period.

7) Santa Fe Laser Prediction Task: Santa Fe Laser data set was used [2], which consists of a cross-cut through periodic to chaotic intensity pulsations of a real laser. The length of training and test set are both 2000, where the first 200 values from them are used as the initial washout period.

8) Dynamic Gesture Recognition (DGR): This consists of spatioatemporal time series data for the task of recognizing dynamic gestures. These dynamic gestures are recorded in three dimensions at a sample frequency of 10 Hz. Typical

[1]https://pkg.yangzhuoranyang.com/tsdl/
[2]http://web.cecs.pdx.edu/~mcnames/DataSets/index.html



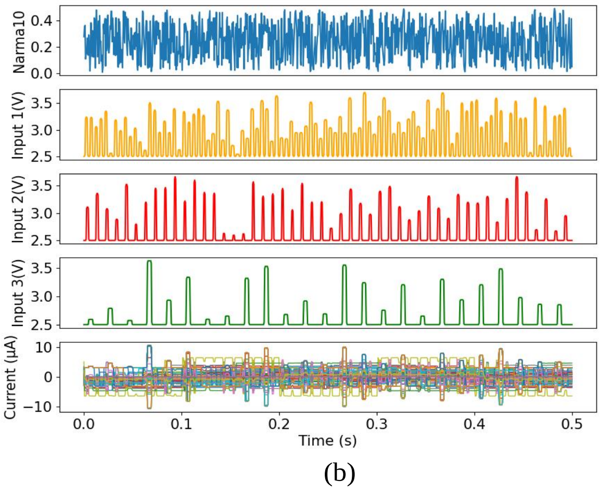


Fig. 7. Example visualization of related signals of memristive reservoir circuit. (a) Wave generation task: Input sine wave, three target waves and current signals of memristive reservoir circuits; (b) Narma-10 system prediction task: Target series, three discretized input signal and current signals of memristive reservoir circuits.

signals are normalised as the voltage with amplitude (-1 to 1). The gestures in the dataset are then divided, where 600 selected samples for training and the remaining 300 samples for testing.

B. Experimental Setting

In our experiments, Root Mean Squared Error (RMSE) is used as a measure of predictive performance:

$$RMSE = \sqrt{\left\langle \|\hat{\mathbf{y}}(t) - \mathbf{y}(t)\|^2 \right\rangle}, \tag{14}$$

where $\mathbf{y}(t)$ is the desired output (target), $\hat{\mathbf{y}}(t)$ is the readout output, $\|.\|$ denotes the Euclidean norm, and $\langle . \rangle$ denotes the empirical mean.

With the objective of evaluating the predictive performance of the proposed approach, we conduct comparisons with several existing baseline approaches for time series prediction, which are vanilla RNN [43], ESN [4], vanilla LSTM [44], memory-augmented LSMT and memory-augmented RNN [45]. These approaches have been chosen as baseline for time series information processing widely [45] [46].

We have applied not only grid search, but also differential evolution, which is a classic optimization method for searching optimal parameters [47], [48], to optimize the hyperparameters of the compared models. The parameters of the differential evolution algorithm are maxIter=200, pop_Size=20, mutation_Rate=(0.5, 1), and recombination_Rate=0.7. As for the parameters of our work, P_c represents the crossover possibility and sets it as 0.5. The mutation possibility P_m for all five types of mutation operators is set as 0.8. The num_ini_node and num_max_node represent the number of initial nodes in the memristive reservoir and the number of the max nodes allowed in the memristive reservoir. ε indicates the sparsity used in Algorithm 2 *Adapt()*, and is set as 0.3. The parameter A used in Equation (10) is set to 10000.

In order to prevent the issue of data leakage, time series cross-validation (3 folds) is applied to both of our proposed method and other SOTA models. The training of the models is done based on the training folds, whereas the hyperparameter or topology tuning is based on the RMSE computed over the validation folds. This is in line with the use of cross-validation for model selection [49].

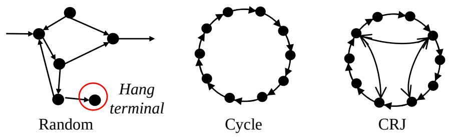


Fig. 8. Different topology of reservoir.

We have adopted Mann-Whitney U test as the statistical tests to support the comparison of the predictive performance of the approaches. The comparisons are based on 10 runs of the approaches.

Besides these software models, three memristive reservoir circuits with different fixed topologies have also been used to compare with our proposed evolutionary approach, which are random, cycle, and cycle with jumps, respectively. In order to ensure a fair comparison, the nodes applied in circuit with these topologies are set as the same num_max_node regarding with the different tasks, where 60 nodes for wave generation task and 30 nodes for six prediction tasks. Random topology represents that the memristors are connected randomly, cycle topology represents that the memristors are connected in series constructing a cycle, and cycle with jump represents that the memristors are connected in series constructing a cycle and some of them are connected with jumps directly. The diagram of different topologies of reservoir are shown in Fig. 8. As with our proposed approach, the output layer of the reservoirs with fixed topology is also trained based on Ridge regression.

The circuit setup is also required for evolving the memristive reservoir circuits. The circuit setup is different for the wave generation task and prediction tasks, which is mainly related to the embryo circuit design, will be introduced as follows:

- **Wave Generation Task:** The memristive reservoir will be evolved starting from an embryo circuit, which represents the initial circuit configuration of the to-be evolved circuit. The diagram of embryo circuit for the wave generation task is shown in Fig. 9 (a). The input signal is the sine wave with 1K HZ and 5V amplitude. This input terminal and GND terminal should be connected to the evolvable memristive reservoir circuit. In addition, the simulation time is set as 0.5s. After collecting the reservoir output, the data are fed into the readout function. Briefly, the target of this task is to generate three different waves based on one single input signal. Therefore, there is only one terminal for the input single in the embryo circuit.
- **Seven Other Tasks:** Besides the wave generation task, there are also seven other tasks (six are one-dimension and one is multi-dimension). These tasks require preprocessing the input signal $s\left(k\right)$ before feeding it into the memristive reservoir circuit. The amplitude of the input signal $s\left(k\right)$ is linearly converted into a voltage pulse with amplitude $V_{in}(k)$ that is then applied to the memristor reservoir:
$$V_{in}(k) = 2.5 \times s(k) + 2.5. \tag{15}$$
This linear conversion allows the input voltage pulses to fall in the range of $2.5–5V$ for memristor stimulation. In order to extract input features for the time series, there is another preprocessing step to the input signals before feeding into the memristive reservoir, which has been used in the preprocessing of the physical reservoir computing [12] [50]. Specifically, we fed three input signals with different sampling frequencies (200HZ, 100HZ and 50HZ) to the time series into the memristive reservoir circuits for one dimension. Therefore, there will three input voltages of the reservoir circuit for one-dimension tasks, and nine input voltages for the tasks with multi-dimension. Taking a prediction task as an example, its embryo circuit is shown in Fig. 9 (b). Moreover, GND terminal of the overall circuit are also set in the embryo circuit. After collecting the reservoir output, the data are fed into the readout function. Taking nonlinear dynamic series prediction task (Narma-10) as an example, Fig. 7(b) shows the time series of Narma-10 system, the input signals with different timeframes, and the current states recorded from the memristive reservoir circuit. We have also performed experiments which show that such preprocessing step is important to obtain better predictive performance, as shown in the supplementary material.

C. Experimental result

1) Ablation study of our proposed algorithm: In this section, we investigate the effectiveness of different operations in our proposed algorithm. Table III gives the comparison of our approach with or without crossover, *adapt*, and mutation operations. In general, removing crossover, *adapt* or mutation operations all degrade the RSME of the proposed algorithm. The Mann–Whitney U tests of the ablated approaches with all-equipped approach are conducted, and their p-values are also shown in Table IV.

From Table III, we can see that the reservoir with *adapt* outperforms the one without *adapt* and the memristive reservoir circuits with other topologies, implying that the *adapt* operation is capable of generating the effective connections for reservoir evolution.

Similarly, the effectiveness of crossover and mutation operations have also been investigated, where five types of mutation operations are all ablated in this experiment. As shown in Table III, after removing the crossover or mutations, the performance is worse than those of all-equipped one. In general, the results without crossover or mutations are worse than the results without *adapt*, which indicates that genetic operations among reservoirs is important for the reservoir evolution. Moreover, the evolved results are more sensitive to the ablation of mutations since there will be much reservoir diversity brought by the five types of mutation operations, which is beneficial to the reservoir evolution.

We also perform ablation experiments that remove the five types of mutation operations one by one. The results of ablating each specific operation are given in Table III. We can see that removing each individual mutation operation leads to performance degradation. According to the results shown in Table III, we can see that removing the add node mutation operation will lead to the most degradation of the performance, which indicates that the add node mutation operation plays the most important role within five mutation operations during the circuit evolution. The step mutation operation plays the second most important role to the circuit evolution. Other mutation operations also have impact on the circuit evolution.

TABLE II
PARAMETER SETTINGS OF OTHER OPTIMIZATION METHOD AND OURS

Models	Run time*	The parameters
DE	8.7s	maxIter=200; pop_Size=20; mutation_Rate=(0.5, 1); recombination_Rate=0.7
Our work	11.4s	num_Gen=200; num_Pop=20; num_Tour=3; P_m=0.8; P_c=0.5; num_ini_node=10; num_max_node=30 or 60; ε =0.3; A=10000

* Run time is the algorithm execution time of one generation with 20 populations. In order to give a fair comparison, both of our proposed method and other optimization method are conducted on the Intel(R) Xeon(R) W-2235 CPU.

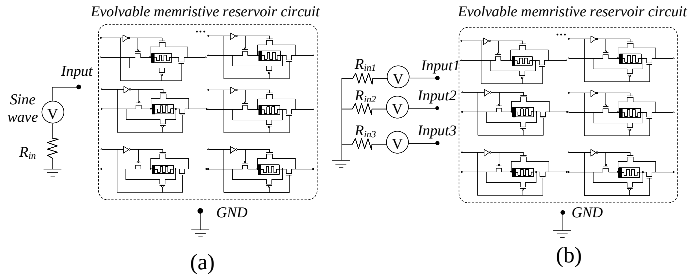


Fig. 9. The diagrams of embryo circuits for evolving memristive reservoir circuit to wave generation (a) 10th-order Narma dynamic system prediction (b).

TABLE III
EXPERIMENT RESULTS WITH DIFFERENT ALGORITHM COMPONENTS ABLATION OR DIFFERENT CIRCUIT CONDITIONS

	Wave	Narma-10	DJI	Audio	Tree	ARF	Santa	P-value
Performance comparisons with different ablated algorithm components								
No *Adapt*	0.0131	0.0329	0.1065	0.0890	0.0766	0.1283	0.1238	0.0226
No *Crossover*	0.1485	0.0378	0.1703	0.1648	0.0791	0.2736	0.1648	0.0075
No *Mutations*	0.2736	0.0472	0.1917	0.2011	0.0817	0.3008	0.1783	0.0074
Performance comparisons of removing the specific mutation operations								
No *Add node*	0.1029	0.0340	0.1733	0.0973	0.0860	0.2735	0.2834	0.0075
No *Delete node*	0.0834	0.0283	0.0784	0.0526	0.0839	0.1023	0.2719	0.0275
No *Step mutation*	0.1015	0.0312	0.1223	0.0918	0.0931	0.1980	0.1549	0.0076
No *GND/Input*	0.0235	0.0290	0.0881	0.0634	0.0920	0.1366	0.2920	0.0483
No *Weight mutation*	0.0917	0.0398	0.1336	0.0786	0.0432	0.2041	0.0853	0.0368
Performance comparisons under different circuit conditions								
With noise (σ=0.1)	0.0105	0.0277	0.0719	0.0533	0.0648	0.0778	0.0602	-
With noise (σ=0.2)	0.0124	0.0298	0.0826	0.0529	0.0649	0.0686	0.0678	-
With noise (σ=0.3)	0.0148	0.0303	0.0852	0.0685	0.0685	0.0881	0.0696	-
With noise (σ=0.4)	0.0245	0.0314	0.1097	0.0734	0.0788	0.1099	0.0737	-
All Equipped & No noise	0.0099	0.0239	0.0657	0.0493	0.0641	0.0743	0.0582	-

2) Comparisons of Memristive Reservoir Circuits under Different Circuit Conditions: First, we investigate the noise impact on the circuit perspective of memristive reservoir. Fig. 12 shows the current and memristor states of one memristive element based on our proposed reconfigurable memristive reservoir (Fig. 12 (a)) and the pure memristive reservoir (Fig. 12 (b)) under noise injection with different σ. As shown in Fig. 12, the memristor states should be set to start as 0.84. However, due to the noise injection, it will not be 0.84 accurately. In our proposed reconfigurable memristive reservoir, the current signal is not influenced so much by this noise. However, in a pure memristive reservoir like [18], the current signals are influenced heavier when there is noise injection to the reservoir. Therefore, compared with the pure memristors, our proposed memristor-based reconfigurable unit is more noise-tolerant to be configured as a reservoir computer since the current signal is not influenced so much by this noise.

Second, we investigate the impact of memristor variability on the performance. Table III gives the performance comparisons of our proposed memristive reservoir circuits under noise scenarios with different σ of Gaussian noise. In order to get a fair comparison, each result listed in the table is the average value under 10 runs in the corresponding σ and task. According to Table III, we can see that due to the noise injection to the memristors, the performance degrades. With increasing σ, there is an increasingly negative influence on the predictive performance. Even though there is predictive performance degradation when the noise is injected, the reservoir is still valid, i.e., it can still work as a reservoir computer.

3) Comparisons of Memristive Reservoir Circuits With Other Models: In this section, we discuss the regression performance of the memristive reservoir circuits with different topologies, which are random, cycle, cycle with jump and evolved topologies by our proposed approach. Their regression performance comparisons are shown in Table IV.

In terms of circuit feasibility, the circuits with random topology cannot ensure the feasibility since they may generate the dangling terminals of the circuit, leading to the failure of the circuit simulation. Moreover, our proposed memristive reservoirs can be evolved adaptively for different tasks, and cylce and CRJ memristive reservoirs cannot be evolved directly.

Considering the scalability of the circuit, we only use 30 nodes to construct the memristive reservoir circuit. As for the memristive reservoir circuit, CRJ memristive reservoir has the better performance over the cycle memristive reservoir. And the memristive reservoir evolved by our proposed approach outperforms over both of the cycle and CRJ memristive reservoir circuits.

Moreover, in order to further verify the performance of our proposed memristive reservoir circuits, several baseline approaches of RC are compared as reference, which are vanilla RNN [43], ESN [4], vanilla LSTM [44], memory-augmented LSMT and memory-augmented RNN [45]. As for the latter two, another parameter D is introduced for the memory augmentation, therefore, memory-augmented LSMT and memory-augmented RNN will be abbreviated as mLSTM and mRNN in the following Sections. In order to make a fair comparisons with other approaches, the evaluation number of the each algorithm is fixed, which is set as 4000, then we compare the predictive performance (RMSE and accuracy for DGR) of the reservoir on the 4000-th evaluation. Table IV shows the comparisons of different optimization methods applied to the reservoir, where the gird search, differential evolution, manual optimization are applied to be compared with our proposed method. According to Table IV, with the fixed computational budget (4000 evaluations), our proposed reservoir outperforms other baseline models in terms of the average ranking on different tasks. The Mann–Whitney U tests of the existing models with our proposed method are conducted, and their p-values are shown in Table IV. Based on a level of significance of 0.05, we can confirm that our proposed evolvable memristive reservoir circuit can improve the performance compared with the existing models.

We visualize the memristive reservoir topology and their corresponding circuits of the evolved results in Fig. 13. The visualization of the matrix indicates the result of multiplying W_{bool} and W_{res}. Specifically, the darker orange indicates larger pulse width of control signal V_c, and lighter orange indicates shorter pulse width. In the corresponding circuits, the input signals are connected to the evolved reservoir circuit and one of the nodes is connected to the GND.

According to the working principle of RMU introduced in Section III-C (an example of Jazz task is shown in Fig. 10), the larger pulse width of control signal V_c will lead to the longer memory dependency of input signal for the reservoir states, while the shorter pulse width of control signal V_c will lead to more frequent memory fading for the reservoir states. As shown in Fig. 13, the connections with the long dependency memory (darker color cubes) are more likely to appear nearby the terminals of the input signals, which is beneficial to spread the useful information across the whole reservoir.

Moreover, jump connections with long dependency memory (darker color cubes) tend to be generated to the reservoir circuits for solving the tasks with long-term memory, such as tree ring, DJI, and nonlinear audio. It indicates that these jump connections with long dependency memory are beneficial to tackle the long-memory tasks [45]. And, the connections with short dependency memory (lighter color cubes) tend to be generated to the reservoir circuits for solving the tasks with

TABLE IV
COMPARISONS OF MEMRISTIVE RESERVOIR CIRCUITS WITH DIFFERENT TOPOLOIES

	Implemen.	Cir. feas.	Opti. Method	Evo.?	Wave.* (#Node=60)	DGR	Nar. 10	DJI	Audio	Tree	ARF	Santa	Ave. Ran*.
Baseline approaches of RC													
RNN [43]-GS	Software	-	Grid search	No	0.0158 (9)	0.9494 (5)	0.0448 (9)	0.2605 (11)	0.0277 (4)	0.2871 (13)	1.1620 (11)	0.0398 (2)	13
ESN [4]-GS	Software	-	Grid search	No	0.0351 (10)	0.8300 (11)	0.0773 (11)	0.1357 (3)	0.1321 (11)	0.2192 (5)	1.5120 (12)	0.0612 (8)	11
LSTM [44]-GS	Software	-	Grid search	No	0.0129 (5)	0.8566 (8)	0.0415 (7)	0.2492 (8)	0.0393 (8)	0.2833 (11)	1.1340 (9)	0.0676 (9)	12
mLSTM [45]-GS	Software	-	Grid search	No	0.0118 (4)	0.8466 (9)	0.0506 (10)	0.2531 (8)	0.0231 (2)	0.2859 (12)	1.1490 (9)	0.0532 (3)	9
mRNN [45]-GS	Software	-	Grid search	No	0.0144 (6)	0.9767 (3)	0.0219 (1)	0.2487 (7)	0.0543 (11)	0.2818 (10)	1.0880 (7)	0.0648 (7)	10
RNN [43]-DE	Software	-	Differential evolution	No	0.0138 (4)	0.9567 (4)	0.0325 (4)	0.2256 (6)	0.0241 (3)	0.2735 (8)	1.0781 (6)	0.0376 (1)	4
ESN[4]-DE	Software	-	Differential evolution	No	0.0287 (5)	0.8433 (10)	0.0413 (6)	0.1219 (2)	0.0725 (10)	0.2014 (4)	1.3978 (7)	0.0543 (3)	4
LSTM[44]-DE	Software	-	Differential evolution	No	0.0116 (3)	0.9066 (6)	0.0401 (5)	0.2033 (2)	0.0373 (7)	0.2758 (9)	1.1270 (6)	0.0539 (2)	6
mLSTM [45]-DE	Software	-	Differential evolution	No	0.0104 (2)	0.8800 (7)	0.0432 (8)	0.2146 (2)	0.0211 (1)	0.2632 (6)	1.0375 (5)	0.0449 (1)	2
mRNN [45]-DE	Software	-	Differential evolution	No	0.0139 (2)	0.9833 (2)	0.0273 (3)	0.2234 (2)	0.0456 (9)	0.2707 (7)	1.0249 (4)	0.0572 (1)	3
Memristive reservoir circuit													
Random	IC	No	Manually	No	-	-	-	-	-	-	-	-	-
Cycle	IC	Yes	Manually	No	0.4772 (3)	0.7516 (13)	0.1943 (13)	0.2735 (3)	0.1569 (13)	0.1971 (3)	0.4365 (3)	0.4256 (3)	8
CRJ	IC	Yes	Manually	No	0.3551 (2)	0.7519 (12)	0.1387 (12)	0.2634 (2)	0.1498 (12)	0.0961 (2)	0.2114 (2)	0.1993 (2)	6
Ours	IC	Yes	Our proposed	Yes	0.0099 (1)	0.9892(1)	0.0239 (2)	0.0657 (1)	0.0493 (10)	0.0641 (1)	0.0743 (1)	0.0582 (1)	1
P-value	Ours VS CRJ: 0.00253; Ours VS Cycle: 0.00253; Ours VS RNN: 0.0483; Ours VS ESN:0.0075; Ours VS LSTM:0.0483; Ours VS mLST:0.0439; Ours VS mRNN:0.0402												

* Ave. Ran. represents the average value of the model's ranking in different datasets, where the ranking for one dataset is recorded first and then the average ranking in different datasets will be further given to indicate the average performance of the models.
* As for the wave generation and DGR tasks, 60 nodes are applied. As for the left prediction tasks, 30 nodes are applied.

TABLE V
COMPARISON BETWEEN OUR PROPOSED METHOD AND OTHER EXISTING PHYSICAL RESERVOIRS

Works	Design method	Hardware characteristics		Reservoir characteristics	
		Implementation	On-chip evolvable?	If reservoir changes adaptively to different tasks?	If reservoir changes dynamically during circuit execution?
[20]	Manually design	Sepcified memristive circuit	No	No	No
[18]	Manually design	Sepcified memristive circuit	No	No	No
[51]	Manually design	Sepcified atomic switch circuit	No	No	No
Our work	**Evolutionary approach**	**Reconfigurable memristive circuit**	**Yes**	**Yes**	**Yes**

TABLE VI
COMPARISON OF DIFFERENT HARDWARE RESERVOIRS

Methods	Implement.	Components used to construct a reservoir		
		#Memristor*	#MosFets	#Capacitor
FPGA-based ESN [10]	FPGA	0	37n*	n
Mosfet crossbar-based ESN [8]	PCB	0	16n	0
CMOS-based LSM [52]	IC	0	24n	3n
Ours	IC	1n	4n	0

* n represents the number of neurons existing in a reservoir.
* #Memristor, #Mosfet and #Capacitor refer to the number of memoristor, Mosfets and capacitors.

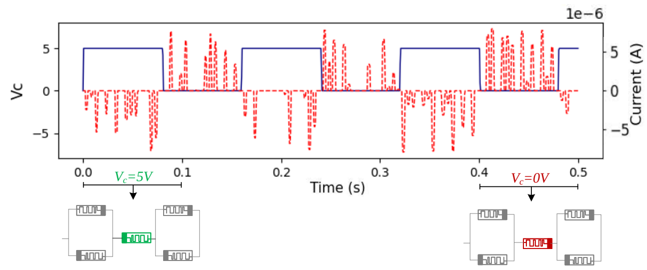


Fig. 10. The current and control signal visualization of one RMU under the task of audio, and its equivalent circuit state under different V_C states.

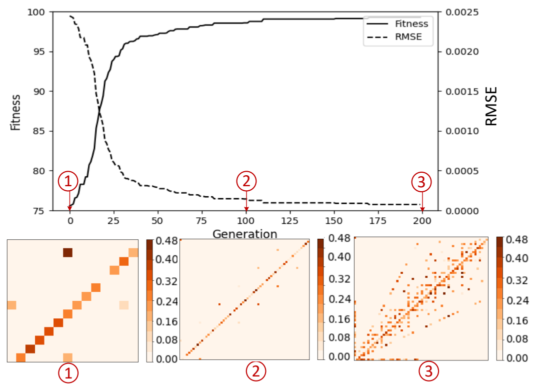


Fig. 11. Visualization of the network structure changes with the generation and its corresponding fitness results

short-term memory, such as Narma 10.

We can also see from Fig. 13 that the evolved reservoir circuit are based on cycle connections with irregular jumps between nodes and various memory dependency, which is difficult to be designed manually. These evolved cycle-based irregular jump reservoirs are achieved especially thanks to two components of our evolutionary process, namely our proposed sparse initialization and mutation operations. The sparse initialization will lead to not only the sparse connection but also the circuit validity. And the mutation operations will enable varieties of reservoir connections and memory dependencies of reservoir states.

Fig. 14 shows two examples of superposition between the actual outputs of our proposed memristive reservoir vs corresponding targets. They show that the signal generated by our proposed memristive reservoir circuit is mimicking the desired signal well. The results of other tasks are given in supplementary material.

4) Circuit Performance Analysis: Table V lists the comparison of existing methods with our proposed method. We have compared our proposed work with the existing methods [20] and [51], [18] by multiple perspectives, which are design method, hardware characteristics, and reservoir characteristics.

Regarding the design method, the reservoir of other ex-

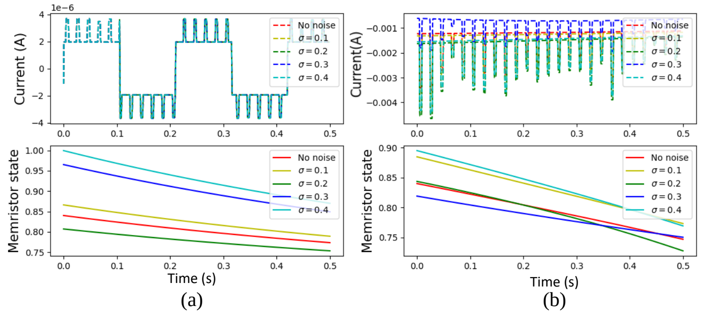


Fig. 12. (a) The visualization of the current and memristior state of one memristive element based on our proposed reconfigurable memristive reservoir. (b) The visualization of the current and memristior state of one memristive element based on pure memristive reservoir.

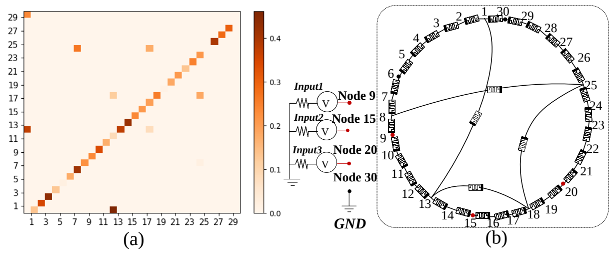


Fig. 13. Visualization of memristive reservoir topology and equivalent circuits. Taking the evolved result of Narma 10 as an example, and evolved results of other tasks are provided in the supplementary material.

isting work [20] [18] [51] are designed manually, which is an intensive task due to the large search space and high requirement to the designers. However, our work proposes an evolutionary approach that can automate the design and optimization procedure of the memristive reservoir.

Regarding the hardware characteristics, other works [20], [51], [18] are based on their specified memristive or atomic switch circuits, which cannot be reconfigured to change their topologies. Our work is based on our proposed memristive reconfigureable circuit. By changing the configuration signals, the circuit topology can be changed and further evolved by our proposed evolutionary algorithm.

Regarding the reservoir characteristics, because of the reconfigurable circuits, our proposed memrsitive reservoir can be evolved by searching for the optimal reservoir architecture for different tasks, leading to better predictive performance. In contrast, the existing methods [20], [51], [18] are based on a fixed circuit architecture that cannot be reconfigured, which can limit their predictive performance. Moreover, for other work [20], [51], [18], their circuits will be fixed not only for different tasks, but also within the circuit execution time. For our work, the circuit states of one Reconfigurable Memristive Unit (RMU) can also change within one cycle of circuit execution. Fig. 10 illustrates the current and control signal V_c of one RMU, and the equivalent circuit state under different V_c state. The dynamic changes of our proposed reservoir enrich the reservoir dynamics, which is beneficial to tackle the time series problems that involve long and short-term dependency.

Table VI shows the comparisons of different hardware reservoirs. We can see that ESN or LSM models have to use n

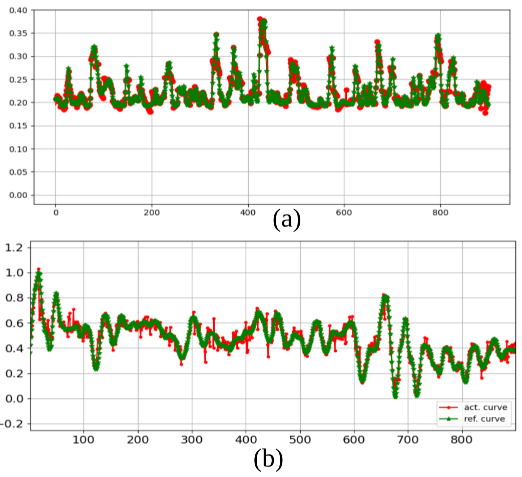


Fig. 14. Actual outputs of our proposed memristive reservoir vs corresponding targets. (a) Narma-10; (b) Nonlinear audio, and other tasks' are provided in the supplementary material.

neurons to construct the reservoir, however, one single neuron always needs several Mosfets to mimic nonlinear behavior between the input and output signal. Taking FPGA-based ESN [10] as an example, 37 Mosfets and 1 capacitor are required to construct one neuron, and there may be more than 60 neurons used to construct a reservoir in their work, so that 2368 Mosfets and 64 capacitors will be existed in its circuit counterpart. Compared with these work [10], [8], [52], by manipulating the currents flowing across the memristors, our approach only applied 4 Mosfets and 1 memristor to generate varied nonlinear behavior, which alleviates the problem of circuit scalability. This is because, by benefiting from the various nonlinear behaviors generated from RMUs, we were able to use a small number of 30 nodes for realizing the function of reservoir computing.

Moreover, we also analyzed the system performance improvement as the generations of the algorithm proceed. Taking the wave generation task as an example, Fig. 11 shows the fitness (solid line) and predictive performance (dot line) curves as the generations proceed. We also monitor the best individuals in three generations, namely the 1st generation, the 100-th generation and the 200-th generation. We display them by the matrix of the multiplication between W_{bool} and W_{res}. As shown in Fig. 11, the fitness (solid line) was improved from 75.5 to 99.2 by our proposed evolutionary algorithm. Regarding the best individual of the 1-st generation, it is initialized by a sparse topology (CRJ) and the initial number of the nodes is the minimum number. In Fig. 11 (bottom), the darker orange indicates larger pulse width of control signal V_c passing through a given node, and lighter orange indicates shorter pulse width passing through a given node. As we can see, lighter orange dominates the matrix corresponding to the 1-st generation, representing the CRJ sparse topology. With the evolution procedure (1-st generation to 100-th generation), the number of reservoir nodes increases while the topology still keeps the CRJ structure, with more connections evolved. By optimizing the topology and weights, the fitness can be improved a lot compared with the initial one. Through further evolution (100-th generation to 200-th generation), more connections that are beneficial to improve fitness are evolved, while the topology remains CRJ.

In summary, our proposed memristive reservoir circuit outperforms the SOTA approaches of RC, RNN, ESN, LSTM, mLSTM, and mRNN. Furthermore, when compared to other existing memristive reservoir circuits with fixed topologies, our memristive reservoir circuits not only ensured circuit feasibility but also achieved superior regression performance. From a circuit metric standpoint, our suggested reservoir circuit is made up of memristors and MosFests, which are more compact in terms of component count than other circuit implementations.

VI. Conclusion

In this paper, a scalable evolutionary algorithm for evolving the reconfigurable memristive reservoir circuits is proposed. Based on the reconfigurable memrsitve units, we design a memristive reservoir circuit that can be evolved on-chip in a reconfigurable way. Avoiding the variances driven by the differences between the actual and ideal memristors, the configuration signals of memristor are evolved directly and the current states flowing through the memristors are recorded as reservoir states. In addition, the feasibility and scalability of circuits are taken into the consideration in the proposed algorithm, where sparse initialization and several evolutionary operators are designed for the memristive reservoir circuits, alleviating these two issues. To validate our proposed approach, one generation task and six prediction tasks were applied. As shown experimentally, the memristive reservoir circuit evolved by our proposed algorithm can obtain better regression performance over the other memristive reservoir with a fixed topology and baseline approaches in terms of the average ranking.

Due to the nonlinear dynamics of memristor, our evolved memristive reservoir applies Mosfets and memristors to construct the nonlinear behavior of reservoir, incurring less number of Mosfets and no capacitor in the circuit.

Future work will focus on further improving both evolution efficiency and the result performance. Furthermore, we will also investigate the hardware implementation of the memristive output layer to achieve a fully analog adaptive memristive reservoir circuit. The evolutionary approach proposed in this paper could serve as the first step towards a fully evolvable hardware [53] [54] based on memristive circuits.

Acknowledgments

This work was support by the Research Institute of Trustworthy Autonomous Systems (RITAS), the Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), the Shenzhen Science and Technology Program (Grant No. KQTD2016112514355531).

References

[1] P. Verzelli, C. Alippi, L. Livi, and P. Tiňo, "Input-to-state representation in linear reservoirs dynamics," *IEEE Trans. Neural Netw. Learn. Syst.*, 2021.

[2] H. Chen, P. Tiňo, A. Rodan, and X. Yao, "Learning in the model space for cognitive fault diagnosis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 1, pp. 124–136, 2013.

[3] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Netw.*, vol. 115, pp. 100–123, 2019.

[4] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *German Nat. Res. Cntr. Inf. Technol., GMD Report*, vol. 148, no. 34, p. 13, 2001.

[5] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Comput.*, vol. 14, no. 11, pp. 2531–2560, 2002.

[6] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[7] H. O. Sillin, R. Aguilera, H.-H. Shieh, A. V. Avizienis, M. Aono, A. Z. Stieg, and J. K. Gimzewski, "A theoretical and experimental study of neuromorphic atomic switch networks for reservoir computing," *Nanotechnology*, vol. 24, no. 38, p. 384004, 2013.

[8] Y. Kume, S. Bian, and T. Sato, "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)*. IEEE, 2020, pp. 458–463.

[9] P. R. Prucnal, B. J. Shastri, T. F. de Lima, M. A. Nahmias, and A. N. Tait, "Recent progress in semiconductor excitable lasers for photonic spike processing," *Advances in Optics and Photonics*, vol. 8, no. 2, pp. 228–299, 2016.

[10] Y. Yi, Y. Liao, B. Wang, X. Fu, F. Shen, H. Hou, and L. Liu, "Fpga based spike-time dependent encoder and reservoir design in neuromorphic computing processors," *Microprocess. Microsyst.*, vol. 46, pp. 175–183, 2016.

[11] X. Zhu, Q. Wang, and W. D. Lu, "Memristor networks for real-time neural activity analysis," *Nat. Commun.*, vol. 11, no. 1, pp. 1–9, 2020.

[12] C. Du, F. Cai, M. A. Zidan, W. Ma, S. H. Lee, and W. D. Lu, "Reservoir computing using dynamic memristors for temporal information processing," *Nat. Commun.*, vol. 8, no. 1, pp. 1–10, 2017.

[13] L. Chua, "Memristor-the missing circuit element," *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[14] X. Shi, Z. Zeng, L. Yang, and Y. Huang, "Memristor-based circuit design for neuron with homeostatic plasticity," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 5, pp. 359–370, 2018.

[15] Q. Hong, H. Chen, J. Sun, and C. Wang, "Memristive circuit implementation of a self-repairing network based on biological astrocytes in robot application," *IEEE Trans. Neural Netw. Learn. Syst.*, 2020.

[16] S. Wen, R. Hu, Y. Yang, T. Huang, Z. Zeng, and Y.-D. Song, "Memristor-based echo state network with online least mean square," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 9, pp. 1787–1796, 2018.

[17] Q. Wang, Y. Li, and P. Li, "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)*, Montréal, QC, Canada, 2016, pp. 361–364.

[18] M. S. Kulkarni and C. Teuscher, "Memristor-based reservoir computing," in *Proc. IEEE/ACM Int. Symp. Nano. Archit. (NANOARCH)*. IEEE, 2012, pp. 226–232.

[19] G. Tanaka, R. Nakane, T. Yamane, S. Takeda, D. Nakano, S. Nakagawa, and A. Hirose, "Waveform classification by memristive reservoir computing," in *Proc. 24th Int. Conf. Neur. Inf. Process.* Springer, 2017, pp. 457–465.

[20] G. Tanaka and R. Nakane, "Simulation platform for pattern recognition based on reservoir computing with memristor networks," *Sci. Rep.*, vol. 12, no. 1, pp. 1–13, 2022.

[21] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Sci. Rev.*, vol. 3, no. 3, pp. 127–149, 2009.

[22] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez, "Training recurrent networks by evolino," *Neural Comput.*, vol. 19, no. 3, pp. 757–779, 2007.

[23] B. Schrauwen, D. Verstraeten, and J. M. V. Campenhout, "An overview of reservoir computing: theory, applications and implementations," in *Proc. 15th Eur. Symp. on Artif. Neur. Netw.*, 2007.

[24] K. C. Chatzidimitriou and P. A. Mitkas, "Adaptive reservoir computing through evolution and learning," *Neurocomputing*, vol. 103, pp. 198–209, 2013.

[25] M. Dale, J. F. Miller, S. Stepney, and M. A. Trefzer, "Evolving carbon nanotube reservoir computers," in *Proc. Int. Conf. Unconv. Comput. Nat. Comput. (UCNC)*. Springer, 2016, pp. 49–61.

[26] F. Duport, A. Akrout, A. Smerieri, M. Haelterman, and S. Massar, "Analog input layer for optical reservoir computers," *arXiv preprint arXiv:1406.3238*, 2014.

[27] M. L. Alomar, V. Canals, N. Perez-Mora, V. Martínez-Moll, and J. L. Rosselló, "Fpga-based stochastic echo state networks for time-series forecasting," *Comput. Intell. Neurosci.*, vol. 2016, 2016.

[28] P. Amil, C. Cabeza, and A. C. Marti, "Exact discrete-time implementation of the mackey–glass delayed model," *IEEE Trans. Circuits Syst. II, Exp. Brief*, vol. 62, no. 7, pp. 681–685, 2015.

[29] K. Bai and Y. Yi, "Dfr: An energy-efficient analog delay feedback reservoir computing system for brain-inspired computing," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 4, pp. 1–22, 2018.

[30] L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer, "Information processing using a single dynamical node as complex system," *Nature Commu.*, vol. 2, no. 1, pp. 1–6, 2011.

[31] A. Bala, I. Ismail, R. Ibrahim, and S. M. Sait, "Applications of metaheuristics in reservoir computing techniques: a review," *IEEE Access*, vol. 6, pp. 58 012–58 029, 2018.

[32] N. Chouikhi, B. Ammar, N. Rokbani, and A. M. Alimi, "Pso-based analysis of echo state network parameters for time series forecasting," *Appl. Soft Comput.*, vol. 55, pp. 211–225, 2017.

[33] T. van der Zant, V. Bečanović, K. Ishii, H.-U. Kobialka, and P. Ploeger, "Finding good echo state networks to control an underwater robot using evolutionary computations," *IFAC Proceedings Volumes*, vol. 37, no. 8, pp. 215–220, 2004.

[34] X. Wang, Y. Jin, and K. Hao, "Evolving local plasticity rules for synergistic learning in echo state networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1363–1374, 2019.

[35] A. A. Ferreira, T. B. Ludermir, and R. R. De Aquino, "An approach to reservoir computing design and training," *Expert Syst. Appl.*, vol. 40, no. 10, pp. 4172–4182, 2013.

[36] L. Chen, C. Li, T. Huang, X. Hu, and Y. Chen, "The bipolar and unipolar reversible behavior on the forgetting memristor model," *Neurocomputing*, vol. 171, pp. 1637–1643, 2016.

[37] L. Yang, Z. Zeng, and X. Shi, "A memristor-based neural network circuit with synchronous weight adjustment," *Neurocomputing*, vol. 363, pp. 114–124, 2019.

[38] A. Rodan and P. Tiňo, "Simple deterministically constructed cycle reservoirs with regular jumps," *Neural Comput.*, vol. 24, no. 7, pp. 1822–1852, 2012.

[39] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nat. Commun.*, vol. 9, no. 1, pp. 1–12, 2018.

[40] A. G. Parlos, O. T. Rais, and A. F. Atiya, "Multi-step-ahead prediction using dynamic recurrent neural networks," *Neural Netw.*, vol. 13, no. 7, pp. 765–786, 2000.

[41] A. Greaves-Tunnell and Z. Harchaoui, "A statistical investigation of long memory in language and music," in *Proc. 36th Int. Conf.Mach. Learn.*, Long Beach, California, USA, 2019, pp. 2394–2403.

[42] G. Holzmann, "Reservoir computing: a powerful black-box framework for nonlinear audio processing," in *Proc. 12th Int. Conf. Digit. Audio Eff. (DAFx)*, Como, Italy, 2009.

[43] C. L. Giles, G. M. Kuhn, and R. J. Williams, "Dynamic recurrent neural networks: Theory and applications," *IEEE Transs Neural Netws*, vol. 5, no. 2, pp. 153–156, 1994.

[44] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[45] J. Zhao, F. Huang, J. Lv, Y. Duan, Z. Qin, G. Li, and G. Tian, "Do rnn and lstm have long memory?" in *Proc. 37th Int. Conf. Mach. Learn.*, Virtual Event, 2020, pp. 11 365–11 375.

[46] A. Rodan and P. Tino, "Minimum complexity echo state network," *IEEE Trans. Neural Netw.*, vol. 22, no. 1, pp. 131–144, 2010.

[47] L. Wang, H. Hu, X.-Y. Ai, and H. Liu, "Effective electricity energy consumption forecasting using echo state network improved by differential evolution algorithm," *Energy*, vol. 153, pp. 801–815, 2018.

[48] B. Subudhi and D. Jena, "A differential evolution based neural network approach to nonlinear system identification," *Applied Soft Computing*, vol. 11, no. 1, pp. 861–871, 2011.

[49] J. Wainer and G. Cawley, "Nested cross-validation when selecting classifiers is overzealous for most practical applications," *Expert Systems with Applications*, vol. 182, p. 115222, 2021.

[50] W. Du, C. Li, Y. Huang, J. Zou, L. Luo, C. Teng, H.-C. Kuo, J. Wu, and Z. Wang, "An optoelectronic reservoir computing for temporal information processing," *IEEE Electron Device Letters*, vol. 43, no. 3, pp. 406–409, 2022.

[51] S. Lilak, W. Woods, K. Scharnhorst, C. Dunham, C. Teuscher, A. Z. Stieg, and J. K. Gimzewski, "Spoken digit classification by in-materio reservoir computing with neuromorphic atomic switch networks," *Frontiers in Nanotechnology*, p. 38, 2021.

[52] S. Roy, A. Banerjee, and A. Basu, "Liquid state machine with dendritically enhanced readout for low-power, neuromorphic vlsi implementations," *IEEE Trans. Biomed. Circuits Syst.*, vol. 8, no. 5, pp. 681–695, 2014.

[53] X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware," *IEEE Trans. Syst. Man Cybern. Syst. C*, vol. 29, no. 1, pp. 87–97, 1999.

[54] T. Higuchi, Y. Liu, and X. Yao, *Evolvable Hardware (Genetic and Evolutionary Computation)*. Berlin, Heidelberg: Springer-Verlag, 2006.